```fortran
      program Noah_driver

      use module_Noahlsm_utility
      use module_sf_noahlsm_param_init
      use module_date_utilities
      use module_sf_Noahlsm_gridded_input
      use module_sf_noahlsm
      use module_sf_Noah_NC_output

      implicit none

      character(len=6)                     :: mdt          !model start month, day, and hour
      character(len=2)                     :: minute       !model start minute
      character(len=4)                     :: year         !model start year

      integer                              :: imonth       !only used to compute cosz
      integer                              :: iday         !only used to compute cosz
      integer                              :: itime        !only used to compute cosz

      integer                              :: istep        !counting records in forcing data
      integer                              :: imstep       !for monthly output
      integer                              :: ND           !for monthly output
      integer                              :: idstep       !for 3-hourly output
      integer                              :: startstep    !starting time step to read forcings
      integer                              :: iyloop       !year loop index
      integer                              :: imloop       !month loop index
      integer                              :: idloop       !day loop index
      integer                              :: ihloop       !hour loop index
      integer                              :: startday     !starting day
      integer                              :: startmonth   !strating month
      integer                              :: ierr         !error message index
      integer, dimension(0:11)             :: jday         !julian day
      integer, dimension(0:11)             :: gday         !gorgian day
      integer, dimension(1:12)             :: nday         !days per month

      data (nday(imonth),imonth=1,12) /31,  28,  31,  30,  31,  30,  31,  31,  30,  31,  30,  31/
      data (jday(imonth),imonth=0,11) / 0,  31,  59,  90,120,151,181,212,242,273,304,334/
      data (gday(imonth),imonth=0,11) / 0,  31,  60,  91,121,152,182,213,243,274,305,335/

      integer                              :: ix           !do loop index in x-direction
      integer                              :: iy           !do loop index in y-direction
      integer                              :: iz           !do loop index in z-direction
      integer                              :: it           !do loop index for time
      integer                              :: nsoil        !number of soil layers
      integer                              :: nx           !total x-direction points
      integer                              :: ny           !total y-direction points
      integer                              :: npoint       !number of land points
      integer                              :: isnow        !actual no. of snow layers
      integer, parameter                   :: nsnow = 3    !maximum no. of snow layers
      real, dimension(1:100)               :: zsoil        !depth of the bottom of soil layers(m)

      integer                              :: ispin        !index for sinp-up
      integer                              :: nspin        !total spin-up times

!grided variables

      integer, allocatable, dimension(:,:) :: vegtypxy     !vegetation type
      integer, allocatable, dimension(:,:) :: soiltypxy    !soil type
      integer, allocatable, dimension(:,:) :: sloptypxy    !sloptypxy (only used for Noah runoff scheme)
      integer, allocatable, dimension(:,:) :: landseaxy    !land-sea mask (0-ocean; 1-land)
      integer, allocatable, dimension(:,:) :: iscxy        !soil color index
      real, allocatable, dimension(:,:)    :: latxy        !latitude
      real, allocatable, dimension(:,:)    :: lonxy        !longitude
      real, allocatable, dimension(:,:)    :: tbotxy       !soil temperature at zbot
      real, allocatable, dimension(:,:)    :: laixy        !leaf area index
      real, allocatable, dimension(:,:,:)  :: shdfacxy     !greenness vegetation (shaded) fraction

      real, allocatable, dimension(:,:)    :: sfctmpxy     !surface air temperature
      real, allocatable, dimension(:,:)    :: q2xy         !surface air specific humidity
```

```
    real, allocatable, dimension(:,:)     :: lwdnxy      !downward longwave radiation
    real, allocatable, dimension(:,:)     :: uxy         !wind speed at x-direction
    real, allocatable, dimension(:,:)     :: vxy         !wind speed at y-direction
    real, allocatable, dimension(:,:)     :: sfcprsxy    !surfae air pressure
    real, allocatable, dimension(:,:)     :: soldnxy     !downward shortwave radiation
    real, allocatable, dimension(:,:)     :: prcpxy      !precipitation
    real, allocatable, dimension(:,:)     :: varin      !

    integer, allocatable, dimension(:,:) :: isnowxy     !actual no. of snow layers
    real, allocatable, dimension(:,:,:)   :: stcxy       !snow/soil tmperatures
    real, allocatable, dimension(:,:,:)   :: smcxy       !vol. soil moisture (m3/m3)
    real, allocatable, dimension(:,:,:)   :: sh2oxy      !vol. soil liquid water (m3/m3)
    real, allocatable, dimension(:,:)     :: tvxy        !vegetation canopy temperature
    real, allocatable, dimension(:,:)     :: tgxy        !ground surface temperature
    real, allocatable, dimension(:,:)     :: canicexy    !canopy-intercepted ice (mm)
    real, allocatable, dimension(:,:)     :: canliqxy    !canopy-intercepted liquid water (mm)
    real, allocatable, dimension(:,:)     :: snowhxy     !snow depth (m)
    real, allocatable, dimension(:,:)     :: sneqvxy     !snow water equivalent (mm)
    real, allocatable, dimension(:,:)     :: eahxy       !canopy air vapor pressure (pa)
    real, allocatable, dimension(:,:)     :: tahxy       !canopy air temperature (k)
    real, allocatable, dimension(:,:)     :: cmxy        !momentum drag coefficient
    real, allocatable, dimension(:,:)     :: chxy        !sensible heat exchange coefficient

    real, allocatable, dimension(:,:)     :: fwetxy      !wetted or snowed fraction of the canopy (-)
    real, allocatable, dimension(:,:)     :: sneqvoxy    !snow mass at last time step(mm h2o)
    real, allocatable, dimension(:,:)     :: alboldxy    !snow albedo at last time step (-)
    real, allocatable, dimension(:,:)     :: qsnowxy     !snowfall on the ground [mm/s]
    real, allocatable, dimension(:,:)     :: wslakexy    !lake water storage [mm]

    real, allocatable, dimension(:,:)     :: zwtxy       !water table depth [m]
    real, allocatable, dimension(:,:)     :: waxy        !water in the "aquifer" [mm]
    real, allocatable, dimension(:,:)     :: wtxy        !groundwater storage [mm]
    real, allocatable, dimension(:,:,:)   :: tsnoxy      !snow temperature [K]
    real, allocatable, dimension(:,:,:)   :: zsnsoxy     !snow layer depth [m]
    real, allocatable, dimension(:,:,:)   :: snicexy     !snow layer ice [mm]
    real, allocatable, dimension(:,:,:)   :: snliqxy     !snow layer liquid water [mm]
    REAL, allocatable, dimension(:,:)     :: lfmassxy    !leaf mass [g/m2]
    REAL, allocatable, dimension(:,:)     :: rtmassxy    !mass of fine roots [g/m2]
    REAL, allocatable, dimension(:,:)     :: stmassxy    !stem mass [g/m2]
    REAL, allocatable, dimension(:,:)     :: woodxy      !mass of wood (incl. woody roots) [g/m2]
    REAL, allocatable, dimension(:,:)     :: stblcpxy    !stable carbon in deep soil [g/m2]
    REAL, allocatable, dimension(:,:)     :: fastcpxy    !short-lived carbon, shallow soil [g/m2]
    REAL, allocatable, dimension(:,:)     :: xlaixy      !leaf area index
    REAL, allocatable, dimension(:,:)     :: xsaixy      !stem area index
    REAL, allocatable, dimension(:,:)     :: tradxy      !surface radiative temperature (k)
    REAL, allocatable, dimension(:,:)     :: tsxy        !surface temperature (k)
    REAL, allocatable, dimension(:,:)     :: neexy       !net ecosys exchange (g/m2/s CO2)
    REAL, allocatable, dimension(:,:)     :: gppxy       !gross primary assimilation [g/m2/s C]
    REAL, allocatable, dimension(:,:)     :: nppxy       !net primary productivity [g/m2/s C]
    REAL, allocatable, dimension(:,:)     :: fvegxy      !greenness vegetation fraction [-]

    real, allocatable, dimension(:,:)     :: qinxy       !groundwater recharge [mm/s]
    real, allocatable, dimension(:,:)     :: runsfxy     !surface runoff [mm/s]
    real, allocatable, dimension(:,:)     :: runsbxy     !subsurface runoff [mm/s]
    real, allocatable, dimension(:,:)     :: ecanxy      !evaporation of intercepted water (mm/s)
    real, allocatable, dimension(:,:)     :: edirxy      !soil surface evaporation rate (mm/s]
    real, allocatable, dimension(:,:)     :: etranxy     !transpiration rate (mm/s)

    real, allocatable, dimension(:,:)     :: fsaxy       !total absorbed solar radiation (w/m2)
    real, allocatable, dimension(:,:)     :: firaxy      !total net longwave rad (w/m2) [+ to atm]
    real, allocatable, dimension(:,:)     :: fshxy       !total sensible heat (w/m2) [+ to atm]
    real, allocatable, dimension(:,:)     :: flhxy       !total latent heat (w/m2) [+ to atm]
    real, allocatable, dimension(:,:)     :: fghxy       !ground heat flux (w/m2) [+ to soil]
    real, allocatable, dimension(:,:)     :: aparxy      !photosyn active energy by canopy (w/m2)
    real, allocatable, dimension(:,:)     :: psnxy       !total photosynthesis (umol co2/m2/s) [+]
    real, allocatable, dimension(:,:)     :: savxy       !solar rad absorbed by veg. (w/m2)
    real, allocatable, dimension(:,:)     :: sagxy       !solar rad absorbed by ground (w/m2)
    real, allocatable, dimension(:,:)     :: fsnoxy      !snow cover fraction (-)
```

```
! monthly mean variables

      real, allocatable, dimension(:,:)     :: snowhm      !snow depth (m)
      real, allocatable, dimension(:,:)     :: sneqvm      !snow water equivelant (mm or kg/m2)
      real, allocatable, dimension(:,:)     :: tgm         !ground surface temperature (K)
      real, allocatable, dimension(:,:)     :: prcpm       !precipitation (mm/s)
      real, allocatable, dimension(:,:)     :: runsfm      !surface runoff (mm/s)
      real, allocatable, dimension(:,:)     :: runsbm      !subsurface runoff (mm/s)
      real, allocatable, dimension(:,:)     :: ecanm       !evaporation of intercepted water (mm/s)
      real, allocatable, dimension(:,:)     :: edirm       !soil surface evaporation rate (mm/s]
      real, allocatable, dimension(:,:)     :: etranm      !transpiration rate (mm/s)
      real, allocatable, dimension(:,:)     :: zwtm        !the depth to water table (m)
      real, allocatable, dimension(:,:)     :: fsam        !total absorbed solar radiation (w/m2)
      real, allocatable, dimension(:,:)     :: firam       !total net longwave rad (w/m2) [+ to atm]
      real, allocatable, dimension(:,:)     :: fshm        !total sensible heat (w/m2) [+ to atm]
      real, allocatable, dimension(:,:)     :: flhm        !total latent heat (w/m2) [+ to atm]
      real, allocatable, dimension(:,:)     :: fghm        !ground heat flux (w/m2) [+ to soil]
      real, allocatable, dimension(:,:,:)   :: stcm        !soil temperature (K)
      real, allocatable, dimension(:,:,:)   :: smcm        !volumetric soil moisture (m3/m3)
      real, allocatable, dimension(:,:,:)   :: sh2om       !volumetric liquid water content in soil (m3/m3)
      real, allocatable, dimension(:,:)     :: aparm       !photosyn active energy by canopy (w/m2)
      real, allocatable, dimension(:,:)     :: psnm        !total photosynthesis (umol co2/m2/s) [+]
      real, allocatable, dimension(:,:)     :: savm        !solar rad absorbed by veg. (w/m2)
      real, allocatable, dimension(:,:)     :: sagm        !solar rad absorbed by ground (w/m2)
      real, allocatable, dimension(:,:)     :: fsnom       !snow cover fraction (-)
      real, allocatable, dimension(:,:)     :: xlaim       !leaf area index (-)
      real, allocatable, dimension(:,:)     :: xsaim       !stem area index (-)
      real, allocatable, dimension(:,:)     :: tradm       !surface radiative temperature (K)
      real, allocatable, dimension(:,:)     :: tsm         !surface temperature (K)
      real, allocatable, dimension(:,:)     :: neem        !net ecosystem exchange (g/m2/s CO2)
      real, allocatable, dimension(:,:)     :: gppm        !gross primary productivity [g/m2/s C]
      real, allocatable, dimension(:,:)     :: nppm        !net primary productivity [g/m2/s C]
      real, allocatable, dimension(:,:)     :: fvegm       !greenness vegetation fraction [-]
      real, allocatable, dimension(:,:)     :: cmm         !surface exchange coeffcient for momentum [-]
      real, allocatable, dimension(:,:)     :: chm         !surface exchange coeffcient for heat [-]

!----------------------------------------------------------------------
! 1-d variables
!----------------------------------------------------------------------
      integer                          :: ice      !sea-ice flag  (=1: sea-ice, =0: land)
      integer                          :: isc      !soil color index (1-lightest; 8-darkest)
      integer                          :: ipoint   !gridcell index
      integer                          :: ist      !surface type: 1->soil; 2-> lake
      real                             :: dt       !timestep (second)
      real                             :: zlvl     !height (m) above ground of atmos. forcing
      real                             :: lat      !latitude

      real                             :: lwdn     !longwave downward radiation (w/m2)
      real                             :: soldn    !solar downward radiation (wm-2)
      real                             :: sfcprs   !pressure (Pa) at height ZLVL m above ground
      real                             :: prcp     !precip rate (kg m-2 s-1) (NOTE: this is a rate)
      real                             :: sfctmp   !air temperature (K) at height ZLVL m above ground
      real                             :: q2       !mixing ratio (kg kg-1) at height ZLVL m above ground
      real                             :: uu       !wind speed in eastward dir (m/s)
      real                             :: vv       !wind speed in northward dir (m/s)
      real                             :: co2air   !atmospheric co2 concentration (pa)
      real                             :: o2air    !atmospheric o2 concentration (pa)
      real                             :: foln     !foliage nitrogen (%) (1.0 - saturated)

      integer                          :: vegtyp   !vegetation type (integer index)
      integer                          :: soiltyp  !soil type (integer index)
      integer                          :: sloptyp  !soil type (integer index)
      real                             :: tbot     !bottom soil temperature (yearly-mean surf air temp.)
      real                             :: shdfac   !greeness vegetation fraction (-)
      real                             :: cosz     !solar zenith angle
      real                             :: z0       !roughness length (m)

! intent(inout) to/from SFLX:
      real, allocatable, dimension(:) :: ficeold !snow layer liquid water [mm]
```

```
      real                         :: fwet    !wetted or snowed fraction of the canopy (-)
      real                         :: sneqvo  !snow mass at last time step(mm h2o)
      real                         :: albold  !snow albedo at last time step (-)
      real                         :: qsnow   !snowfall on the ground [mm/s]
      real                         :: eah     !canopy air vapor pressure (pa)
      real                         :: tah     !canopy air temperature (k)
      real                         :: cm      !momentum drag coefficient
      real                         :: ch      !sensible heat exchange coefficient

      real                         :: canliq  !intercepted liquid water (mm h2o)
      real                         :: canice  !intercepted ice mass (mm h2o)

      real                         :: tv      !vegetation temperature (kelvin)
      real                         :: tg      !ground temperature (k)
      real, allocatable, dimension(:) :: sldpth  !the thickness (m) of each soil layer
      real, allocatable, dimension(:) :: stc     !soil temp (K)
      real, allocatable, dimension(:) :: smc     !total soil moisture content (volumetric m3/m3)
      real, allocatable, dimension(:) :: sh2o    !unfrozen soil moisture content (volumetric m3/m3)
      real, allocatable, dimension(:) :: zsnso   !snow layer depth [m]
      real, allocatable, dimension(:) :: snice   !snow layer ice [mm]
      real, allocatable, dimension(:) :: snliq   !snow layer liquid water [mm]

      real                         :: snowh   !actual snow depth (m)
      real                         :: sneqv   !liquid water-equivalent snow depth (m)

      real                         :: zwt     !water table depth [m]
      real                         :: wa      !water in the "aquifer" (below the soil column) [mm]
      real                         :: wt      !groundwater storage [mm]
      real                         :: wslake  !lake water storage [mm]

      REAL                         :: lfmass  !leaf mass [g/m2]
      REAL                         :: rtmass  !mass of fine roots [g/m2]
      REAL                         :: stmass  !stem mass [g/m2]
      REAL                         :: wood    !mass of wood (incl. woody roots) [g/m2]
      REAL                         :: stblcp  !stable carbon in deep soil [g/m2]
      REAL                         :: fastcp  !short-lived carbon, shallow soil [g/m2]
      REAL                         :: xlai    !leaf area index
      REAL                         :: xsai    !stem area index
      REAL                         :: nee     !net ecosys exchange (g/m2/s CO2)
      REAL                         :: gpp     !net instantaneous assimilation [g/m2/s C]
      REAL                         :: npp     !net primary productivity [g/m2/s C]
      REAL                         :: fveg    !greenness vegetation fraction [-]

! intent(out) from SFLX:
      real                         :: fsa     !total absorbed solar radiation (w/m2)
      real                         :: fsr     !total reflected solar radiation (w/m2)
      real                         :: fira    !total net longwave rad (w/m2)   [+ to atm]
      real                         :: fsh     !total sensible heat (w/m2)      [+ to atm]
      real                         :: fcev    !canopy evaporation heat (w/m2) [+ to atm]
      real                         :: fgev    !ground evaporation heat (w/m2) [+ to atm]
      real                         :: fctr    !transpiration heat flux (w/m2) [+ to atm]
      real                         :: ssoil   !ground heat flux (w/m2) [+ to soil]
      REAL                         :: trad    !surface radiative temperature (k)
      REAL                         :: ts      !surface temperature (k)
      real                         :: ecan    !evaporation of intercepted water (mm/s) [+]
      real                         :: etran   !transpiration rate (mm/s) [+]
      real                         :: edir    !ground surface evaporation rate (mm/s) [+]
      real                         :: qin     !groundwater recharge [mm/s]  for output
      real                         :: runsf   !surface runoff [mm/s]
      real                         :: runsb   !subsurface runoff [mm/s]
      REAL                         :: psn     !total photosynthesis (umol co2/m2/s) [+]
      REAL                         :: apar    !photosyn active energy by canopy (w/m2)
      REAL                         :: sav     !solar rad absorbed by veg. (w/m2)
      REAL                         :: sag     !solar rad absorbed by ground (w/m2)
      REAL                         :: fsno    !snow cover fraction on the ground (-)

      real                         :: calday  !calendar day for computing solar zenith angle
!----------------------------------------------------------------------
!  declare/initialize constants
```

```fortran
!----------------------------------------------------------------------
      character(len=14)  :: olddate
      character(len=14)  :: newdate
      character(len=256) :: dir       !directory where input and output data files are
      character(len=256) :: fini      !directory where initial data files are
      character(len=8)   :: exp       !directory where an experiment is

      integer :: start_year,start_month,start_day,start_hour,start_min
      integer :: end_year
      namelist /noahlsm_offline/dir,fini     ,exp     ,nsoil    ,nx      ,ny       ,&
      npoint     ,zsoil   ,start_year,start_month,start_day,start_hour,start_min ,&
      end_year  ,dt      ,zlvl

      open(30,file="noah_offline.namelist",status='old',form="formatted")
      read(30,noahlsm_offline,iostat=ierr)
      if (ierr /= 0) then
         stop
      end if
      close(30)

      write(mdt,'(i2.2,i2.2,i2.2)') start_month, start_day, start_hour
      write(minute,'(i2.2)') start_min
      print*, 'mdt, minute = ', mdt, minute
      write(year, '(i4.4)') start_year

      write(olddate,'(I4.4,I2.2,I2.2,I2.2,I2.2,I2.2)') &
      start_year, start_month, start_day, start_hour, start_min, 0

!----------------------------------------------------------------------
      call lsm_parm_init
!----------------------------------------------------------------------
! Allocate arrays for our gridded domain, now that we know the size
!----------------------------------------------------------------------
! land surface parameters

      allocate( latxy    (nx,ny) )
      allocate( lonxy    (nx,ny) )
      allocate( vegtypxy (nx,ny) )
      allocate( soiltypxy(nx,ny) )
      allocate( sloptypxy(nx,ny) )
      allocate( tbotxy   (nx,ny) )
      allocate( landseaxy(nx,ny) )
      allocate( iscxy    (nx,ny) )
      allocate( laixy    (nx,ny) )
      allocate( shdfacxy (nx,ny,12) )

! forcing data

      allocate( sfctmpxy (nx,ny) )
      allocate( q2xy     (nx,ny) )
      allocate( lwdnxy   (nx,ny) )
      allocate( uxy      (nx,ny) )
      allocate( vxy      (nx,ny) )
      allocate( sfcprsxy (nx,ny) )
      allocate( soldnxy  (nx,ny) )
      allocate( prcpxy   (nx,ny) )
      allocate( varin  (npoint,8))

! prognostic varaibles
! 1-d
!      allocate( zsoil  (       1:nsoil) )
      allocate( sldpth (       1:nsoil) )
      allocate( sh2o   (       1:nsoil) )
      allocate( smc    (       1:nsoil) )
      allocate( stc    (-nsnow+1:nsoil) )
      allocate( zsnso  (-nsnow+1:nsoil) )
      allocate( snice  (-nsnow+1:     0) )
      allocate( snliq  (-nsnow+1:     0) )
      allocate( ficeold(-nsnow+1:     0) )
```

```
! 2-d prognostic variables
      allocate( smcxy    (nx,ny,       1:nsoil) ) ! 1
      allocate( stcxy    (nx,ny,       1:nsoil) ) ! 2
      allocate( sh2oxy   (nx,ny,       1:nsoil) ) ! 3
      allocate( tsnoxy   (nx,ny,-nsnow+1:    0) ) ! 4
      allocate( snicexy  (nx,ny,-nsnow+1:    0) ) ! 5
      allocate( snliqxy  (nx,ny,-nsnow+1:    0) ) ! 6
      allocate( zsnsoxy  (nx,ny,-nsnow+1:nsoil) ) ! 7
      allocate( tvxy     (nx,ny) )                ! 8
      allocate( tgxy     (nx,ny) )                ! 9
      allocate( canliqxy (nx,ny) )                !10
      allocate( canicexy (nx,ny) )                !11
      allocate( snowhxy  (nx,ny) )                !12
      allocate( sneqvxy  (nx,ny) )                !13
      allocate( zwtxy    (nx,ny) )                !14
      allocate( waxy     (nx,ny) )                !15
      allocate( wtxy     (nx,ny) )                !16
      allocate( isnowxy  (nx,ny) )                !17
      allocate( lfmassxy (nx,ny) )                !18
      allocate( rtmassxy (nx,ny) )                !19
      allocate( stmassxy (nx,ny) )                !20
      allocate( woodxy   (nx,ny) )                !21
      allocate( stblcpxy (nx,ny) )                !22
      allocate( fastcpxy (nx,ny) )                !23
      allocate( xlaixy   (nx,ny) )                !24
      allocate( xsaixy   (nx,ny) )                !25
      allocate( eahxy    (nx,ny) )                !26
      allocate( tahxy    (nx,ny) )                !27

      allocate( fwetxy   (nx,ny) )                !28
      allocate( sneqvoxy (nx,ny) )                !29
      allocate( alboldxy (nx,ny) )                !30
      allocate( qsnowxy  (nx,ny) )                !31
      allocate( wslakexy (nx,ny) )                !32

      allocate( cmxy     (nx,ny) )                !
      allocate( chxy     (nx,ny) )                !

! for output

      allocate( qinxy    (nx,ny) )
      allocate( runsfxy  (nx,ny) )
      allocate( runsbxy  (nx,ny) )
      allocate( ecanxy   (nx,ny) )
      allocate( edirxy   (nx,ny) )
      allocate( etranxy  (nx,ny) )
      allocate( fsaxy    (nx,ny) )
      allocate( firaxy   (nx,ny) )
      allocate( fshxy    (nx,ny) )
      allocate( flhxy    (nx,ny) )
      allocate( fghxy    (nx,ny) )
      allocate( aparxy   (nx,ny) )
      allocate( psnxy    (nx,ny) )
      allocate( savxy    (nx,ny) )
      allocate( sagxy    (nx,ny) )
      allocate( fsnoxy   (nx,ny) )
      allocate( tradxy   (nx,ny) )
      allocate( tsxy     (nx,ny) )
      allocate( neexy    (nx,ny) )
      allocate( gppxy    (nx,ny) )
      allocate( nppxy    (nx,ny) )
      allocate( fvegxy   (nx,ny) )

! monthly mean variables for output

      allocate(  snowhm  (nx,ny) )
      allocate(  sneqvm  (nx,ny) )
      allocate(  tgm     (nx,ny) )
```

```
      allocate(  prcpm   (nx,ny) )
      allocate(  runsfm  (nx,ny) )
      allocate(  runsbm  (nx,ny) )
      allocate(  ecanm   (nx,ny) )
      allocate(  edirm   (nx,ny) )
      allocate(  etranm  (nx,ny) )
      allocate(  zwtm    (nx,ny) )
      allocate(  fsam    (nx,ny) )
      allocate(  firam   (nx,ny) )
      allocate(  fshm    (nx,ny) )
      allocate(  flhm    (nx,ny) )
      allocate(  fghm    (nx,ny) )
      allocate(  stcm    (nx,ny,nsoil) )
      allocate(  smcm    (nx,ny,nsoil) )
      allocate(  sh2om   (nx,ny,nsoil) )
      allocate(  aparm   (nx,ny) )
      allocate(  psnm    (nx,ny) )
      allocate(  savm    (nx,ny) )
      allocate(  sagm    (nx,ny) )
      allocate(  fsnom   (nx,ny) )
      allocate(  xlaim   (nx,ny) )
      allocate(  xsaim   (nx,ny) )
      allocate(  tradm   (nx,ny) )
      allocate(  tsm     (nx,ny) )
      allocate(  neem    (nx,ny) )
      allocate(  gppm    (nx,ny) )
      allocate(  nppm    (nx,ny) )
      allocate(  fvegm   (nx,ny) )
      allocate(  cmm   (nx,ny) )
      allocate(  chm   (nx,ny) )
!----------------------------------------------------------------------
! Initialize gridded domain
!----------------------------------------------------------------------
! sldpth is the thickness of each layer

      sldpth(1) = -zsoil(1)                              ! can be sent to REDPRM
      do iz = 2, nsoil
         sldpth(iz) = zsoil(iz-1)-zsoil(iz)
      enddo

      zsnso   = -999.9
      snice   = -999.9
      snliq   = -999.9
      stc     = -999.9
      smc     = -999.9
      sh2o    = -999.9

      snowhm = -999.9
      sneqvm = -999.9
      tgm    = -999.9
      prcpm  = -999.9
      runsfm = -999.9
      runsbm = -999.9
      ecanm  = -999.9
      edirm  = -999.9
      etranm = -999.9
      zwtm   = -999.9
      fsam   = -999.9
      firam  = -999.9
      fshm   = -999.9
      flhm   = -999.9
      fghm   = -999.9
      stcm   = -999.9
      smcm   = -999.9
      sh2om  = -999.9
      aparm  = -999.9
      psnm   = -999.9
      savm   = -999.9
      sagm   = -999.9
```

```
      fsnom  = -999.9
      xlaim  = -999.9
      xsaim  = -999.9
      tradm  = -999.9
      tsm    = -999.9
      neem   = -999.9
      gppm   = -999.9
      nppm   = -999.9
      fvegm  = -999.9
      cmm  = -999.9
      chm  = -999.9
!----------------------------------------------------------------------
! read landuse type and soil texture and other information
!----------------------------------------------------------------------

      call readland(dir,nx,ny,vegtypxy,soiltypxy,sloptypxy,  &
                    latxy,lonxy,landseaxy,tbotxy,iscxy)

       foln   = 1.5        !saturated value
       z0     = 0.01       !ground roughness length

! green vegetation fraction

       call readveg(dir,nx,ny,landseaxy,shdfacxy)

!----------------------------------------------------------------------
! initialize model state
!----------------------------------------------------------------------

    call readinit(dir   ,nx      ,ny      ,nsoil   ,nsnow   ,fini    ,&
                soiltypxy,nsltype ,maxsmc  ,zsoil   ,alboldxy,         &
                fwetxy   ,sneqvoxy,qsnowxy ,wslakexy,eahxy   ,tahxy   ,&
                smcxy    ,stcxy   ,sh2oxy  ,tsnoxy  ,snicexy ,snliqxy ,&
                zsnsoxy  ,isnowxy ,snowhxy ,sneqvxy ,canliqxy,canicexy,&
                tgxy     ,tvxy    ,waxy    ,wtxy    ,zwtxy   ,lfmassxy,&
                rtmassxy ,stmassxy,woodxy  ,stblcpxy,fastcpxy,xlaixy  ,&
                xsaixy   )

     cmxy =   0.           ! only used for SFCDIF2
     chxy =   0.           ! only used for SFCDIF2

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

        nspin = 1
        it = 0

! ==============================================================================
! BEGIN TIME LOOP
! ==============================================================================
      do ispin = 1, nspin

      yloop : do iyloop  = start_year, end_year
       startmonth = 1
       if(iyloop == start_year) startmonth = start_month

       if(mod(iyloop,4).eq.0) then
          nday(2)  = 29
          istep    = gday(startmonth-1)*8
       else
          nday(2)  = 28
          istep    = jday(startmonth-1)*8
       end if
       startstep  = istep+1

      mloop : do imloop  = startmonth, 12
       startday = 1
       if(iyloop == start_year .and.  imloop == start_month) &
       startday = start_day
```

```
 imstep = 0
dloop : do idloop  = startday, nday(imloop)
 idstep = 0
hloop : do ihloop  = 1, 24

 it     = it     + 1
 istep  = istep  + 1
 imstep = imstep + 1
 idstep = idstep + 1

 write(6,'(7i8)') it,iyloop,imloop,idloop,ihloop,istep

 read(mdt(1:2),'(i2)') imonth
 read(mdt(3:4),'(i2)') iday
 read(mdt(5:6),'(i2)') itime

 !read forcing data.

 call readforc(nx   ,ny      ,dt      ,iyloop  ,imonth  ,iday     ,&
               itime,dir     ,sfctmpxy,q2xy    ,lwdnxy  ,uxy      ,&
               vxy  ,sfcprsxy,soldnxy ,prcpxy  )

 !calday to compute cosz

 call calendr (it, dt, imonth, iday, itime, calday)

 ipoint               = 0
 do 100 iy=1,ny
 do 200 ix=1,nx

 !compute cosz
 call lsmzen (calday,lonxy(ix,iy),latxy(ix,iy),cosz)

 if(vegtypxy(ix,iy) > 0) then
     ist = 1
     if(vegtypxy(ix,iy)  == 16) ist = 2   ! lake points

     ipoint               = ipoint + 1
     ice                  = abs(landseaxy(ix,iy)-1)
     lat                  = latxy    (ix,iy)
     tbot                 = tbotxy   (ix,iy)
     isc                  = iscxy    (ix,iy)
     vegtyp               = vegtypxy (ix,iy)
     soiltyp              = soiltypxy(ix,iy)
     shdfac               = shdfacxy (ix,iy,imonth)

     call redprm (vegtyp,soiltyp,sloptyp,sldpth,zsoil,nsoil)

     sfctmp               = sfctmpxy (ix,iy)
     uu                   = uxy      (ix,iy)
     vv                   = vxy      (ix,iy)
     sfcprs               = sfcprsxy (ix,iy)
     q2                   = q2xy     (ix,iy)
     prcp                 = prcpxy   (ix,iy)
     soldn                = soldnxy  (ix,iy)
     lwdn                 = lwdnxy   (ix,iy)
     sloptyp              = sloptypxy(ix,iy)
     co2air               = 355.e-06 * sfcprs       !partial pressure co2 (pa)
     o2air                = 0.209    * sfcprs       !partial pressure  o2 (pa)

 if(lwdn < 0) then
     write(*,*) ipoint, cosz, soldn,lwdn,sfctmp, uu, vv, prcp*dt,q2*1000,sfcprs
 end if

     eah                  = eahxy    (ix,iy)
     tah                  = tahxy    (ix,iy)
     fwet                 = fwetxy   (ix,iy)
     sneqvo               = sneqvoxy (ix,iy)
     albold               = alboldxy (ix,iy)
```

```
           qsnow                = qsnowxy   (ix,iy)
           wslake               = wslakexy  (ix,iy)
           cm                   = cmxy      (ix,iy)
           ch                   = chxy      (ix,iy)

           canliq               = canliqxy(ix,iy)
           canice               = canicexy(ix,iy)
           tv                   = tvxy     (ix,iy)
           tg                   = tgxy     (ix,iy)
           zwt                  = zwtxy    (ix,iy)
           wa                   = waxy     (ix,iy)
           wt                   = wtxy     (ix,iy)
           isnow                = isnowxy  (ix,iy)
           lfmass               = lfmassxy(ix,iy)
           rtmass               = rtmassxy(ix,iy)
           stmass               = stmassxy(ix,iy)
           wood                 = woodxy   (ix,iy)
           stblcp               = stblcpxy(ix,iy)
           fastcp               = fastcpxy(ix,iy)

           zsnso(isnow+1:nsoil) = zsnsoxy  (ix,iy,isnow+1:nsoil)
           snice(isnow+1:     0) = snicexy  (ix,iy,isnow+1:     0)
           snliq(isnow+1:     0) = snliqxy  (ix,iy,isnow+1:     0)
           stc   (       1:nsoil) = stcxy    (ix,iy,       1:nsoil)
           stc   (isnow+1:     0) = tsnoxy   (ix,iy,isnow+1:     0)
           smc  (       1:nsoil) = smcxy    (ix,iy,       1:nsoil)
           sh2o(       1:nsoil) = sh2oxy   (ix,iy,       1:nsoil)
           snowh                = snowhxy  (ix,iy)
           sneqv                = sneqvxy  (ix,iy)
           xlai                 = xlaixy   (ix,iy)
           xsai                 = xsaixy   (ix,iy)

           ficeold(isnow+1:0)   = snicexy(ix,iy,isnow+1:0) &
           /(snicexy(ix,iy,isnow+1:0)+snliqxy(ix,iy,isnow+1:0))

      call SFLX (ice    ,ist    ,vegtyp ,isc    ,nsnow  ,nsoil  ,&  !in
                 zsoil  ,dt     ,q2     ,sfctmp ,uu     ,vv     ,&  !in
                 soldn  ,lwdn   ,prcp   ,zlvl   ,co2air ,o2air  ,&  !in
                 cosz   ,tbot   ,foln   ,sfcprs ,imonth ,iday   ,&  !in
                 shdfac ,lat    ,z0     ,ix     ,iy     ,ipoint ,&  !in
                 eah    ,tah    ,fwet   ,ficeold,qsnow  ,sneqvo ,&  !inout
                 isnow  ,zsnso  ,canliq ,canice ,snowh  ,sneqv  ,&  !inout
                 snice  ,snliq  ,tv     ,tg     ,stc    ,sh2o   ,&  !inout
                 smc    ,zwt    ,wa     ,wt     ,wslake ,lfmass ,&  !inout
                 rtmass ,stmass ,wood   ,stblcp ,fastcp ,xlai   ,&  !inout
                 xsai   ,albold ,cm     ,ch     ,                &  !inout
                 fsa    ,fsr    ,fira   ,fsh    ,ssoil  ,fcev   ,&  !out
                 fgev   ,fctr   ,trad   ,ecan   ,etran  ,edir   ,&  !out
                 runsf  ,runsb  ,apar   ,psn    ,sav    ,sag    ,&  !out
                 fsno   ,nee    ,gpp    ,npp    ,ts     ,fveg   )   !out

!       call SFLX (ice    ,ist    ,vegtyp ,isc    ,nsnow  ,nsoil  ,&  !in
!                  zsoil  ,dt     ,q2     ,sfctmp ,uu     ,vv     ,&  !in
!                  soldn  ,lwdn   ,prcp   ,zlvl   ,co2air ,o2air  ,&  !in
!                  cosz   ,tbot   ,foln   ,sfcprs ,imonth ,iday   ,&  !in
!                  shdfac ,lat    ,ficeold,qsnow  ,sneqvo ,eah    ,&  !in
!                  tah    ,fwet   ,z0     ,ix     ,iy     ,ipoint ,&  !in     !niu
!                  isnow  ,zsnso  ,canliq ,canice ,snowh  ,sneqv  ,&  !inout
!                  snice  ,snliq  ,tv     ,tg     ,stc    ,sh2o   ,&  !inout
!                  smc    ,zwt    ,wa     ,wt     ,wslake ,lfmass ,&  !inout
!                  rtmass ,stmass ,wood   ,stblcp ,fastcp ,xlai   ,&  !inout
!                  xsai   ,albold ,                                &  !inout
!                  fsa    ,fsr    ,fira   ,fsh    ,ssoil  ,fcev   ,&  !out
!                  fgev   ,fctr   ,trad   ,ecan   ,etran  ,edir   ,&  !out
!                  runsf  ,runsb  ,apar   ,psn    ,sav    ,sag    ,&  !out
!                  fsno   ,nee    ,gpp    ,npp    ,ts     ,fveg   )   !out

           isnowxy   (ix,iy)              = isnow
           canliqxy (ix,iy)              = canliq
```

```
           canicexy (ix,iy)                = canice
           snowhxy  (ix,iy)                = snowh
           sneqvxy  (ix,iy)                = sneqv
           zsnsoxy  (ix,iy,isnow+1:nsoil) = zsnso (isnow+1:nsoil)
           stcxy    (ix,iy,       1:nsoil) = stc   (       1:nsoil)
           tsnoxy   (ix,iy,isnow+1:     0) = stc   (isnow+1:     0)
           smcxy    (ix,iy,1:nsoil)        = smc   (       1:nsoil)
           sh2oxy   (ix,iy,1:nsoil)        = sh2o  (       1:nsoil)
           snicexy  (ix,iy,isnow+1:     0) = snice (isnow+1:     0)
           snliqxy  (ix,iy,isnow+1:     0) = snliq (isnow+1:     0)
           tvxy     (ix,iy)                = tv
           tgxy     (ix,iy)                = tg
           zwtxy    (ix,iy)                = zwt
           waxy     (ix,iy)                = wa
           wtxy     (ix,iy)                = wt
           lfmassxy (ix,iy)                = lfmass
           rtmassxy (ix,iy)                = rtmass
           stmassxy (ix,iy)                = stmass
           woodxy   (ix,iy)                = wood
           stblcpxy (ix,iy)                = stblcp
           fastcpxy (ix,iy)                = fastcp
           xlaixy   (ix,iy)                = xlai
           xsaixy   (ix,iy)                = xsai

           eahxy    (ix,iy)                = eah
           tahxy    (ix,iy)                = tah
           fwetxy   (ix,iy)                = fwet
           sneqvoxy (ix,iy)                = sneqvo
           alboldxy (ix,iy)                = albold
           qsnowxy  (ix,iy)                = qsnow
           wslakexy (ix,iy)                = wslake
           cmxy     (ix,iy)                = cm
           chxy     (ix,iy)                = ch

           !for output

           runsfxy  (ix,iy)                = runsf
           runsbxy  (ix,iy)                = runsb
           ecanxy   (ix,iy)                = ecan
           edirxy   (ix,iy)                = edir
           etranxy  (ix,iy)                = etran
           aparxy   (ix,iy)                = apar
           psnxy    (ix,iy)                = psn
           savxy    (ix,iy)                = sav
           sagxy    (ix,iy)                = sag
           fsnoxy   (ix,iy)                = fsno
           fsaxy    (ix,iy)                = fsa
           firaxy   (ix,iy)                = fira
           fshxy    (ix,iy)                = fsh
           flhxy    (ix,iy)                = fcev + fgev + fctr
           fghxy    (ix,iy)                = ssoil
           tradxy   (ix,iy)                = trad
           tsxy     (ix,iy)                = ts
           neexy    (ix,iy)                = nee
           gppxy    (ix,iy)                = gpp
           nppxy    (ix,iy)                = npp
           fvegxy   (ix,iy)                = fveg

           endif           ! endif of land-points

  200   continue
  100   continue

!--------------------------------------------------------------------
! end of 1-d Noah processing
!--------------------------------------------------------------------

       if(ispin == nspin) then
          !monthly output, one file per month (1 time layer)
```

```
          call nc_out(nsoil    ,nx       ,ny       ,it       ,dt       ,  &
                      iyloop   ,imonth   ,iday     ,DIR      ,EXP,lonxy    ,  &
                      latxy    ,vegtypxy,imstep   ,nday     ,ND       ,  &
                      snowhxy  ,sneqvxy  ,tgxy     ,stcxy    ,smcxy    ,  &
                      sh2oxy   ,prcpxy   ,runsfxy  ,runsbxy  ,ecanxy   ,  &
                      edirxy   ,etranxy  ,zwtxy    ,fsaxy    ,firaxy   ,  &
                      fshxy    ,flhxy    ,fghxy    ,aparxy   ,psnxy    ,  &
                      savxy    ,sagxy    ,fsnoxy   ,xlaixy   ,xsaixy   ,  &
                      tradxy   ,neexy    ,gppxy    ,nppxy    ,tsxy     ,  &
                      fvegxy   ,cmxy     ,chxy     ,                    &
                      snowhm   ,sneqvm   ,tgm      ,stcm     ,smcm     ,  &
                      sh2om    ,prcpm    ,runsfm   ,runsbm   ,ecanm    ,  &
                      edirm    ,etranm   ,zwtm     ,fsam     ,firam    ,  &
                      fshm     ,flhm     ,fghm     ,aparm    ,psnm     ,  &
                      savm     ,sagm     ,fsnom    ,xlaim    ,xsaim    ,  &
                      tradm    ,neem     ,gppm     ,nppm     ,tsm      ,  &
                      fvegm    ,cmm      ,chm      )

         !hrly output, one file per day (24 time layers)

         if(iyloop == 2004) then
          call nc_out_3hr(nsoil    ,nx       ,ny       ,it       ,dt   ,  &
                      iyloop   ,imonth   ,iday     ,DIR      ,EXP,lonxy    ,  &
                      latxy    ,vegtypxy,idstep   ,nday     ,              &
                      snowhxy  ,sneqvxy  ,tgxy     ,stcxy    ,smcxy    ,  &
                      sh2oxy   ,prcpxy   ,runsfxy  ,runsbxy  ,ecanxy   ,  &
                      edirxy   ,etranxy  ,zwtxy    ,fsaxy    ,firaxy   ,  &
                      fshxy    ,flhxy    ,fghxy    ,aparxy   ,psnxy    ,  &
                      savxy    ,sagxy    ,fsnoxy   ,xlaixy   ,xsaixy   ,  &
                      tradxy   ,neexy    ,gppxy    ,nppxy    ,tsxy     ,  &
                      fvegxy   )
         end if
       endif

!update the time

          call geth_newdate(newdate, olddate, nint(dt))
          year     = newdate(1:4)
          mdt      = newdate(5:10)
          minute   = newdate(11:12)
          olddate  = newdate

       enddo hloop
       enddo dloop
       call write_ini(DIR,EXP     ,nx       ,ny       ,nsoil    ,nsnow    ,  &
     &      iyloop    ,imonth   ,iday     ,itime    ,latxy    ,lonxy    ,  &
     &      smcxy     ,stcxy    ,sh2oxy   ,tsnoxy   ,snicexy  ,snliqxy  ,  &
     &      zsnsoxy   ,isnowxy  ,snowhxy  ,sneqvxy  ,canliqxy,canicexy,  &
     &      tgxy      ,tvxy     ,waxy     ,wtxy     ,zwtxy    ,lfmassxy,  &
     &      rtmassxy  ,stmassxy,woodxy   ,stblcpxy,fastcpxy)
       enddo mloop
       enddo yloop

       enddo ! spinup

      end program Noah_driver
```

```
module noahlsm_globals
  implicit none

! ==========================================================================================
!------------------------------------------------------------------------------------------!
! Physical Constants:                                                                      !
!------------------------------------------------------------------------------------------!

  REAL, PARAMETER :: GRAV   = 9.80616   !acceleration due to gravity (m/s2)
  REAL, PARAMETER :: SB     = 5.67E-08  !Stefan-Boltzmann constant (w/m2/k4)
  REAL, PARAMETER :: VKC    = 0.40      !von Karman constant
  REAL, PARAMETER :: TFRZ   = 273.16    !freezing/melting point (k)
  REAL, PARAMETER :: HSUB   = 2.8440E06 !latent heat of sublimation (j/kg)
  REAL, PARAMETER :: HVAP   = 2.5104E06 !latent heat of vaporization (j/kg)
  REAL, PARAMETER :: HFUS   = 0.3336E06 !latent heat of fusion (j/kg)
  REAL, PARAMETER :: CWAT   = 4.188E06  !specific heat capacity of water (j/m3/k)
  REAL, PARAMETER :: CICE   = 2.094E06  !specific heat capacity of ice (j/m3/k)
  REAL, PARAMETER :: CPAIR  = 1004.64   !heat capacity dry air at const pres (j/kg/k)
  REAL, PARAMETER :: TKWAT  = 0.6       !thermal conductivity of water (w/m/k)
  REAL, PARAMETER :: TKICE  = 2.2       !thermal conductivity of ice (w/m/k)
  REAL, PARAMETER :: TKAIR  = 0.023     !thermal conductivity of air (w/m/k)
  REAL, PARAMETER :: RAIR   = 287.04    !gas constant for dry air (j/kg/k)
  REAL, PARAMETER :: RW     = 461.269   !gas constant for  water vapor (j/kg/k)
  REAL, PARAMETER :: DENH2O = 1000.     !density of water (kg/m3)
  REAL, PARAMETER :: DENICE = 917.      !density of ice (kg/m3)

!------------------------------------------------------------------------------------------!
! From the VEGPARM.TBL tables, as functions of vegetation category.
! The tables themselves are in module_sf_noahlsm_param_init.  These
! scalar variables are set in subroutine REDPRM (which must be
! called before the call to SFLX), but the user may override those
! settings by resetting these variables after the call to REDPRM and
! before the call to SFLX.
!------------------------------------------------------------------------------------------!
  INTEGER :: NROOT        !rooting depth [as the number of layers]
  REAL    :: RGL          !parameter used in radiation stress function
  REAL    :: RSMIN        !minimum Canopy Resistance [s/m]
  REAL    :: HS           !parameter used in vapor pressure deficit function
  REAL    :: RSMAX        !maximum stomatal resistance
  REAL    :: TOPT         !optimum transpiration air temperature.
!------------------------------------------------------------------------------------------!
! From the SOILPARM.TBL tables, as functions of soil category.
! The tables themselves are in module_sf_noahlsm_param_init.  These
! scalar variables are set in subroutine REDPRM (which must be
! called before the call to SFLX), but the user may override those
! settings by resetting these variables after the call to REDPRM and
! before the call to SFLX.
!------------------------------------------------------------------------------------------!
  REAL    :: BEXP     !B parameter
  REAL    :: SMCDRY   !dry soil moisture threshold where direct evap from top
                      !layer ends (volumetric)
  REAL    :: F1       !soil thermal diffusivity/conductivity coef
  REAL    :: SMCMAX   !porosity, saturated value of soil moisture (volumetric)
  REAL    :: SMCREF   !reference soil moisture (field capacity) (volumetric)
  REAL    :: PSISAT   !saturated soil matric potential
  REAL    :: DKSAT    !saturated soil hydraulic conductivity
  REAL    :: DWSAT    !saturated soil hydraulic diffusivity
  REAL    :: SMCWLT   !wilting point soil moisture (volumetric)
  REAL    :: QUARTZ   !soil quartz content
!------------------------------------------------------------------------------------------!
! From the GENPARM.TBL file. These scalar variables are set in
! subroutine REDPRM (which must be called before the call to SFLX), but
! the user may override those settings by resetting these variables
! after the call to REDPRM and before the call to SFLX.
!------------------------------------------------------------------------------------------!
  REAL    :: SLOPE   !slope index (0 - 1)
  REAL    :: CSOIL   !vol. soil heat capacity [j/m3/K]
  REAL    :: ZBOT    !Depth (m) of lower boundary soil temperature
  REAL    :: CZIL    !Calculate roughness length of heat
```

```
  REAL    :: REFDK   !used in compute maximum infiltration rate (used in INFIL)
  REAL    :: REFKDT  !used in compute maximum infiltration rate (used in INFIL)
  REAL    :: FRZK    !used in compute maximum infiltration rate (used in INFIL)
  REAL    :: KDT     !used in compute maximum infiltration rate (used in INFIL)
  REAL    :: FRZX    !used in compute maximum infiltration rate (used in INFIL)

! =====================================options for different schemes================================
! options for dynamic vegetation:
! 1 -> off ; 2 -> on (together with OPT_CRS = 1)

  INTEGER, PARAMETER :: DVEG    = 2   !

! options for canopy stomatal resistance
! 1-> Ball-Berry; 2->Jarvis

  INTEGER, PARAMETER :: OPT_CRS = 1    !(must 1 when DVEG = 2)

! options for soil moisture factor for stomatal resistance
! 1-> Noah (soil moisture)
! 2-> CLM  (matric potential)
! 3-> SSiB (matric potential)

  INTEGER, PARAMETER :: OPT_BTR = 1    !(suggested 1)

! options for runoff and groundwater
! 1 -> TOPMODEL with groundwater (Niu et al. 2007 JGR) ;
! 2 -> TOPMODEL with an equilibrium water table (Niu et al. 2005 JGR) ;
! 3 -> original surface and subsurface runoff (free drainage)
! 4 -> BATS surface and subsurface runoff (free drainage)

  INTEGER, PARAMETER :: OPT_RUN = 1    !(suggested 1)

! options for surface layer drag coeff (CH & CM)
! 1->M-O ; 2->original Noah (Chen97)

  INTEGER, PARAMETER :: OPT_SFC = 1    !(1 or 2)

! options for supercooled liquid water (or ice fraction)
! 1-> no iteration (Niu and Yang, 2006 JHM); 2: Koren's iteration

  INTEGER, PARAMETER :: OPT_FRZ = 1    !(1 or 2)

! options for frozen soil permeability
! 1 -> linear effects, more permeable (Niu and Yang, 2006, JHM)
! 2 -> nonlinear effects, less permeable (old)

  INTEGER, PARAMETER :: OPT_INF = 1    !(suggested 1)

! options for radiation transfer
! 1 -> modified two-stream (gap = F(solar angle, 3D structure ...)<1-FVEG)
! 2 -> two-stream applied to grid-cell (gap = 0)
! 3 -> two-stream applied to vegetated fraction (gap=1-FVEG)

  INTEGER, PARAMETER :: OPT_RAD = 1    !(suggested 1)

! options for ground snow surface albedo
! 1-> BATS; 2 -> CLASS

  INTEGER, PARAMETER :: OPT_ALB = 2    !(suggested 2)

! options for partitioning  precipitation into rainfall & snowfall
! 1 -> Jordan (1991); 2 -> BATS: when SFCTMP<TFRZ+2.2 ; 3-> SFCTMP<TFRZ

  INTEGER, PARAMETER :: OPT_SNF = 1    !(suggested 1)

! options for lower boundary condition of soil temperature
! 1 -> zero heat flux from bottom (ZBOT and TBOT not used)
! 2 -> TBOT at ZBOT (8m) read from a file (original Noah)
```

```
  INTEGER, PARAMETER :: OPT_TBOT = 2   !(suggested 2)

! options for snow/soil temperature time scheme (only layer 1)
! 1 -> semi-implicit; 2 -> full implicit (original Noah)

  INTEGER, PARAMETER :: OPT_STC = 1    !(suggested 1)
! ====================================================================================================
! runoff parameters used for SIMTOP and SIMGM:
  REAL, PARAMETER :: TIMEAN = 10.5   !gridcell mean topgraphic index (global mean)
  REAL, PARAMETER :: FSATMX = 0.38   !maximum surface saturated fraction (global mean)
  REAL :: FFF                        !runoff decay factor (m-1)
  REAL :: RSBMX                      !baseflow coefficient [mm/s]

! adjustable parameters for snow processes

  REAL, PARAMETER :: M      = 1.00   !melting factor (-)
  REAL, PARAMETER :: Z0SNO  = 0.002  !snow surface roughness length (m) (0.002)
  REAL, PARAMETER :: SSI    = 0.03   !liquid water holding capacity for snowpack (m3/m3) (0.03)
  REAL, PARAMETER :: SWEMX  = 1.00   !new snow mass to fully cover old snow (mm)
                                     !equivalent to 10mm depth (density = 100 kg/m3)

! NOTES: things to add or improve
! 1. lake model: explicit representation of lake water storage, sunlight through lake
!    with different purity, turbulent mixing of surface laker water, snow on frozen lake, etc.
! 2. shallow snow wihtout a layer: melting energy
! 3. urban model to be added.
! 4. irrigation
!------------------------------------------------------------------------------------------!

END MODULE NOAHLSM_GLOBALS
! ---------------------------------------------------------------------------
MODULE VEG_PARAMETERS

    IMPLICIT NONE

    INTEGER :: i
    INTEGER, PARAMETER :: MVT   = 27
    INTEGER, PARAMETER :: MBAND = 2

    REAL :: CH2OP(MVT)        !maximum intercepted h2o per unit lai+sai (mm)
    REAL :: DLEAF(MVT)        !characteristic leaf dimension (m)
    REAL :: Z0MVT(MVT)        !momentum roughness length (m)
    REAL :: HVT(MVT)          !top of canopy (m)
    REAL :: HVB(MVT)          !bottom of canopy (m)
    REAL :: DEN(MVT)          !tree density (no. of trunks per m2)
    REAL :: RC(MVT)           !tree crown radius (m)
    REAL :: SAIM(MVT,12)      !monthly stem area index, one-sided
    REAL :: LAIM(MVT,12)      !monthly leaf area index, one-sided
    REAL :: SLA(MVT)          !single-side leaf area per Kg [m2/kg]
    REAL :: DILEFC(MVT)       !coeficient for leaf stress death [1/s]
    REAL :: DILEFW(MVT)       !coeficient for leaf stress death [1/s]
    REAL :: FRAGR(MVT)        !fraction of growth respiration  !original was 0.3
    REAL :: LTOVRC(MVT)       !leaf turnover [1/s]

    REAL :: C3PSN(MVT)        !photosynthetic pathway: 0. = c4, 1. = c3
    REAL :: KC25(MVT)         !co2 michaelis-menten constant at 25c (pa)
    REAL :: AKC(MVT)          !q10 for kc25
    REAL :: KO25(MVT)         !o2 michaelis-menten constant at 25c (pa)
    REAL :: AKO(MVT)          !q10 for ko25
    REAL :: VCMX25(MVT)       !maximum rate of carboxylation at 25c (umol co2/m**2/s)
    REAL :: AVCMX(MVT)        !q10 for vcmx25
    REAL :: BP(MVT)           !minimum leaf conductance (umol/m**2/s)
    REAL :: MP(MVT)           !slope of conductance-to-photosynthesis relationship
    REAL :: QE25(MVT)         !quantum efficiency at 25c (umol co2 / umol photon)
    REAL :: AQE(MVT)          !q10 for qe25
    REAL :: RMF25(MVT)        !leaf maintenance respiration at 25c (umol co2/m**2/s)
    REAL :: RMS25(MVT)        !stem maintenance respiration at 25c (umol co2/kg bio/s)
    REAL :: RMR25(MVT)        !root maintenance respiration at 25c (umol co2/kg bio/s)
```

```
    REAL :: ARM(MVT)          !q10 for maintenance respiration
    REAL :: FOLNMX(MVT)       !foliage nitrogen concentration when f(n)=1 (%)
    REAL :: TMIN(MVT)         !minimum temperature for photosynthesis (k)

    REAL :: XL(MVT)           !leaf/stem orientation index
    REAL :: RHOL(MVT,MBAND)   !leaf reflectance: 1=vis, 2=nir
    REAL :: RHOS(MVT,MBAND)   !stem reflectance: 1=vis, 2=nir
    REAL :: TAUL(MVT,MBAND)   !leaf transmittance: 1=vis, 2=nir
    REAL :: TAUS(MVT,MBAND)   !stem transmittance: 1=vis, 2=nir

    REAL :: MRP(MVT)          !microbial respiration parameter (umol co2 /kg c/ s)
    REAL :: CWPVT(MVT)        !empirical canopy wind parameter

    REAL :: WRRAT(MVT)        !wood to non-wood ratio
    REAL :: WDPOOL(MVT)       !wood pool (switch 1 or 0) depending on woody or not [-]
    REAL :: TDLEF(MVT)        !characteristic T for leaf freezing [K]

!  maximum intercepted h2o per unit lai+sai (mm)
    DATA CH2OP /27*0.1/

!  characteristic leaf dimension (m)
    DATA DLEAF /27*0.04/

! momentum roughness length (m)
    DATA Z0MVT /1.00,0.06,0.06,0.06,0.06,   0.15,0.06,0.06,0.06,0.86,  &
                0.80,0.85,1.10,1.09,0.80,   0.00,0.06,0.05,0.00,0.04,  &
                0.06,0.06,0.03,0.00,0.01,   0.00,0.00/

! top of canopy (m)
    DATA HVT   /15.0,0.50,0.50,0.50,0.50,   1.25,0.50,0.50,0.50,16.0,  &
                16.0,18.0,20.0,20.0,16.0,   0.00,0.50,0.80,0.00,0.50,  &
                0.80,0.80,0.50,0.00,0.10,   0.00,0.00/

! bottom of canopy (m)
    DATA HVB   /1.00,0.10,0.10,0.10,0.10,   0.15,0.05,0.10,0.10,5.00,  &
                11.5,7.00,8.00,8.50,10.0,   0.00,0.05,0.10,0.00,0.10,  &
                0.10,0.10,0.10,0.00,0.10,   0.00,0.00/

! canopy density
    DATA DEN   /0.01,25.0,25.0,25.0,25.0,   25.0,100.,10.0,10.0,0.02,  &
                0.10,0.28,0.02,0.28,0.10,   0.01,10.0,0.10,0.01,1.00,  &
                1.00,1.00,1.00,0.00,0.01,   0.01,0.01/

! canopy radius
    DATA RC    /1.00,0.08,0.08,0.08,0.08,   0.08,0.03,0.12,0.12,3.00,  &
                1.40,1.20,3.60,1.20,1.40,   0.01,0.10,1.40,0.01,0.30,  &
                0.30,0.30,0.30,0.00,0.01,   0.01,0.01/

! leaf reflectance: 1=vis, 2=nir
    DATA (RHOL(I,1),I=1,MVT) &
               /0.00,0.11,0.11,0.11,0.11,   0.11,0.11,0.07,0.10,0.10,  &
                0.10,0.07,0.10,0.07,0.10,   0.00,0.11,0.10,0.00,0.10,  &
                0.10,0.10,0.10,0.00,0.10,   0.00,0.00/

    DATA (RHOL(I,2),I=1,MVT) &
               /0.00,0.58,0.58,0.58,0.58,   0.58,0.58,0.35,0.45,0.45,  &
                0.45,0.35,0.45,0.35,0.45,   0.00,0.58,0.45,0.00,0.45,  &
                0.45,0.45,0.45,0.00,0.45,   0.00,0.00/

! stem reflectance: 1=vis, 2=nir
    DATA (RHOS(I,1),I=1,MVT) &
               /0.00,0.36,0.36,0.36,0.36,   0.36,0.36,0.16,0.16,0.16,  &
                0.16,0.16,0.16,0.16,0.16,   0.00,0.36,0.16,0.00,0.16,  &
                0.16,0.16,0.16,0.00,0.16,   0.00,0.00/

    DATA (RHOS(I,2),I=1,MVT) &
               /0.00,0.58,0.58,0.58,0.58,   0.58,0.58,0.39,0.39,0.39,  &
                0.39,0.39,0.39,0.39,0.39,   0.00,0.58,0.39,0.00,0.39,  &
                0.39,0.39,0.39,0.00,0.39,   0.00,0.00/
```

```
! leaf transmittance: 1=vis, 2=nir
    DATA (TAUL(I,1),I=1,MVT) &
               /0.00,0.07,0.07,0.07,0.07,   0.07,0.07,0.05,0.05,0.05,  &
                0.05,0.05,0.05,0.05,0.05,   0.00,0.07,0.05,0.00,0.05,  &
                0.05,0.05,0.05,0.00,0.05,   0.00,0.00/

    DATA (TAUL(I,2),I=1,MVT) &
               /0.00,0.25,0.25,0.25,0.25,   0.25,0.25,0.10,0.10,0.25,  &
                0.25,0.10,0.25,0.10,0.25,   0.00,0.25,0.25,0.00,0.25,  &
                0.25,0.25,0.25,0.00,0.25,   0.00,0.00/

! stem transmittance: 1=vis, 2=nir
    DATA (TAUS(I,1),I=1,MVT) &
          /0.000,0.220,0.220,0.220,0.220,   0.220,0.220,0.001,0.001,0.001,  &
           0.001,0.001,0.001,0.001,0.001,   0.000,0.220,0.001,0.000,0.220,  &
           0.001,0.001,0.001,0.000,0.001,   0.000,0.000/

    DATA (TAUS(I,2),I=1,MVT) &
          /0.000,0.380,0.380,0.380,0.380,   0.380,0.380,0.001,0.001,0.001,  &
           0.001,0.001,0.001,0.001,0.001,   0.000,0.380,0.001,0.000,0.380,  &
           0.001,0.001,0.001,0.000,0.001,   0.000,0.000/

! leaf/stem orientation index: valid range = -0.4 to 0.6
    DATA XL /0.000,-0.30,-0.30,-0.30,-0.30,   -0.30,-0.30,0.010,0.250,0.010,  &
             0.250,0.010,0.010,0.010,0.250,   0.000,-0.30,0.250,0.000,-0.30,  &
             0.250,0.250,0.250,0.000,0.250,   0.000,0.000/

! empirical canopy wind parameter
    DATA CWPVT /27*3.0/

! photosynthetic pathway: c3 = 1, c4 = 0 (warm grass)
    DATA C3PSN /27*1./

! co2 michaelis-menten constant at 25c (pa)
    DATA KC25 /27*30./

! q10 for kc25
    DATA AKC /27*2.1/

! o2 michaelis-menten constant at 25c (pa)
    DATA KO25 /27*30000./

! q10 for ko25
    DATA AKO /27*1.2/

! q10 for vcmx25
    DATA AVCMX /27*2.4/

! q10 for qe25
    DATA AQE /27*1.0/
!----------------------------------------------------------------------------------------------
! leaf turnover rate (1/s) ! original was 0.02 e-6
    DATA LTOVRC /0.0 ,1.6,1.8 ,1.2,1.2,    1.30,0.50,0.65,0.70,0.65,  &     !MODIS
                 0.55,0.2,0.55,0.5,0.5,    0.0 ,1.4 ,1.4 ,0.0 ,1.2 ,  &     !MODIS
                 1.3 ,1.4,1.0 ,0.0,1.0,    0.0 ,0.0/                        !MODIS

! leaf dying rate (1.e-6;original was 0.2e-6)
    DATA DILEFC /0.00, 0.50, 0.50, 0.50, 0.35,    0.20, 0.20, 0.20, 0.50, 0.50,  &   !MODIS
                 0.60, 1.80, 0.50, 1.20, 0.80,    0.00, 0.40, 0.40, 0.00, 0.40,  &   !MODIS
                 0.30, 0.40, 0.30, 0.00, 0.30,    0.00, 0.00  /                      !MODIS

! leaf dying rate (1.e-6;original was 0.2e-6)
    DATA DILEFW /0.00, 0.20, 0.20, 0.20, 0.20,    0.20, 0.10, 0.20, 0.20, 0.50,  &   !MODIS
                 0.20, 0.20, 4.00, 0.20, 0.20,    0.00, 0.20, 0.20, 0.00, 0.20,  &   !MODIS
                 0.20, 0.20, 0.20, 0.00, 0.20,    0.00, 0.00  /                      !MODIS

! foliage maintenance respiration rate at 25c (umol co2 /m**2 /s)
!      data RMF25 /0.00,0.75,0.75,0.75,0.75,    0.75,0.50,0.26,0.26,0.50,  &
```

```
!                   0.50,0.50,0.75,0.40,0.40,    0.00,0.50,0.50,0.00,0.26,  &
!                   0.26,0.26,0.26,0.00,0.26,    0.00,0.00/
      data RMF25 /0.00,1.00,1.40,1.45,1.45,    1.45,1.80,0.26,0.26,0.80,  &
                  3.00,4.00,0.65,3.00,3.00,    0.00,3.20,3.20,0.00,3.20,  &
                  3.00,3.00,3.00,0.00,3.00,    0.00,0.00/
      ! 0.82 for warm grass

! leaf area per unit mass (m2/kg)
    DATA SLA   / 60,  80,  80,  80,  80,    80,  60,  60,  60,  50,  &   !MODIS
                 80,  80,  80,  80,  80,     0,  80,  80,   0,  80,  &   !MODIS
                 80,  80,  80,   0,  80,     0,   0 /                    !MODIS

! growth respiration fraction
    DATA FRAGR  /0.00,  0.20,  0.20,  0.20,  0.20,    0.20,  0.20,  0.20,  0.20,  0.20,  &   !MODIS
                 0.20,  0.10,  0.20,  0.10,  0.10,    0.00,  0.10,  0.10,  0.10,  0.10,  &   !MODIS
                 0.10,  0.10,  0.10,  0.00,  0.10,    0.00,  0.00  /                        !MODIS
!-----------------------------------------------------------------------------------------------

! minimum temperature for photosynthesis (k)
    DATA TMIN   /   0,273,273,273,273,   273,273,273,273,273,  &
                 273,268,273,265,268,      0,268,268,   0,268,  &
                 268,268,268,   0,268,      0,   0/

! maximum rate of carboxylation at 25c (umol co2/m**2/s)
    DATA VCMX25/0.00,  80.0,  80.0,  80.0,  60.0,    70.0,40.0,40.0,40.0,40.0,  &
                60.0,  60.0,  60.0,  50.0,  55.0,    0.00,50.0,50.0,0.00,50.0,  &
                50.0,  50.0,  50.0,  0.00,  50.0,    0.00,0.00/

! leaf death temperature [K]
    data tdlef /278,278,278,278,278,    278,278,278,278,278,  &
                278,268,278,278,268,    0,268,268,0,268,  &
                268,268,268,0  ,268,    0,0/

! minimum leaf conductance (umol/m**2/s)
    DATA BP /1.E15,14*2000.,1.e15,7*2000.,1.e15,2000.,1.e15,1.e15/

! slope for conductance-to-photosynthesis relationship
!niu    DATA MP /9.,9.,9.,9.,9.,    9.,5.,9.,9.,9.,  &  ! 5 - for warm grass
    DATA MP /9.,9.,9.,9.,9.,    9.,9.,9.,9.,9.,  &  ! 5 - for warm grass
             9.,6.,9.,6.,9.,    9.,9.,9.,9.,9.,  &
             9.,9.,9.,9.,9.,    9.,9./

! quantum efficiency at 25c (umol co2 / umol photon) ! 0.04 for warm grass
    DATA QE25 /0.00,14*0.06,0.00,7*0.06,0.00,0.06,0.00,0.00/

! stem maintenance respiration at 25c (umol co2/kg biomass/s)
    DATA RMS25 /0.00,0.10,0.10,0.10,0.10,    0.10,0.10,0.10,0.10,0.32,  &
                0.10,0.64,0.30,0.90,0.80,    0.00,0.10,0.10,0.00,0.10,  &
                0.10,0.10,0.00,0.00,0.00,    0.00,0.00/

! root maintenance respiration at 25c (umol co2/kg biomass/s)
    DATA RMR25 /0.00,0.00,0.00,0.00,0.00,    0.00,1.20,0.00,0.00,0.01,  &
                0.01,0.05,0.05,0.36,0.03,    0.00,0.00,0.00,0.00,2.11,  &
                2.11,2.11,0.00,0.00,0.00,    0.00,0.00/

! q10 for maintenance respiration
    DATA ARM /27*2.0/

! foliage nitrogen concentration when f(n)=1 (-)
    DATA FOLNMX /0.00,14*1.5,0.00,7*1.5,0.00,1.5,0.00,0.00/

! wood pool (switch 1 or 0) depending on woody or not
    DATA WDPOOL/0.00,0.00,0.00,0.00,0.00,   0.00,0.00,1.00,1.00,1.00,  &
                1.00,1.00,1.00,1.00,1.00,   0.00,0.00,1.00,0.00,0.00,  &
                1.00,1.00,0.00,0.00,0.00,   0.00,0.00/

! wood to non-wood ratio                          ! 30.0
    DATA WRRAT /0.00,0.00,0.00,0.00,0.00,   0.00,0.00,3.00,3.00,3.00,  &
                30.0,30.0,30.0,30.0,30.0,   0.00,0.00,30.0,0.00,0.00,  &
```

```
                3.00,3.00,0.00,0.00,0.00,   0.00,0.00/

! microbial respiration parameter (umol co2 /kg c /s)
    DATA MRP   /0.00,0.23,0.23,0.23,0.23,   0.23,0.17,0.19,0.19,0.40,  &
                0.40,0.37,0.23,0.37,0.30,   0.00,0.17,0.40,0.00,0.17,  &
                0.23,0.20,0.00,0.00,0.20,   0.00,0.00/

! monthly stem area index
    DATA (SAIM( 1,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM( 2,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM( 3,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM( 4,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM( 5,I),I=1,12) /0.0,0.0,0.0,0.0,0.1,0.2,0.3,0.5,0.4,0.1,0.0,0.0/
    DATA (SAIM( 6,I),I=1,12) /0.1,0.1,0.1,0.1,0.2,0.2,0.3,0.2,0.1,0.1,0.1,0.1/
    DATA (SAIM( 7,I),I=1,12) /0.3,0.3,0.3,0.3,0.3,0.4,0.8,1.3,1.1,0.4,0.4,0.4/
    DATA (SAIM( 8,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.1,0.1,0.1,0.1,0.1/
    DATA (SAIM( 9,I),I=1,12) /0.2,0.2,0.2,0.2,0.2,0.3,0.5,0.8,0.5,0.2,0.2,0.2/
    DATA (SAIM(10,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1/
    DATA (SAIM(11,I),I=1,12) /0.4,0.4,0.4,0.4,0.4,0.4,0.9,1.2,1.6,1.4,0.6,0.4/
    DATA (SAIM(12,I),I=1,12) /0.3,0.3,0.3,0.4,0.4,0.7,1.3,1.2,1.0,0.8,0.6,0.5/
    DATA (SAIM(13,I),I=1,12) /0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5/
    DATA (SAIM(14,I),I=1,12) /0.4,0.4,0.4,0.3,0.4,0.5,0.5,0.6,0.6,0.7,0.6,0.5/
    DATA (SAIM(15,I),I=1,12) /0.2,0.2,0.2,0.2,0.2,0.4,0.4,0.5,0.5,0.6,0.5,0.3/
    DATA (SAIM(16,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(17,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.4,0.6,0.5,0.2,0.2,0.1/
    DATA (SAIM(18,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.4,0.6,0.5,0.2,0.2,0.1/
    DATA (SAIM(19,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(20,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.3,0.3,0.2,0.2,0.1/
    DATA (SAIM(21,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.3,0.3,0.2,0.2,0.1/
    DATA (SAIM(22,I),I=1,12) /0.1,0.1,0.1,0.1,0.1,0.2,0.2,0.3,0.3,0.2,0.2,0.1/
    DATA (SAIM(23,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(24,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(25,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(26,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (SAIM(27,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/

! monthly leaf area index, one-sided
    DATA (LAIM( 1,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM( 2,I),I=1,12) /0.0,0.0,0.0,0.0,1.0,2.0,3.0,3.0,1.5,0.0,0.0,0.0/
    DATA (LAIM( 3,I),I=1,12) /0.4,0.5,0.6,0.7,1.2,3.0,3.5,1.5,0.7,0.6,0.5,0.4/
    DATA (LAIM( 4,I),I=1,12) /0.4,0.5,0.6,0.7,1.2,3.0,3.5,1.5,0.7,0.6,0.5,0.4/
    DATA (LAIM( 5,I),I=1,12) /0.4,0.5,0.6,0.7,1.2,3.0,3.5,1.5,0.7,0.6,0.5,0.4/
    DATA (LAIM( 6,I),I=1,12) /0.0,0.0,0.0,0.5,1.5,2.5,3.5,3.5,2.0,1.0,0.0,0.0/
    DATA (LAIM( 7,I),I=1,12) /0.4,0.5,0.6,0.7,1.2,3.0,3.5,1.5,0.7,0.6,0.5,0.4/
    DATA (LAIM( 8,I),I=1,12) /1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0/
    DATA (LAIM( 9,I),I=1,12) /1.0,1.0,1.0,1.5,2.0,2.5,3.0,2.5,1.5,1.0,1.0,1.0/
    DATA (LAIM(10,I),I=1,12) /1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0/
    DATA (LAIM(11,I),I=1,12) /0.0,0.0,0.3,1.2,3.0,4.7,4.5,3.4,1.2,0.3,0.0,0.0/
    DATA (LAIM(12,I),I=1,12) /0.0,0.0,0.0,0.6,1.2,2.0,2.6,1.7,1.0,0.5,0.2,0.0/
    DATA (LAIM(13,I),I=1,12) /4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5,4.5/
    DATA (LAIM(14,I),I=1,12) /1.6,1.6,1.6,1.6,5.3,5.5,5.3,5.3,4.2,2.2,2.2,2.2/
    DATA (LAIM(15,I),I=1,12) /1.0,1.0,1.0,1.0,2.3,3.5,4.3,3.3,2.2,1.2,1.2,1.2/
    DATA (LAIM(16,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(17,I),I=1,12) /0.4,0.5,0.6,0.7,1.2,3.0,3.5,1.5,0.7,0.6,0.5,0.4/
    DATA (LAIM(18,I),I=1,12) /0.2,0.4,0.4,0.4,0.5,0.7,1.7,3.0,2.5,1.6,0.8,0.4/
    DATA (LAIM(19,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(20,I),I=1,12) /0.0,0.0,0.0,0.0,0.2,0.5,1.0,2.0,1.0,0.5,0.2,0.0/
    DATA (LAIM(21,I),I=1,12) /0.0,0.0,0.0,0.0,0.2,0.5,1.0,2.0,1.0,0.5,0.2,0.0/
    DATA (LAIM(22,I),I=1,12) /0.0,0.0,0.0,0.0,0.2,0.5,1.0,2.0,1.0,0.5,0.2,0.0/
    DATA (LAIM(23,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(24,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(25,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(26,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/
    DATA (LAIM(27,I),I=1,12) /0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0/

END MODULE VEG_PARAMETERS
! ==================================================================================================
! ==================================================================================================
MODULE RAD_PARAMETERS
```

```
    IMPLICIT NONE

    INTEGER I                  ! loop index
    INTEGER, PARAMETER :: MSC   = 9
    INTEGER, PARAMETER :: MBAND = 2

    REAL :: ALBSAT(MSC,MBAND)   !saturated soil albedos: 1=vis, 2=nir
    REAL :: ALBDRY(MSC,MBAND)   !dry soil albedos: 1=vis, 2=nir
    REAL :: ALBICE(MBAND)       !albedo land ice: 1=vis, 2=nir
    REAL :: ALBLAK(MBAND)       !albedo frozen lakes: 1=vis, 2=nir
    REAL :: OMEGAS(MBAND)       !two-stream parameter omega for snow
    REAL :: BETADS              !two-stream parameter betad for snow
    REAL :: BETAIS              !two-stream parameter betad for snow
    REAL :: EG(2)               !emissivity

! saturated soil albedos: 1=vis, 2=nir
    DATA(ALBSAT(I,1),I=1,8)/0.15,0.11,0.10,0.09,0.08,0.07,0.06,0.05/
    DATA(ALBSAT(I,2),I=1,8)/0.30,0.22,0.20,0.18,0.16,0.14,0.12,0.10/

! dry soil albedos: 1=vis, 2=nir
    DATA(ALBDRY(I,1),I=1,8)/0.27,0.22,0.20,0.18,0.16,0.14,0.12,0.10/
    DATA(ALBDRY(I,2),I=1,8)/0.54,0.44,0.40,0.36,0.32,0.28,0.24,0.20/

! albedo land ice: 1=vis, 2=nir
    DATA (ALBICE(I),I=1,MBAND) /0.80, 0.55/

! albedo frozen lakes: 1=vis, 2=nir
    DATA (ALBLAK(I),I=1,MBAND) /0.60, 0.40/

! omega,betad,betai for snow
    DATA (OMEGAS(I),I=1,MBAND) /0.8, 0.4/
    DATA BETADS, BETAIS /0.5, 0.5/

! emissivity ground surface
      DATA EG /0.97, 0.98/ ! 1-soil;2-lake

END MODULE RAD_PARAMETERS
! ==================================================================================================

MODULE NOAHLSM_ROUTINES
  USE NOAHLSM_GLOBALS
  IMPLICIT NONE

  public  :: SFLX
  public  :: REDPRM
  public  :: LSMZEN
  public  :: CALENDR

  private :: ATM
  private :: PHONOLOGY
  private :: ENERGY
  private ::       THERMOPROP
  private ::               CSNOW
  private ::               TDFCND
  private ::       RADIATION
  private ::               ALBEDO
  private ::                         SNOW_AGE
  private ::                         SNOWALB_BATS
  private ::                         SNOWALB_CLASS
  private ::                         GROUNDALB
  private ::                         TWOSTREAM
  private ::               SURRAD
  private ::       VEGE_FLUX
  private ::               SFCDIF1
  private ::               SFCDIF2
  private ::               STOMATA
  private ::               CANRES
  private ::               ESAT
```

```
  private ::                RAGRB
  private ::        BARE_FLUX
  private ::        TSNOSOI
  private ::                HRT
  private ::                HSTEP
  private ::                          ROSR12
  private ::        PHASECHANGE
  private ::                FRH2O

  private :: WATER
  private ::        CANWATER
  private ::        SNOWWATER
  private ::                SNOWFALL
  private ::                COMBINE
  private ::                DIVIDE
  private ::                          COMBO
  private ::                COMPACT
  private ::                SNOWH2O
  private ::        SOILWATER
  private ::                ZWTEQ
  private ::                INFIL
  private ::                SRT
  private ::                          WDFCND1
  private ::                          WDFCND2
!  private ::                          INFIL
  private ::                SSTEP
  private ::        GROUNDWATER

  private :: CARBON
  private ::        CO2FLUX
!  private ::        BVOCFLUX
!  private ::        CH4FLUX

  private :: ERROR

contains
!
! ==================================================================================================
  SUBROUTINE SFLX (ICE    , IST    , VEGTYP , ISC    , NSNOW  , NSOIL  ,  & !in
                   ZSOIL  , DT     , Q2     , SFCTMP , UU     , VV     ,  & !in
                   SOLDN  , LWDN   , PRCP   , ZLVL   , CO2AIR , O2AIR  ,  & !in
                   COSZ   , TBOT   , FOLN   , SFCPRS , IMONTH , IDAY   ,  & !in
                   SHDFAC , LAT    , Z0     , IX     , IY     , ipoint ,  & !in
                   EAH    , TAH    , FWET   , FICEOLD, QSNOW  , SNEQVO ,  & !inout
                   ISNOW  , ZSNSO  , CANLIQ , CANICE , SNOWH  , SNEQV  ,  & !inout
                   SNICE  , SNLIQ  , TV     , TG     , STC    , SH2O   ,  & !inout
                   SMC    , ZWT    , WA     , WT     , WSLAKE , LFMASS ,  & !inout
                   RTMASS , STMASS , WOOD   , STBLCP , FASTCP , LAI    ,  & !inout
                   SAI    , ALBOLD , CM     , CH     ,                    & !inout
                   FSA    , FSR    , FIRA   , FSH    , SSOIL  , FCEV   ,  & !out
                   FGEV   , FCTR   , TRAD   , ECAN   , ETRAN  , EDIR   ,  & !out
                   RUNSRF , RUNSUB , APAR   , PSN    , SAV    , SAG    ,  & !out
                   FSNO   , NEE    , GPP    , NPP    , TS     , FVEG   )   !out

! --------------------------------------------------------------------------------------------------
! Code history:
! Initial code: Guo-Yue Niu, Oct. 2007
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
  USE RAD_PARAMETERS
! --------------------------------------------------------------------------------------------------
  implicit none
! --------------------------------------------------------------------------------------------------
! input
  INTEGER                        , INTENT(IN)    :: IMONTH !month index
  INTEGER                        , INTENT(IN)    :: IDAY   !day index
  INTEGER                        , INTENT(IN)    :: ICE    !ice (ice = 1)
  INTEGER                        , INTENT(IN)    :: IST    !surface type 1->soil; 2->lake
  INTEGER                        , INTENT(IN)    :: VEGTYP !vegetation type
```

```
  INTEGER                        , INTENT(IN)    :: ISC    !soil color type (1-lighest; 8-darkest)
  INTEGER                        , INTENT(IN)    :: NSNOW  !maximum no. of snow layers
  INTEGER                        , INTENT(IN)    :: NSOIL  !no. of soil layers
  INTEGER                        , INTENT(IN)    :: IX     !grid index in e-w direction
  INTEGER                        , INTENT(IN)    :: IY     !grid index in n-s direction
  INTEGER                        , INTENT(IN)    :: ipoint !grid index
  REAL                           , INTENT(IN)    :: DT     !time step [sec]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)    :: ZSOIL  !layer-bottom depth from soil surf (m)
  REAL                           , INTENT(IN)    :: Q2     !mixing ratio (kg/kg)
  REAL                           , INTENT(IN)    :: SFCTMP !surface air temperature [K]
  REAL                           , INTENT(IN)    :: UU     !wind speed in eastward dir (m/s)
  REAL                           , INTENT(IN)    :: VV     !wind speed in northward dir (m/s)
  REAL                           , INTENT(IN)    :: SOLDN  !downward shortwave radiation (w/m2)
  REAL                           , INTENT(IN)    :: PRCP   !precipitation rate (kg m-2 s-1)
  REAL                           , INTENT(IN)    :: LWDN   !downward longwave radiation (w/m2)
  REAL                           , INTENT(IN)    :: SFCPRS !pressure (pa)
  REAL                           , INTENT(INOUT) :: ZLVL   !reference height (m)
  REAL                           , INTENT(IN)    :: COSZ   !cosine solar zenith angle [0-1]
  REAL                           , INTENT(IN)    :: TBOT   !bottom condition for soil temp. [K]
  REAL                           , INTENT(IN)    :: FOLN   !foliage nitrogen (%) [1-saturated]
  REAL                           , INTENT(IN)    :: Z0     !roughness length (m)
  REAL                           , INTENT(IN)    :: SHDFAC !green vegetation fraction [0.0-1.0]
  REAL                           , INTENT(IN)    :: LAT    !latitude (radians)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN)    :: FICEOLD!ice fraction at last timestep

! input/output : need arbitary intial values
  REAL                           , INTENT(INOUT) :: QSNOW  !snowfall [mm/s]
  REAL                           , INTENT(INOUT) :: FWET   !wetted or snowed fraction of canopy (-)
  REAL                           , INTENT(INOUT) :: SNEQVO !snow mass at last time step (mm)
  REAL                           , INTENT(INOUT) :: EAH    !canopy air vapor pressure (pa)
  REAL                           , INTENT(INOUT) :: TAH    !canopy air tmeperature (k)
  REAL                           , INTENT(INOUT) :: ALBOLD !snow albedo at last time step (CLASS type)
  REAL                           , INTENT(INOUT) :: CM     !momentum drag coefficient
  REAL                           , INTENT(INOUT) :: CH     !sensible heat exchange coefficient

! prognostic variables
  INTEGER                        , INTENT(INOUT) :: ISNOW  !actual no. of snow layers [-]
  REAL                           , INTENT(INOUT) :: CANLIQ !intercepted liquid water (mm)
  REAL                           , INTENT(INOUT) :: CANICE !intercepted ice mass (mm)
  REAL                           , INTENT(INOUT) :: SNEQV  !snow water eqv. [mm]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SMC    !soil moisture (ice + liq.) [m3/m3]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: ZSNSO  !layer-bottom depth from snow surf [m]
  REAL                           , INTENT(INOUT) :: SNOWH  !snow height [m]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNICE  !snow layer ice [mm]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNLIQ  !snow layer liquid water [mm]
  REAL                           , INTENT(INOUT) :: TV     !vegetation temperature (k)
  REAL                           , INTENT(INOUT) :: TG     !ground temperature (k)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC    !snow/soil temperature [k]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SH2O   !liquid soil moisture [m3/m3]
  REAL                           , INTENT(INOUT) :: ZWT    !depth to water table [m]
  REAL                           , INTENT(INOUT) :: WA     !water storage in aquifer [mm]
  REAL                           , INTENT(INOUT) :: WT     !water in aquifer&saturated soil [mm]

! output
  REAL                           , INTENT(OUT)   :: FSA    !total absorbed solar radiation (w/m2)
  REAL                           , INTENT(OUT)   :: FSR    !total reflected solar radiation (w/m2)
  REAL                           , INTENT(OUT)   :: FIRA   !total net LW rad (w/m2)  [+ to atm]
  REAL                           , INTENT(OUT)   :: FSH    !total sensible heat (w/m2) [+ to atm]
  REAL                           , INTENT(OUT)   :: FCEV   !canopy evap heat (w/m2) [+ to atm]
  REAL                           , INTENT(OUT)   :: FGEV   !ground evap heat (w/m2) [+ to atm]
  REAL                           , INTENT(OUT)   :: FCTR   !transpiration heat (w/m2) [+ to atm]
  REAL                           , INTENT(OUT)   :: SSOIL  !ground heat flux (w/m2)   [+ to soil]
  REAL                           , INTENT(OUT)   :: TRAD   !surface radiative temperature (k)
  REAL                           , INTENT(OUT)   :: TS     !surface temperature (k)
  REAL                           , INTENT(OUT)   :: ECAN   !evaporation of intercepted water (mm/s)
  REAL                           , INTENT(OUT)   :: ETRAN  !transpiration rate (mm/s)
  REAL                           , INTENT(OUT)   :: EDIR   !soil surface evaporation rate (mm/s]
  REAL                           , INTENT(OUT)   :: RUNSRF !surface runoff [mm/s]
  REAL                           , INTENT(OUT)   :: RUNSUB !baseflow (saturation excess) [mm/s]
```

```
  REAL                              , INTENT(OUT)   :: PSN    !total photosynthesis (umol co2/m2/s) [+]
  REAL                              , INTENT(OUT)   :: APAR   !photosyn active energy by canopy (w/m2)
  REAL                              , INTENT(OUT)   :: SAV    !solar rad absorbed by veg. (w/m2)
  REAL                              , INTENT(OUT)   :: SAG    !solar rad absorbed by ground (w/m2)
  REAL                              , INTENT(OUT)   :: FSNO   !snow cover fraction on the ground (-)
  REAL                              , INTENT(OUT)   :: FVEG   !green vegetation fraction [0.0-1.0]

! local
  INTEGER                                           :: IZ     !do-loop index
  INTEGER, DIMENSION(-NSNOW+1:NSOIL)                :: IMELT  !phase change index [1-melt; 2-freeze]
  REAL                                              :: CMC    !intercepted water (CANICE+CANLIQ) (mm)
  REAL                                              :: QMELT  !snowmelt [mm/s]
  REAL                                              :: PONDING!surface ponding [mm]
  REAL                                              :: TAUX   !wind stress: e-w (n/m2)
  REAL                                              :: TAUY   !wind stress: n-s (n/m2)
  REAL                                              :: RHOAIR !density air (kg/m3)
!  REAL, DIMENSION(       1:    5)                   :: VOCFLX !voc fluxes [ug C m-2 h-1]
  REAL, DIMENSION(-NSNOW+1:NSOIL)                   :: DZSNSO !snow/soil layer thickness [m]
  REAL                                              :: THAIR  !potential temperature (k)
  REAL                                              :: QAIR   !specific humidity (kg/kg) (q2/(1+q2))
  REAL                                              :: EAIR   !vapor pressure air (pa)
  REAL, DIMENSION(       1:    2)                   :: SOLAD  !incoming direct solar rad (w/m2)
  REAL, DIMENSION(       1:    2)                   :: SOLAI  !incoming diffuse solar rad (w/m2)
  REAL                                              :: QPRECC !convective precipitation (mm/s)
  REAL                                              :: QPRECL !large-scale precipitation (mm/s)
  REAL                                              :: IGS    !growing season index (0=off, 1=on)
  REAL                                              :: ELAI   !leaf area index, after burying by snow
  REAL                                              :: ESAI   !stem area index, after burying by snow
  REAL                                              :: BEVAP  !soil water evaporation factor (0 - 1)
  REAL, DIMENSION(       1:NSOIL)                   :: BTRANI !Soil water transpiration factor (0 - 1)
  REAL                                              :: BTRAN  !soil water transpiration factor (0 - 1)
  REAL                                              :: HTOP   !top of canopy layer (m)
  REAL                                              :: QIN    !groundwater recharge [mm/s]
  REAL                                              :: QDIS   !groundwater discharge [mm/s]
  REAL, DIMENSION(       1:NSOIL)                   :: SICE   !soil ice content (m3/m3)
  REAL, DIMENSION(-NSNOW+1:    0)                   :: SNICEV !partial volume ice of snow [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0)                   :: SNLIQV !partial volume liq of snow [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0)                   :: EPORE  !effective porosity [m3/m3]
  REAL                                              :: TOTSC  !total soil carbon (g/m2)
  REAL                                              :: TOTLB  !total living carbon (g/m2)
  REAL                                              :: T2M    !2-meter air temperature (k)
  REAL                                              :: WSLAKE !lake water storage (can be neg.) (mm)
  REAL                                              :: QDEW   !ground surface dew rate [mm/s]
  REAL                                              :: QVAP   !ground surface evap. rate [mm/s]
  REAL                                              :: LATHEA !latent heat [j/kg]
  REAL                                              :: ALBEDO !surface albedo [-]
  REAL                                              :: SWDOWN !downward solar [w/m2]
  REAL                                              :: BEG_WB !water storage at begin of a step [mm]

! carbon
! inputs
  REAL                              , INTENT(IN)    :: CO2AIR !atmospheric co2 concentration (pa)
  REAL                              , INTENT(IN)    :: O2AIR  !atmospheric o2 concentration (pa)

! inputs and outputs : prognostic variables
  REAL                           , INTENT(INOUT)    :: LFMASS !leaf mass [g/m2]
  REAL                           , INTENT(INOUT)    :: RTMASS !mass of fine roots [g/m2]
  REAL                           , INTENT(INOUT)    :: STMASS !stem mass [g/m2]
  REAL                           , INTENT(INOUT)    :: WOOD   !mass of wood (incl. woody roots) [g/m2]
  REAL                           , INTENT(INOUT)    :: STBLCP !stable carbon in deep soil [g/m2]
  REAL                           , INTENT(INOUT)    :: FASTCP !short-lived carbon, shallow soil [g/m2]
  REAL                           , INTENT(INOUT)    :: LAI    !leaf area index [-]
  REAL                           , INTENT(INOUT)    :: SAI    !stem area index [-]

! outputs
  REAL                             , INTENT(OUT)    :: NEE    !net ecosys exchange (g/m2/s CO2)
  REAL                             , INTENT(OUT)    :: GPP    !net instantaneous assimilation [g/m2/s C]
  REAL                             , INTENT(OUT)    :: NPP    !net primary productivity [g/m2/s C]
  REAL                                              :: AUTORS !net ecosystem respiration (g/m2/s C)
```

```
  REAL                                         :: HETERS !organic respiration (g/m2/s C)
  REAL                                         :: TROOT  !root-zone averaged temperature (k)

! ----------------------------------------------------------------------------------------------------
! re-process atmospheric forcing

   CALL ATM (SFCPRS ,SFCTMP ,Q2     ,PRCP   ,SOLDN  ,COSZ   ,THAIR  , &
             QAIR   ,EAIR   ,RHOAIR ,QPRECC ,QPRECL ,SOLAD  ,SOLAI  , &
             SWDOWN )

! snow/soil layer thickness (m)

     DO IZ = ISNOW+1, NSOIL
         IF(IZ == ISNOW+1) THEN
           DZSNSO(IZ) = - ZSNSO(IZ)
         ELSE
           DZSNSO(IZ) = ZSNSO(IZ-1) - ZSNSO(IZ)
         END IF
     END DO

! root-zone temperature

     TROOT  = 0.
     DO IZ=1,NROOT
        TROOT = TROOT + STC(IZ)*DZSNSO(IZ)/(-ZSOIL(NROOT))
     ENDDO

! total water storage for water balance check

     IF(IST == 1) THEN
     BEG_WB = CANLIQ + CANICE + SNEQV + WA
     DO IZ = 1,NSOIL
        BEG_WB = BEG_WB + SMC(IZ) * DZSNSO(IZ) * 1000.
     END DO
     END IF

! vegetation phonology

     !input GVF should be consistent with LAI
     IF(DVEG == 1) THEN
        FVEG = SHDFAC
        IF(FVEG <= 0.05) FVEG = MAX(SHDFAC,1.-EXP(-0.52*(LAI+SAI)))
     ELSE
        FVEG = 1.-EXP(-0.52*(LAI+SAI))
     ENDIF

     CALL PHONOLOGY (VEGTYP,IMONTH ,IDAY   ,SNOWH  ,TV     ,LAT   , & !in
                     LAI   ,SAI    ,TROOT  ,                 & !in
                     HTOP  ,ELAI   ,ESAI   ,IGS    )    !out

! compute energy budget (momentum & energy fluxes and phase changes)

    CALL ENERGY (ICE    ,VEGTYP ,IST    ,ISC    ,NSNOW  ,NSOIL  , & !in
                 ISNOW  ,NROOT  ,DT     ,RHOAIR ,SFCPRS ,QAIR   , & !in
                 SFCTMP ,THAIR  ,LWDN   ,UU     ,VV     ,ZLVL   , & !in
                 CO2AIR ,O2AIR  ,SOLAD  ,SOLAI  ,COSZ   ,IGS    , & !in
                 EAIR   ,HTOP   ,TBOT   ,ZBOT   ,ZSNSO  ,ZSOIL  , & !in
                 ELAI   ,ESAI   ,CSOIL  ,FWET   ,FOLN   ,Z0     , & !in
                 FVEG   ,                                         & !in
                 QSNOW  ,DZSNSO ,LAT    ,CANLIQ ,CANICE ,ipoint , & !in
                 IMELT  ,SNICEV ,SNLIQV ,EPORE  ,T2M    ,FSNO   , & !out
                 SAV    ,SAG    ,QMELT  ,FSA    ,FSR    ,TAUX   , & !out
                 TAUY   ,FIRA   ,FSH    ,FCEV   ,FGEV   ,FCTR   , & !out
                 TRAD   ,PSN    ,APAR   ,SSOIL  ,BTRANI ,BTRAN  , & !out
                 PONDING,TS     ,LATHEA ,                         & !out
                 TV     ,TG     ,STC    ,SNOWH  ,EAH    ,TAH    , & !inout
                 SNEQVO ,SNEQV  ,SH2O   ,SMC    ,SNICE  ,SNLIQ  , & !inout
                 ALBOLD ,CM     ,CH     )    !inout
```

```
!     write(*,'(a20,10F15.5)') 'SFLX:FSH=',SAG,FSH,FGEV,SNEQV

    SICE(:) = MAX(0.0, SMC(:) - SH2O(:))
    SNEQVO  = SNEQV

    QVAP = MAX( FGEV/LATHEA, 0.)        ! positive part of fgev
    QDEW = ABS( MIN(FGEV/LATHEA, 0.))   ! negative part of fgev
    EDIR = QVAP - QDEW

! compute water budgets (water storages, ET components, and runoff)

     CALL WATER (VEGTYP ,NSNOW  ,NSOIL  ,IMELT  ,DT     ,UU     , & !in
                 VV     ,FCEV   ,FCTR   ,QPRECC ,QPRECL ,ELAI   , & !in
                 ESAI   ,SFCTMP ,QVAP   ,QDEW   ,ZSOIL  ,BTRANI , & !in
                 FICEOLD,PONDING,TG     ,IST    ,FVEG   ,ipoint , & !in
                 ISNOW  ,CANLIQ ,CANICE ,TV     ,SNOWH  ,SNEQV  , & !inout
                 SNICE  ,SNLIQ  ,STC    ,ZSNSO  ,SH2O   ,SMC    , & !inout
                 SICE   ,ZWT    ,WA     ,WT     ,DZSNSO ,WSLAKE , & !inout
                 CMC    ,ECAN   ,ETRAN  ,FWET   ,RUNSRF ,RUNSUB , & !out
                 QIN    ,QDIS   ,QSNOW  )                           !out

!     write(*,'(a20,10F15.5)') 'SFLX:RUNOFF=',RUNSRF*DT,RUNSUB*DT,EDIR*DT

! compute carbon budgets (carbon storages and co2 & bvoc fluxes)

   IF (DVEG == 2) THEN
    CALL CARBON (NSNOW  ,NSOIL  ,VEGTYP ,NROOT  ,DT     ,ZSOIL  , & !in
                 DZSNSO ,STC    ,SMC    ,TV     ,TG     ,PSN    , & !in
                 FOLN   ,SMCMAX ,BTRAN  ,APAR   ,FVEG   ,IGS    , & !in
                 TROOT  ,IST    ,IMONTH ,LAT    ,ipoint ,           & !in
                 LFMASS ,RTMASS ,STMASS ,WOOD   ,STBLCP ,FASTCP , & !inout
                 GPP    ,NPP    ,NEE    ,AUTORS ,HETERS ,TOTSC  , & !out
                 TOTLB  ,LAI    ,SAI    )                   !out
   END IF

! water and energy balance check

     CALL ERROR (SWDOWN ,FSA    ,FSR    ,FIRA   ,FSH    ,FCEV   , & !in
                 FGEV   ,FCTR   ,SSOIL  ,BEG_WB ,CANLIQ ,CANICE , & !in
                 SNEQV  ,WA     ,SMC    ,DZSNSO ,PRCP   ,ECAN   , & !in
                 ETRAN  ,EDIR   ,RUNSRF ,RUNSUB ,DT     ,NSOIL  , & !in
                 NSNOW  ,IST    ,ix     ,iy     ,ipoint )           !in

    IF(SWDOWN.NE.0.) THEN
      ALBEDO = FSR / SWDOWN
    ELSE
      ALBEDO = -999.9
    END IF

  END SUBROUTINE SFLX
! ==================================================================================================
  SUBROUTINE ATM (SFCPRS ,SFCTMP ,Q2     ,PRCP   ,SOLDN  ,COSZ   ,THAIR  , &
                  QAIR   ,EAIR   ,RHOAIR ,QPRECC ,QPRECL ,SOLAD  ,SOLAI  , &
                  SWDOWN )
! --------------------------------------------------------------------------------------------------
! re-process atmospheric forcing
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs

  REAL                          , INTENT(IN)  :: SFCPRS !pressure (pa)
  REAL                          , INTENT(IN)  :: SFCTMP !surface air temperature [k]
  REAL                          , INTENT(IN)  :: Q2     !mixing ratio (kg/kg)
  REAL                          , INTENT(IN)  :: SOLDN  !downward shortwave radiation (w/m2)
  REAL                          , INTENT(IN)  :: PRCP   !precipitation rate (kg m-2 s-1)
  REAL                          , INTENT(IN)  :: COSZ   !cosine solar zenith angle [0-1]

! outputs
```

```
  REAL                          , INTENT(OUT) :: THAIR  !potential temperature (k)
  REAL                          , INTENT(OUT) :: QAIR   !specific humidity (kg/kg) (q2/(1+q2))
  REAL                          , INTENT(OUT) :: EAIR   !vapor pressure air (pa)
  REAL, DIMENSION(       1:    2), INTENT(OUT) :: SOLAD  !incoming direct solar radiation (w/m2)
  REAL, DIMENSION(       1:    2), INTENT(OUT) :: SOLAI  !incoming diffuse solar radiation (w/m2)
  REAL                          , INTENT(OUT) :: QPRECC !convective precipitation (mm/s)
  REAL                          , INTENT(OUT) :: QPRECL !large-scale precipitation (mm/s)
  REAL                          , INTENT(OUT) :: RHOAIR !density air (kg/m3)
  REAL                          , INTENT(OUT) :: SWDOWN !downward solar filtered by sun angle [w/m2]

!locals

  REAL                                         :: PAIR   !atm bottom level pressure (pa)
! --------------------------------------------------------------------------------------------------

       PAIR   = SFCPRS                   ! atm bottom level pressure (pa)
       THAIR  = SFCTMP * (SFCPRS/PAIR)**(RAIR/CPAIR)
!       QAIR   = Q2 / (1.0+Q2)           ! mixing ratio to specific humidity [kg/kg]
       QAIR   = Q2                       ! GLDAS forcing: Q2 = specific humidity [kg/kg]
       EAIR   = QAIR*SFCPRS / (0.622+0.378*QAIR)
       RHOAIR = (SFCPRS-0.378*EAIR) / (RAIR*SFCTMP)

       QPRECC = 0.10 * PRCP          ! should be from the atmospheric model
       QPRECL = 0.90 * PRCP          ! should be from the atmospheric model

       IF(COSZ <= 0.) THEN
          SWDOWN = 0.
       ELSE
          SWDOWN = SOLDN
       END IF

       SOLAD(1) = SWDOWN*0.7*0.5     ! direct  vis
       SOLAD(2) = SWDOWN*0.7*0.5     ! direct  nir
       SOLAI(1) = SWDOWN*0.3*0.5     ! diffuse vis
       SOLAI(2) = SWDOWN*0.3*0.5     ! diffuse nir

  END SUBROUTINE ATM
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE PHONOLOGY (VEGTYP,IMONTH ,IDAY   ,SNOWH  ,TV     ,LAT   , & !in
                        LAI   ,SAI    ,TROOT  ,                   & !in
                        HTOP  ,ELAI   ,ESAI   ,IGS    )  !out
! --------------------------------------------------------------------------------------------------
! vegetation phenology considering vegeation canopy being buries by snow and evolution in time
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs
  INTEGER                       , INTENT(IN) :: VEGTYP !vegetation type
  INTEGER                       , INTENT(IN) :: IMONTH !month index
  INTEGER                       , INTENT(IN) :: IDAY   !day index
  REAL                          , INTENT(IN) :: SNOWH  !snow height [m]
  REAL                          , INTENT(IN) :: TV     !vegetation temperature (k)
  REAL                          , INTENT(IN) :: LAT    !latitude (radians)
  real                          , INTENT(IN) :: TROOT  !root-zone averaged temperature (k)
  REAL                        , INTENT(INOUT) :: LAI    !LAI, unadjusted for burying by snow
  REAL                        , INTENT(INOUT) :: SAI    !SAI, unadjusted for burying by snow

! outputs
  REAL                          ,INTENT(OUT) :: HTOP   !top of canopy layer (m)
  REAL                          ,INTENT(OUT) :: ELAI   !leaf area index, after burying by snow
  REAL                          ,INTENT(OUT) :: ESAI   !stem area index, after burying by snow
  REAL                          ,INTENT(OUT) :: IGS    !growing season index (0=off, 1=on)

! locals
```

```
  REAL                                  :: DB     !thickness of canopy buried by snow (m)
  REAL                                  :: FB     !fraction of canopy buried by snow
  REAL                                  :: SNOWHC !critical snow depth at which short vege
                                                  !is fully covered by snow

  INTEGER                               :: K       !index
  INTEGER                               :: NDAYN   !days in current year since jan 1: 1, ..., 365
  INTEGER                               :: NDAYS   !ndayn shifted 6 mon for SH
  INTEGER                               :: IT1,IT2 !interpolation months
  REAL                                  :: DAY     !current day of year
  REAL                                  :: WT1,WT2 !interpolation weights
  REAL                                  :: T       !current month (1.00, ..., 12.00)
  INTEGER                               :: NDAYPM(12)   !days per month
  DATA NDAYPM /31,28,31,30,31,30,31,31,30,31,30,31/
  SAVE NDAYPM
! --------------------------------------------------------------------------------------------------
! ndayn = days in current year since jan 1: 1, ..., 365
! ndays = ndayn shifted 6 mon for SH: 1 -> 183; 183 -> 365; 184 -> 1; 365 -> 182

   IF(DVEG == 1) THEN
     NDAYN = 0
     DO K = 1, IMONTH
         NDAYN = NDAYN + NDAYPM(K)
     END DO
     NDAYN = NDAYN - NDAYPM(IMONTH) + IDAY
     NDAYS = MOD (NDAYN-1+365/2, 365) + 1

     IF (LAT >= 0.) THEN
        DAY = NDAYN
     ELSE
        DAY = NDAYS
     END IF
     T = 12. * (DAY-0.5)/365.
     IT1 = T + 0.5
     IT2 = IT1 + 1
     WT1 = (IT1+0.5) - T
     WT2 = 1.-WT1
     IF (IT1 .LT.  1) IT1 = 12
     IF (IT2 .GT. 12) IT2 = 1

     LAI = WT1*LAIM(VEGTYP,IT1) + WT2*LAIM(VEGTYP,IT2)
     SAI = WT1*SAIM(VEGTYP,IT1) + WT2*SAIM(VEGTYP,IT2)
   END IF

     IF(VEGTYP == 16 .OR. VEGTYP == 19 .OR. VEGTYP ==24)  THEN
        LAI  = 0.
        SAI  = 0.
     END IF

!buried by snow

     DB = MIN( MAX(SNOWH - HVB(VEGTYP),0.), HVT(VEGTYP)-HVB(VEGTYP) )
     FB = DB / MAX(1.E-06,HVT(VEGTYP)-HVB(VEGTYP))

     IF(HVT(VEGTYP)> 0. .AND. HVT(VEGTYP) <= 0.5) THEN
       SNOWHC = HVT(VEGTYP)*EXP(-SNOWH/0.1)
       FB     = MIN(SNOWH,SNOWHC)/SNOWHC
     END IF

     ELAI =  LAI*(1.-FB)
     ESAI =  SAI*(1.-FB)

     IF (TV .GT. TMIN(VEGTYP)) THEN
         IGS = 1.
     ELSE
         IGS = 0.
     END IF

     HTOP = HVT(VEGTYP)
```

```
  END SUBROUTINE PHONOLOGY
! ==================================================================================================
  SUBROUTINE ERROR (SWDOWN ,FSA    ,FSR    ,FIRA   ,FSH    ,FCEV   , &
                    FGEV   ,FCTR   ,SSOIL  ,BEG_WB ,CANLIQ ,CANICE , &
                    SNEQV  ,WA     ,SMC    ,DZSNSO ,PRCP   ,ECAN   , &
                    ETRAN  ,EDIR   ,RUNSRF ,RUNSUB ,DT     ,NSOIL  , &
                    NSNOW  ,IST    ,ix     ,iy     ,ipoint )
! --------------------------------------------------------------------------------------------------
! check surface energy balance and water balance
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs
  INTEGER                          , INTENT(IN) :: NSNOW  !maximum no. of snow layers
  INTEGER                          , INTENT(IN) :: NSOIL  !number of soil layers
  INTEGER                          , INTENT(IN) :: IST    !surface type 1->soil; 2->lake
  INTEGER                          , INTENT(IN) :: IX     !grid index in e-w direction
  INTEGER                          , INTENT(IN) :: IY     !grid index in n-s direction
  INTEGER                          , INTENT(IN) :: ipoint !grid index
  REAL                             , INTENT(IN) :: SWDOWN !downward solar filtered by sun angle [w/m2]
  REAL                             , INTENT(IN) :: FSA    !total absorbed solar radiation (w/m2)
  REAL                             , INTENT(IN) :: FSR    !total reflected solar radiation (w/m2)
  REAL                             , INTENT(IN) :: FIRA   !total net longwave rad (w/m2)  [+ to atm]
  REAL                             , INTENT(IN) :: FSH    !total sensible heat (w/m2)     [+ to atm]
  REAL                             , INTENT(IN) :: FCEV   !canopy evaporation heat (w/m2) [+ to atm]
  REAL                             , INTENT(IN) :: FGEV   !ground evaporation heat (w/m2) [+ to atm]
  REAL                             , INTENT(IN) :: FCTR   !transpiration heat flux (w/m2) [+ to atm]
  REAL                             , INTENT(IN) :: SSOIL  !ground heat flux (w/m2)        [+ to soil]

  REAL                             , INTENT(IN) :: PRCP   !precipitation rate (kg m-2 s-1)
  REAL                             , INTENT(IN) :: ECAN   !evaporation of intercepted water (mm/s)
  REAL                             , INTENT(IN) :: ETRAN  !transpiration rate (mm/s)
  REAL                             , INTENT(IN) :: EDIR   !soil surface evaporation rate[mm/s]
  REAL                             , INTENT(IN) :: RUNSRF !surface runoff [mm/s]
  REAL                             , INTENT(IN) :: RUNSUB !baseflow (saturation excess) [mm/s]
  REAL                             , INTENT(IN) :: CANLIQ !intercepted liquid water (mm)
  REAL                             , INTENT(IN) :: CANICE !intercepted ice mass (mm)
  REAL                             , INTENT(IN) :: SNEQV  !snow water eqv. [mm]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: SMC    !soil moisture (ice + liq.) [m3/m3]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !snow/soil layer thickness [m]
  REAL                             , INTENT(IN) :: WA     !water storage in aquifer [mm]
  REAL                             , INTENT(IN) :: DT     !time step [sec]
  REAL                             , INTENT(IN) :: BEG_WB !water storage at begin of a timesetp [mm]

  INTEGER                                       :: IZ     !do-loop index
  REAL                                          :: END_WB !water storage at end of a timestep [mm]
  REAL                                          :: ERRWAT !error in water balance [mm/timestep]
  REAL                                          :: ERRENG !error in surface energy balance [w/m2]
  REAL                                          :: ERRSW  !error in shortwave radiation balance [w/m2]
! --------------------------------------------------------------------------------------------------

   ERRSW   = SWDOWN - (FSA + FSR)
   IF (ERRSW > 0.01) THEN            ! w/m2
        WRITE(*,*) 'ERRSW =',ERRSW
        STOP
   END IF

   ERRENG = FSA-(FIRA+FSH+FCEV+FGEV+FCTR+SSOIL)
   IF(ERRENG > 0.01) THEN
        WRITE(*,*) 'ERRENG =',ERRENG
        WRITE(*,'(i6,7F10.4)')ipoint,FSA,FIRA,FSH,FCEV,FGEV,FCTR,SSOIL
        STOP
   END IF

   IF (IST == 1) THEN                                     !soil
        END_WB = CANLIQ + CANICE + SNEQV + WA
        DO IZ = 1,NSOIL
          END_WB = END_WB + SMC(IZ) * DZSNSO(IZ) * 1000.
```

```
        END DO
        ERRWAT = END_WB-BEG_WB-(PRCP-ECAN-ETRAN-EDIR-RUNSRF-RUNSUB)*DT

        IF(ABS(ERRWAT) > 0.1) THEN
        WRITE(*,*) 'The model is losing(-)/gaining(+) fake water'
        WRITE(*,*) 'ERRWAT =',ERRWAT
        WRITE(*,'(i6,2f10.2,8f10.4)')ipoint,END_WB,BEG_WB,PRCP*DT,ECAN*DT,&
                                   EDIR*DT,ETRAN*DT,RUNSRF*DT,RUNSUB*DT
        STOP
        END IF
   ENDIF

 END SUBROUTINE ERROR
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE ENERGY (ICE    ,VEGTYP ,IST    ,ISC    ,NSNOW  ,NSOIL  , & !in
                     ISNOW  ,NROOT  ,DT     ,RHOAIR ,SFCPRS ,QAIR   , & !in
                     SFCTMP ,THAIR  ,LWDN   ,UU     ,VV     ,ZREF   , & !in
                     CO2AIR ,O2AIR  ,SOLAD  ,SOLAI  ,COSZ   ,IGS    , & !in
                     EAIR   ,HTOP   ,TBOT   ,ZBOT   ,ZSNSO  ,ZSOIL  , & !in
                     ELAI   ,ESAI   ,CSOIL  ,FWET   ,FOLN   ,Z0     , & !in
                     FVEG   ,                                         & !in
                     QSNOW  ,DZSNSO ,LAT    ,CANLIQ ,CANICE ,ipoint , & !in
                     IMELT  ,SNICEV ,SNLIQV ,EPORE  ,T2M    ,FSNO   , & !out
                     SAV    ,SAG    ,QMELT  ,FSA    ,FSR    ,TAUX   , & !out
                     TAUY   ,FIRA   ,FSH    ,FCEV   ,FGEV   ,FCTR   , & !out
                     TRAD   ,PSN    ,APAR   ,SSOIL  ,BTRANI ,BTRAN  , & !out
                     PONDING,TS     ,LATHEA ,                         & !out
                     TV     ,TG     ,STC    ,SNOWH  ,EAH    ,TAH    , & !inout
                     SNEQVO ,SNEQV  ,SH2O   ,SMC    ,SNICE  ,SNLIQ  , & !inout
                     ALBOLD ,CM     ,CH     )    !inout
! --------------------------------------------------------------------------------------------------
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
  USE RAD_PARAMETERS
! --------------------------------------------------------------------------------------------------
! we use different approaches to deal with subgrid features of radiation transfer and turbulent
! transfer. We use 'tile' approach to compute turbulent fluxes, while we use modified two-
! stream to compute radiation transfer. Tile approach, assemblying vegetation canopies together,
! may expose too much ground surfaces (either covered by snow or grass) to solar radiation. The
! modified two-stream assumes vegetation covers fully the gridcell but with gaps between tree
! crowns.
! --------------------------------------------------------------------------------------------------
! turbulence transfer : 'tile' approach to compute energy fluxes in vegetated fraction and
!                         bare fraction separately and then sum them up weighted by fraction
!                     --------------------------------------
!                    / O  O  O  O  O  O  O  O  /          /
!                   /  |  |  |  |  |  |  |  | /          /
!                  / O  O  O  O  O  O  O  /          /
!                 /  |  |  |tile1|  |  |  | /  tile2   /
!                / O  O  O  O  O  O  O  O  /  bare     /
!               /  |  |  | vegetated |  | /          /
!              / O  O  O  O  O  O  O  O  /          /
!             /  |  |  |  |  |  |  |  | /          /
!            --------------------------------------
! --------------------------------------------------------------------------------------------------
! radiation transfer : modified two-stream (Yang and Friedl, 2003, JGR; Niu ang Yang, 2004, JGR)
!                     --------------------------------------  two-stream treats leaves as
!                    /   O   O   O   O   O   O   O   O    /  cloud over the entire grid-cell,
!                   /    |   |   |   |   |   |   |   |   / while the modified two-stream
!                  /   O   O   O   O   O   O   O   O    / aggregates cloudy leaves into
!                 /    |   |   |   |   |   |   |   |   / tree crowns with gaps (as shown in
!                /   O   O   O   O   O   O   O   O    / the left figure). We assume these
!               /    |   |   |   |   |   |   |   |   / tree crowns are evenly distributed
!              /   O   O   O   O   O   O   O   O    / within the gridcell with 100% veg
!             /    |   |   |   |   |   |   |   |   / fraction, but with gaps. The 'tile'
!            -------------------------------------- approach overlaps too much shadows.
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
```

```fortran
! --------------------------------------------------------------------------------------------------
! inputs
  integer                           , INTENT(IN)    :: ipoint
  INTEGER                           , INTENT(IN)    :: ICE    !ice (ice = 1)
  INTEGER                           , INTENT(IN)    :: VEGTYP !vegetation physiology type
  INTEGER                           , INTENT(IN)    :: IST    !surface type: 1->soil; 2->lake
  INTEGER                           , INTENT(IN)    :: ISC    !soil color type (1-lighest; 8-darkest)
  INTEGER                           , INTENT(IN)    :: NSNOW  !maximum no. of snow layers
  INTEGER                           , INTENT(IN)    :: NSOIL  !number of soil layers
  INTEGER                           , INTENT(IN)    :: NROOT  !number of root layers
  INTEGER                           , INTENT(IN)    :: ISNOW  !actual no. of snow layers
  REAL                              , INTENT(IN)    :: DT     !time step [sec]
  REAL                              , INTENT(IN)    :: QSNOW  !snowfall on the ground (mm/s)
  REAL                              , INTENT(IN)    :: RHOAIR !density air (kg/m3)
  REAL                              , INTENT(IN)    :: EAIR   !vapor pressure air (pa)
  REAL                              , INTENT(IN)    :: SFCPRS !pressure (pa)
  REAL                              , INTENT(IN)    :: QAIR   !specific humidity (kg/kg)
  REAL                              , INTENT(IN)    :: SFCTMP !air temperature (k)
  REAL                              , INTENT(IN)    :: THAIR  !potential temperature (k)
  REAL                              , INTENT(IN)    :: LWDN   !downward longwave radiation (w/m2)
  REAL                              , INTENT(IN)    :: UU     !wind speed in e-w dir (m/s)
  REAL                              , INTENT(IN)    :: VV     !wind speed in n-s dir (m/s)
  REAL    , DIMENSION(       1:    2), INTENT(IN)    :: SOLAD  !incoming direct solar rad. (w/m2)
  REAL    , DIMENSION(       1:    2), INTENT(IN)    :: SOLAI  !incoming diffuse solar rad. (w/m2)
  REAL                              , INTENT(IN)    :: COSZ   !cosine solar zenith angle (0-1)
  REAL                              , INTENT(IN)    :: ELAI   !LAI adjusted for burying by snow
  REAL                              , INTENT(IN)    :: ESAI   !LAI adjusted for burying by snow
  REAL                              , INTENT(IN)    :: CSOIL  !vol. soil heat capacity [j/m3/k]
  REAL                              , INTENT(IN)    :: FWET   !fraction of canopy that is wet [-]
  REAL                              , INTENT(IN)    :: HTOP   !top of canopy layer (m)
  REAL                              , INTENT(IN)    :: Z0     !roughness length (m)
  REAL                              , INTENT(IN)    :: FVEG   !greeness vegetation fraction (-)
  REAL                              , INTENT(IN)    :: LAT    !latitude (radians)
  REAL                              , INTENT(IN)    :: CANLIQ !canopy-intercepted liquid water (mm)
  REAL                              , INTENT(IN)    :: CANICE !canopy-intercepted ice mass (mm)
  REAL                              , INTENT(IN)    :: FOLN   !foliage nitrogen (%)
  REAL                              , INTENT(IN)    :: CO2AIR !atmospheric co2 concentration (pa)
  REAL                              , INTENT(IN)    :: O2AIR  !atmospheric o2 concentration (pa)
  REAL                              , INTENT(IN)    :: IGS    !growing season index (0=off, 1=on)

  REAL                              , INTENT(IN)    :: ZREF   !reference height (m)
  REAL                              , INTENT(IN)    :: TBOT   !bottom condition for soil temp. (k)
  REAL                              , INTENT(IN)    :: ZBOT   !depth for TBOT [m]
  REAL    , DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)    :: ZSNSO  !layer-bottom depth from snow surf [m]
  REAL    , DIMENSION(       1:NSOIL), INTENT(IN)    :: ZSOIL  !layer-bottom depth from soil surf [m]
  REAL    , DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)    :: DZSNSO !depth of snow & soil layer-bottom [m]

! outputs
  INTEGER, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT)   :: IMELT  !phase change index [1-melt; 2-freeze]
  REAL    , DIMENSION(-NSNOW+1:    0), INTENT(OUT)   :: SNICEV !partial volume ice [m3/m3]
  REAL    , DIMENSION(-NSNOW+1:    0), INTENT(OUT)   :: SNLIQV !partial volume liq. water [m3/m3]
  REAL    , DIMENSION(-NSNOW+1:    0), INTENT(OUT)   :: EPORE  !effective porosity [m3/m3]
  REAL                              , INTENT(OUT)   :: FSNO   !snow cover fraction (-)
  REAL                              , INTENT(OUT)   :: QMELT  !snowmelt [mm/s]
  REAL                              , INTENT(OUT)   :: PONDING!pounding at ground [mm]
  REAL                              , INTENT(OUT)   :: SAV    !solar rad. absorbed by veg. (w/m2)
  REAL                              , INTENT(OUT)   :: SAG    !solar rad. absorbed by ground (w/m2)
  REAL                              , INTENT(OUT)   :: FSA    !tot. absorbed solar radiation (w/m2)
  REAL                              , INTENT(OUT)   :: FSR    !tot. reflected solar radiation (w/m2)
  REAL                              , INTENT(OUT)   :: TAUX   !wind stress: e-w (n/m2)
  REAL                              , INTENT(OUT)   :: TAUY   !wind stress: n-s (n/m2)
  REAL                              , INTENT(OUT)   :: FIRA   !total net LW. rad (w/m2)    [+ to atm]
  REAL                              , INTENT(OUT)   :: FSH    !total sensible heat (w/m2)  [+ to atm]
  REAL                              , INTENT(OUT)   :: FCEV   !canopy evaporation (w/m2)   [+ to atm]
  REAL                              , INTENT(OUT)   :: FGEV   !ground evaporation (w/m2)   [+ to atm]
  REAL                              , INTENT(OUT)   :: FCTR   !transpiration (w/m2)        [+ to atm]
  REAL                              , INTENT(OUT)   :: TRAD   !radiative temperature (k)
  REAL                              , INTENT(OUT)   :: T2M    !2 m height air temperature (k)
  REAL                              , INTENT(OUT)   :: PSN    !total photosyn. (umolco2/m2/s) [+]
```

```
  REAL                               , INTENT(OUT)   :: APAR   !total photosyn. active energy (w/m2)
  REAL    , DIMENSION(       1:NSOIL), INTENT(OUT)   :: SSOIL  !ground heat flux (w/m2)   [+ to soil]
  REAL                               , INTENT(OUT)   :: BTRANI !soil water transpiration factor (0-1)
  REAL                               , INTENT(OUT)   :: BTRAN  !soil water transpiration factor (0-1)
  REAL                               , INTENT(OUT)   :: LATHEA !latent heat vap./sublimation (j/kg)

! input & output
  REAL                               , INTENT(INOUT) :: TS     !surface temperature (k)
  REAL                               , INTENT(INOUT) :: TV     !vegetation temperature (k)
  REAL                               , INTENT(INOUT) :: TG     !ground temperature (k)
  REAL    , DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC    !snow/soil temperature [k]
  REAL                               , INTENT(INOUT) :: SNOWH  !snow height [m]
  REAL                               , INTENT(INOUT) :: SNEQV  !snow mass (mm)
  REAL                               , INTENT(INOUT) :: SNEQVO !snow mass at last time step (mm)
  REAL    , DIMENSION(       1:NSOIL), INTENT(INOUT) :: SH2O   !liquid soil moisture [m3/m3]
  REAL    , DIMENSION(       1:NSOIL), INTENT(INOUT) :: SMC    !soil moisture (ice + liq.) [m3/m3]
  REAL    , DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNICE  !snow ice mass (kg/m2)
  REAL    , DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNLIQ  !snow liq mass (kg/m2)
  REAL                               , INTENT(INOUT) :: EAH    !canopy air vapor pressure (pa)
  REAL                               , INTENT(INOUT) :: TAH    !canopy air temperature (k)
  REAL                               , INTENT(INOUT) :: ALBOLD !snow albedo at last time step(CLASS type)
  REAL                               , INTENT(INOUT) :: CM     !momentum drag coefficient
  REAL                               , INTENT(INOUT) :: CH     !sensible heat exchange coefficient

! local
  INTEGER                                            :: IZ     !do-loop index
  LOGICAL                                            :: VEG    !true if vegetated surface
  REAL                                               :: UR     !wind speed at height ZLVL (m/s)
  REAL                                               :: ZLVL   !reference height (m)
  REAL                                               :: FSUN   !sunlit fraction of canopy [-]
  REAL                                               :: RB     !leaf boundary layer resistance (s/m)
  REAL                                               :: RSURF  !ground surface resistance (s/m)
  REAL                                               :: BEVAP  !soil water evaporation factor (0- 1)
  REAL                                               :: MOL    !Monin-Obukhov length (m)
  REAL                                               :: VAI    !sum of LAI  + stem area index [m2/m2]
  REAL                                               :: CWP    !canopy wind extinction parameter
  REAL                                               :: ZPD    !zero plane displacement (m)
  REAL                                               :: Z0M    !z0 momentum (m)
  REAL                                               :: ZPDG   !zero plane displacement (m)
  REAL                                               :: Z0MG   !z0 momentum, ground (m)
  REAL                                               :: EMV    !vegetation emissivity
  REAL                                               :: EMG    !ground emissivity
  REAL                                               :: FIRE   !emitted IR (w/m2)

  REAL                                               :: LAISUN !sunlit leaf area index (m2/m2)
  REAL                                               :: LAISHA !shaded leaf area index (m2/m2)
  REAL                                               :: PSNSUN !sunlit photosynthesis (umolco2/m2/s)
  REAL                                               :: PSNSHA !shaded photosynthesis (umolco2/m2/s)
  REAL                                               :: RSSUN  !sunlit stomatal resistance (s/m)
  REAL                                               :: RSSHA  !shaded stomatal resistance (s/m)
  REAL                                               :: PARSUN !par absorbed per sunlit LAI (w/m2)
  REAL                                               :: PARSHA !par absorbed per shaded LAI (w/m2)

  REAL, DIMENSION(-NSNOW+1:NSOIL)                    :: FACT   !temporary used in phase change
  REAL, DIMENSION(-NSNOW+1:NSOIL)                    :: DF     !thermal conductivity [w/m/k]
  REAL, DIMENSION(-NSNOW+1:NSOIL)                    :: HCPCT  !heat capacity [j/m3/k]
  REAL                                               :: BDSNO  !bulk density of snow (kg/m3)
  REAL                                               :: FMELT  !melting factor for snow cover frac
  REAL                                               :: GX     !temporary variable
  REAL                                               :: EFLXB  !energy influx from soil bot. (w/m2)
  REAL, DIMENSION(-NSNOW+1:NSOIL)                    :: PHI    !light through water (w/m2)
  REAL                                               :: GAMMA  !psychrometric constant (pa/k)
  REAL                                               :: PSI    !surface layer soil matrix potential (m)
  REAL                                               :: RHSUR  !raltive humidity in surface soil/snow air
    space (-)

! temperature and fluxes over vegetated fraction

  REAL                                               :: TAUXV  !wind stress: e-w dir [n/m2]
```

```
  REAL                                                :: TAUYV  !wind stress: n-s dir [n/m2]
  REAL                                                :: IRC    !canopy net LW rad. [w/m2] [+ to atm]
  REAL                                                :: IRG    !ground net LW rad. [w/m2] [+ to atm]
  REAL                                                :: SHC    !canopy sen. heat [w/m2]   [+ to atm]
  REAL                                                :: SHG    !ground sen. heat [w/m2]   [+ to atm]
  REAL                                                :: EVC    !canopy evap. heat [w/m2]  [+ to atm]
  REAL                                                :: EVG    !ground evap. heat [w/m2]  [+ to atm]
  REAL                                                :: TR     !transpiration heat [w/m2] [+ to atm]
  REAL                                                :: GHV    !ground heat flux [w/m2]  [+ to soil]
  REAL                                                :: TGV    !ground surface temp. [k]
  REAL                                                :: T2MV   !2-m air temperature [k]
  REAL                                                :: CMV    !momentum drag coefficient
  REAL                                                :: CHV    !sensible heat exchange coefficient

! temperature and fluxes over bare soil fraction

  REAL                                                :: TAUXB  !wind stress: e-w dir [n/m2]
  REAL                                                :: TAUYB  !wind stress: n-s dir [n/m2]
  REAL                                                :: IRB    !net longwave rad. [w/m2] [+ to atm]
  REAL                                                :: SHB    !sensible heat [w/m2]     [+ to atm]
  REAL                                                :: EVB    !evaporation heat [w/m2]  [+ to atm]
  REAL                                                :: GHB    !ground heat flux [w/m2] [+ to soil]
  REAL                                                :: TGB    !ground surface temp. [k]
  REAL                                                :: T2MB   !2-m air temp. [k]
  REAL                                                :: CMB    !momentum drag coefficient
  REAL                                                :: CHB    !sensible heat exchange coefficient

  REAL, PARAMETER                     :: MPE    = 1.E-6
  REAL, PARAMETER                     :: PSIWLT = -150.  !metric potential for wilting point (m)

! ----------------------------------------------------------------------------------------------------
! initialize fluxes from veg. fraction

    TAUXV     = 0.
    TAUYV     = 0.
    IRC       = 0.
    SHC       = 0.
    IRG       = 0.
    SHG       = 0.
    EVG       = 0.
    EVC       = 0.
    TR        = 0.
    GHV       = 0.
    PSNSUN    = 0.
    PSNSHA    = 0.

! wind speed at reference height: ur >= 1

    UR = MAX( SQRT(UU**2.+VV**2.), 1. )

! vegetated or non-vegetated

    VAI = ELAI + ESAI
    VEG = .FALSE.
    IF(VAI > 0.) VEG = .TRUE.

! ground snow cover fraction [Niu and Yang, 2007, JGR]

     FSNO = 0.
     IF(SNOWH.GT.0.)  THEN
         BDSNO    = SNEQV / SNOWH
         FMELT    = (BDSNO/100.)**M
         FSNO     = TANH( SNOWH /(2.5* Z0 * FMELT))
     ENDIF

! ground roughness length

     IF(IST == 2) THEN
       IF(TG .LE. TFRZ) THEN
```

```
           Z0MG = 0.01 * (1.0-FSNO) + FSNO * Z0SNO
         ELSE
           Z0MG = 0.01
         END IF
       ELSE
         Z0MG = Z0 * (1.0-FSNO) + FSNO * Z0SNO
       END IF

! roughness length and displacement height

       ZPDG  = SNOWH
       IF(VEG) THEN
          Z0M  = Z0MVT(VEGTYP)
          ZPD  = 0.65 * HTOP
          IF(SNOWH.GT.ZPD) ZPD  = SNOWH
       ELSE
          Z0M  = Z0MG
          ZPD  = ZPDG
       END IF

       ZLVL = MAX(ZPD,HTOP) + ZREF
       IF(ZPDG >= ZLVL) ZLVL = ZPDG + ZREF
!      UR   = UR*LOG(ZLVL/Z0M)/LOG(10./Z0M)       !input UR is at 10m

! canopy wind absorption coeffcient

       CWP = CWPVT(VEGTYP)

! Thermal properties of soil, snow, lake, and frozen soil

  CALL THERMOPROP (NSOIL   ,NSNOW   ,ISNOW   ,IST     ,DZSNSO  , & !in
                   DT      ,SNOWH   ,SNICE   ,SNLIQ   ,CSOIL   , & !in
                   SMC     ,SH2O    ,TG      ,STC     ,UR      , & !in
                   LAT     ,Z0M     ,ZLVL    ,                   & !in
                   DF      ,HCPCT   ,SNICEV  ,SNLIQV  ,EPORE   , & !out
                   FACT    )                                       !out

! Solar radiation: absorbed & reflected by the ground and canopy

  CALL  RADIATION (VEGTYP  ,IST     ,ISC     ,ICE     ,NSOIL   , & !in
                   SNEQVO  ,SNEQV   ,DT      ,COSZ    ,SNOWH   , & !in
                   TG      ,TV      ,FSNO    ,QSNOW   ,FWET    , & !in
                   ELAI    ,ESAI    ,SMC     ,SOLAD   ,SOLAI   , & !in
                   FVEG    ,ipoint  ,                            & !in
                   ALBOLD  ,                                     & !inout
                   FSUN    ,LAISUN  ,LAISHA  ,PARSUN  ,PARSHA  , & !out
                   SAV     ,SAG     ,FSR     ,FSA     )            !out

! vegetation and ground emissivity

       EMV = 1. - EXP(-(ELAI+ESAI)/1.0)
       IF (ICE == 1) THEN
         EMG = 0.98*(1.-FSNO) + 1.0*FSNO
       ELSE
         EMG = EG(IST)*(1.-FSNO) + 1.0*FSNO
       END IF

! soil moisture factor controlling stomatal resistance

       BTRAN = 0.

       IF(IST ==1 ) THEN
         DO IZ = 1, NROOT
            IF(OPT_BTR == 1) then                  ! Noah
              GX    = (SH2O(IZ)-SMCWLT) / (SMCREF-SMCWLT)
            END IF
            IF(OPT_BTR == 2) then                  ! CLM
              PSI   = MAX(PSIWLT,-PSISAT*(MAX(0.01,SH2O(IZ))/SMCMAX)**(-BEXP) )
              GX    = (1.-PSI/PSIWLT)/(1.+PSISAT/PSIWLT)
```

```
          END IF
          IF(OPT_BTR == 3) then                  ! SSiB
            PSI    = MAX(PSIWLT,-PSISAT*(MAX(0.01,SH2O(IZ))/SMCMAX)**(-BEXP) )
            GX     = 1.-EXP(-5.8*(LOG(PSIWLT/PSI)))
          END IF

          GX = MIN(1.,MAX(0.,GX))
          BTRANI(IZ) = MAX(MPE,DZSNSO(IZ) / (-ZSOIL(NROOT)) * GX)
          BTRAN      = BTRAN + BTRANI(IZ)
       END DO
       BTRAN = MAX(MPE,BTRAN)

       BTRANI(1:NROOT) = BTRANI(1:NROOT)/BTRAN
     END IF

! soil surface resistence for ground evap.

     BEVAP = MAX(0.0,SH2O(1)/SMCMAX)
     IF(IST == 2) THEN
       RSURF = 1.          ! avoid being divided by 0
       RHSUR = 1.0
     ELSE
!niu       RSURF = FSNO * 1.  + (1.-FSNO)* EXP(8.25-4.225*BEVAP) !Sellers (1992)
       RSURF = FSNO * 1.  + (1.-FSNO)* EXP(8.25-6.0*BEVAP) !adjusted to decrease RSURF for wet soil
       IF(SH2O(1) < 0.01 .and. SNOWH == 0.) RSURF = 1.E6
       PSI   = -PSISAT*(MAX(0.01,SH2O(1))/SMCMAX)**(-BEXP)
       RHSUR = FSNO + (1.-FSNO) * EXP(PSI*GRAV/(RW*TG))
     END IF

! set psychrometric constant

     IF (SFCTMP .GT. TFRZ) THEN
        LATHEA = HVAP
     ELSE
        LATHEA = HSUB
     END IF
     GAMMA = CPAIR*SFCPRS/(0.622*LATHEA)

! Surface temperatures of the ground and canopy and energy fluxes

    IF (VEG) THEN
    TGV = TG
    CMV = CM
    CHV = CH
    CALL VEGE_FLUX (NSNOW    ,NSOIL    ,ISNOW    ,VEGTYP   ,VEG      , & !in
                    DT       ,SAV      ,SAG      ,LWDN     ,UR       , & !in
                    UU       ,VV       ,SFCTMP   ,THAIR    ,QAIR     , & !in
                    EAIR     ,RHOAIR   ,SNOWH    ,VAI      ,GAMMA    , & !in
                    FWET     ,LAISUN   ,LAISHA   ,CWP      ,DZSNSO   , & !in
                    HTOP     ,ZLVL     ,ZPD      ,Z0M      ,FVEG     , & !in
                    Z0MG     ,EMV      ,EMG      ,CANLIQ             , & !in
                    CANICE   ,STC      ,DF       ,RSSUN    ,RSSHA    , & !in
                    RSURF    ,LATHEA   ,PARSUN   ,PARSHA   ,IGS      , & !in
                    FOLN     ,CO2AIR   ,O2AIR    ,BTRAN    ,SFCPRS   , & !in
                    RHSUR    ,ipoint   ,                               & !in
                    EAH      ,TAH      ,TV       ,TGV      ,CMV      , & !inout
                    CHV      ,                                         & !inout
                    TAUXV    ,TAUYV    ,IRG      ,IRC      ,SHG      , & !out
                    SHC      ,EVG      ,EVC      ,TR       ,GHV      , & !out
                    T2MV     ,PSNSUN   ,PSNSHA   )                       !out
    END IF

    TGB = TG
    CMB = CM
    CHB = CH
    CALL BARE_FLUX (NSNOW    ,NSOIL    ,ISNOW    ,DT       ,SAG      , & !in
                    LWDN     ,UR       ,UU       ,VV       ,SFCTMP   , & !in
                    THAIR    ,QAIR     ,EAIR     ,RHOAIR   ,SNOWH    , & !in
                    DZSNSO   ,ZLVL     ,ZPDG     ,Z0MG     ,           & !in
```

```
                    EMG     ,STC     ,DF      ,RSURF   ,LATHEA  , & !in
                    GAMMA   ,RHSUR   ,ipoint  ,                   & !in
                    TGB     ,CMB     ,CHB     ,                   & !inout
                    TAUXB   ,TAUYB   ,IRB     ,SHB     ,EVB     , & !out
                    GHB     ,T2MB    )                              !out

!energy balance at vege canopy: SAV          =(IRC+SHC+EVC+TR)     *FVEG  at   FVEG
!energy balance at vege ground: SAG*    FVEG =(IRG+SHG+EVG+GHV)    *FVEG  at   FVEG
!energy balance at bare ground: SAG*(1.-FVEG)=(IRB+SHB+EVB+GHB)*(1.-FVEG) at 1-FVEG

    IF (VEG) THEN
        TAUX  = FVEG * TAUXV     + (1.0 - FVEG) * TAUXB
        TAUY  = FVEG * TAUYV     + (1.0 - FVEG) * TAUYB
        FIRA  = FVEG * IRG       + (1.0 - FVEG) * IRB       + IRC
        FSH   = FVEG * SHG       + (1.0 - FVEG) * SHB       + SHC
        FGEV  = FVEG * EVG       + (1.0 - FVEG) * EVB
        SSOIL = FVEG * GHV       + (1.0 - FVEG) * GHB
        FCEV  = EVC
        FCTR  = TR
        TG    = FVEG * TGV       + (1.0 - FVEG) * TGB
        T2M   = FVEG * T2MV      + (1.0 - FVEG) * T2MB
        TS    = FVEG * TV        + (1.0 - FVEG) * TGB
        CM    = FVEG * CMV       + (1.0 - FVEG) * CMB      ! better way to average?
        CH    = FVEG * CHV       + (1.0 - FVEG) * CHB
    ELSE
        TAUX  = TAUXB
        TAUY  = TAUYB
        FIRA  = IRB
        FSH   = SHB
        FGEV  = EVB
        SSOIL = GHB
        TG    = TGB
        T2M   = T2MB
        FCEV  = 0.
        FCTR  = 0.
        TS    = TG
        CM    = CMB
        CH    = CHB
    END IF

    FIRE = LWDN + FIRA

    IF(FIRE <=0.) THEN
       WRITE(6,*) 'emitted longwave <0; skin T maybe wrong due to inconsistent'
       WRITE(6,*) 'input of SHDFAC with LAI'
       WRITE(6,*) ipoint, 'SHDFAC=',FVEG,'VAI=',VAI,'TV=',TV,'TG=',TG
       WRITE(6,*) 'LWDN=',LWDN,'FIRA=',FIRA,'SNOWH=',SNOWH
       STOP
    END IF

    TRAD = (FIRE/SB)**0.25
    APAR = PARSUN*LAISUN + PARSHA*LAISHA
    PSN  = PSNSUN*LAISUN + PSNSHA*LAISHA

! 3L snow & 4L soil temperatures

    CALL TSNOSOI (ICE     ,NSOIL   ,NSNOW   ,ISNOW   ,IST     , & !in
                  TBOT    ,ZSNSO   ,SSOIL   ,DF      ,HCPCT   , & !in
                  ZBOT    ,SAG     ,DT      ,SNOWH   ,DZSNSO  , & !in
                  TG      ,ipoint  ,                            & !in
                  STC     )                                       !inout

! adjusting snow surface temperature

     IF(OPT_STC == 2) THEN
      IF (SNOWH > 0.05 .AND. TG > TFRZ) THEN
        TGV = TFRZ
        TGB = TFRZ
          IF (VEG) THEN
```

```
            TG     = FVEG * TGV        + (1.0 - FVEG) * TGB
            TS     = FVEG * TV         + (1.0 - FVEG) * TGB
         ELSE
            TG     = TGB
            TS     = TGB
         END IF
     END IF
    END IF

! Energy released or consumed by snow & frozen soil

 CALL PHASECHANGE (NSNOW    ,NSOIL    ,ISNOW    ,DT       ,FACT    , & !in
                   DZSNSO   ,HCPCT    ,IST      ,ipoint   ,          & !in
                   STC      ,SNICE    ,SNLIQ    ,SNEQV    ,SNOWH   , & !inout
                   SMC      ,SH2O     ,                              & !inout
                   QMELT    ,IMELT    ,PONDING )                       !out

  END SUBROUTINE ENERGY
! ==================================================================================================
  SUBROUTINE THERMOPROP (NSOIL   ,NSNOW   ,ISNOW   ,IST     ,DZSNSO  , & !in
                         DT      ,SNOWH   ,SNICE   ,SNLIQ   ,CSOIL   , & !in
                         SMC     ,SH2O    ,TG      ,STC     ,UR      , & !in
                         LAT     ,Z0M     ,ZLVL    ,                   & !in
                         DF      ,HCPCT   ,SNICEV  ,SNLIQV  ,EPORE   , & !out
                         FACT    )                                       !out
! --------------------------------------------------------------------------------------------------
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs
  INTEGER                        , INTENT(IN)  :: NSOIL   !number of soil layers
  INTEGER                        , INTENT(IN)  :: NSNOW   !maximum no. of snow layers
  INTEGER                        , INTENT(IN)  :: ISNOW   !actual no. of snow layers
  INTEGER                        , INTENT(IN)  :: IST     !surface type
  REAL                           , INTENT(IN)  :: DT      !time step [s]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN)  :: SNICE   !snow ice mass (kg/m2)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN)  :: SNLIQ   !snow liq mass (kg/m2)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: DZSNSO  !thickness of snow/soil layers [m]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)  :: SMC     !soil moisture (ice + liq.) [m3/m3]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)  :: SH2O    !liquid soil moisture [m3/m3]
  REAL                           , INTENT(IN)  :: SNOWH   !snow height [m]
  REAL                           , INTENT(IN)  :: CSOIL   !vol. soil heat capacity [j/m3/k]
  REAL,                            INTENT(IN)  :: TG      !surface temperature (k)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: STC     !snow/soil/lake temp. (k)
  REAL,                            INTENT(IN)  :: UR      !wind speed at ZLVL (m/s)
  REAL,                            INTENT(IN)  :: LAT     !latitude (radians)
  REAL,                            INTENT(IN)  :: Z0M     !roughness length (m)
  REAL,                            INTENT(IN)  :: ZLVL    !reference height (m)

! outputs
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: DF      !thermal conductivity [w/m/k]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: HCPCT   !heat capacity [j/m3/k]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: SNICEV  !partial volume of ice [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: SNLIQV  !partial volume of liquid water [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: EPORE   !effective porosity [m3/m3]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: FACT    !computing energy for phase change
! --------------------------------------------------------------------------------------------------
! locals

  INTEGER :: IZ
  REAL, DIMENSION(-NSNOW+1:    0)              :: CVSNO   !volumetric specific heat (j/m3/k)
  REAL, DIMENSION(-NSNOW+1:    0)              :: TKSNO   !snow thermal conductivity (j/m3/k)
  REAL, DIMENSION(       1:NSOIL)              :: SICE    !soil ice content
! --------------------------------------------------------------------------------------------------

! compute snow thermal conductivity and heat capacity

    CALL CSNOW (ISNOW   ,NSNOW   ,NSOIL   ,SNICE   ,SNLIQ   ,DZSNSO  , & !in
                TKSNO   ,CVSNO   ,SNICEV  ,SNLIQV  ,EPORE   )   !out
```

```
    DO IZ = ISNOW+1, 0
      DF    (IZ) = TKSNO(IZ)
      HCPCT(IZ) = CVSNO(IZ)
    END DO

! compute soil thermal properties

    DO  IZ = 1, NSOIL
       SICE(IZ)   = SMC(IZ) - SH2O(IZ)
       HCPCT(IZ) = SH2O(IZ)*CWAT + (1.0-SMCMAX)*CSOIL &
                + (SMCMAX-SMC(IZ))*CPAIR + SICE(IZ)*CICE
       CALL TDFCND (DF(IZ), SMC(IZ), SH2O(IZ))
    END DO

! heat flux reduction effect from the overlying green canopy, adapted from
! section 2.1.2 of Peters-Lidard et al. (1997, JGR, VOL 102(D4)).
! not in use because of the separation of the canopy layer from the ground.
! but this may represent the effects of leaf litter (Niu comments)
!       DF1 = DF1 * EXP (SBETA * SHDFAC)

! compute lake thermal properties
! (no consideration of turbulent mixing for this version)

    IF(IST == 2) THEN
       DO IZ = 1, NSOIL
         IF(STC(IZ) > TFRZ) THEN
            HCPCT(IZ) = CWAT
            DF(IZ)     = TKWAT  !+ KEDDY * CWAT
         ELSE
            HCPCT(IZ) = CICE
            DF(IZ)     = TKICE
         END IF
       END DO
    END IF

! combine a temporary variable used for melting/freezing of snow and frozen soil

    DO IZ = ISNOW+1,NSOIL
     FACT(IZ) = DT/(HCPCT(IZ)*DZSNSO(IZ))
    END DO

! snow/soil interface

    IF(ISNOW == 0) THEN
       DF(1) = (DF(1)*DZSNSO(1)+0.35*SNOWH)       / (SNOWH    +DZSNSO(1))
    ELSE
       DF(1) = (DF(1)*DZSNSO(1)+DF(0)*DZSNSO(0)) / (DZSNSO(0)+DZSNSO(1))
    END IF

  END SUBROUTINE THERMOPROP
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE CSNOW (ISNOW   ,NSNOW   ,NSOIL   ,SNICE   ,SNLIQ   ,DZSNSO  , & !in
                    TKSNO   ,CVSNO   ,SNICEV  ,SNLIQV  ,EPORE   )    !out
! --------------------------------------------------------------------------------------------------
! Snow bulk density,volumetric capacity, and thermal conductivity
!---------------------------------------------------------------------------------------------------
  IMPLICIT NONE
!---------------------------------------------------------------------------------------------------
! inputs

  INTEGER,                          INTENT(IN) :: ISNOW  !number of snow layers (-)
  INTEGER                         , INTENT(IN) :: NSNOW  !maximum no. of snow layers
  INTEGER                         , INTENT(IN) :: NSOIL  !number of soil layers
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN) :: SNICE  !snow ice mass (kg/m2)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN) :: SNLIQ  !snow liq mass (kg/m2)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !snow/soil layer thickness [m]
```

```
! outputs

  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: CVSNO  !volumetric specific heat (j/m3/k)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: TKSNO  !thermal conductivity (w/m/k)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: SNICEV !partial volume of ice [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: SNLIQV !partial volume of liquid water [m3/m3]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(OUT) :: EPORE  !effective porosity [m3/m3]

! locals

  INTEGER :: IZ
  REAL, DIMENSION(-NSNOW+1:    0) :: BDSNOI  !bulk density of snow(kg/m3)

!---------------------------------------------------------------------------------------------------
! thermal capacity of snow

  DO IZ = ISNOW+1, 0
      SNICEV(IZ)   = MIN(1., SNICE(IZ)/(DZSNSO(IZ)*DENICE) )
      EPORE(IZ)    = 1. - SNICEV(IZ)
      SNLIQV(IZ)   = MIN(EPORE(IZ),SNLIQ(IZ)/(DZSNSO(IZ)*DENH2O))
  ENDDO

  DO IZ = ISNOW+1, 0
      BDSNOI(IZ) = (SNICE(IZ)+SNLIQ(IZ))/DZSNSO(IZ)
      CVSNO(IZ) = CICE*SNICEV(IZ)+CWAT*SNLIQV(IZ)
!      CVSNO(IZ) = 0.525E06                          ! constant
  enddo

! thermal conductivity of snow

  DO IZ = ISNOW+1, 0
     TKSNO(IZ) = 3.2217E-6*BDSNOI(IZ)**2.           ! Stieglitz(yen,1965)
!    TKSNO(IZ) = 2E-2+2.5E-6*BDSNOI(IZ)*BDSNOI(IZ)   ! Anderson, 1976
!    TKSNO(IZ) = 0.35                                ! constant
!    TKSNO(IZ) = 2.576E-6*BDSNOI(IZ)**2. + 0.074    ! Verseghy (1991)
!    TKSNO(IZ) = 2.22*(BDSNOI(IZ)/1000.)**1.88      ! Douvill(Yen, 1981)
  ENDDO

  END SUBROUTINE CSNOW
!===================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE TDFCND ( DF, SMC, SH2O)
! --------------------------------------------------------------------------------------------------
! Calculate thermal diffusivity and conductivity of the soil.
! Peters-Lidard approach (Peters-Lidard et al., 1998)
! --------------------------------------------------------------------------------------------------
! Code history:
! June 2001 changes: frozen soil condition.
! --------------------------------------------------------------------------------------------------
    IMPLICIT NONE
    REAL, INTENT(IN)       :: SMC    ! total soil water
    REAL, INTENT(IN)       :: SH2O   ! liq. soil water
    REAL, INTENT(OUT)      :: DF     ! thermal diffusivity

! local variables
    REAL  :: AKE
    REAL  :: GAMMD
    REAL  :: THKDRY
    REAL  :: THKO     ! thermal conductivity for other soil components
    REAL  :: THKQTZ   ! thermal conductivity for quartz
    REAL  :: THKSAT   !
    REAL  :: THKS     ! thermal conductivity for the solids
    REAL  :: THKW     ! water thermal conductivity
    REAL  :: SATRATIO
    REAL  :: XU
    REAL  :: XUNFROZ
! --------------------------------------------------------------------------------------------------
! We now get quartz as an input argument (set in routine redprm):
!      DATA QUARTZ /0.82, 0.10, 0.25, 0.60, 0.52,
```

```
!     &             0.35, 0.60, 0.40, 0.82/
! ----------------------------------------------------------------------------------------------------
! If the soil has any moisture content compute a partial sum/product
! otherwise use a constant value which works well with most soils
! ----------------------------------------------------------------------------------------------------
!  QUARTZ ....QUARTZ CONTENT (SOIL TYPE DEPENDENT)
! ----------------------------------------------------------------------------------------------------
! USE AS IN PETERS-LIDARD, 1998 (MODIF. FROM JOHANSEN, 1975).

!                                  PABLO GRUNMANN, 08/17/98
! Refs.:
!      Farouki, O.T.,1986: Thermal properties of soils. Series on Rock
!              and Soil Mechanics, Vol. 11, Trans Tech, 136 pp.
!      Johansen, O., 1975: Thermal conductivity of soils. PH.D. Thesis,
!              University of Trondheim,
!      Peters-Lidard, C. D., et al., 1998: The effect of soil thermal
!              conductivity parameterization on surface energy fluxes
!              and temperatures. Journal of The Atmospheric Sciences,
!              Vol. 55, pp. 1209-1224.
! ----------------------------------------------------------------------------------------------------
! NEEDS PARAMETERS
! POROSITY(SOIL TYPE):
!      POROS = SMCMAX
! SATURATION RATIO:
! PARAMETERS  W/(M.K)
    SATRATIO = SMC / SMCMAX
    THKW = 0.57
!      IF (QUARTZ .LE. 0.2) THKO = 3.0
    THKO = 2.0
! SOLIDS' CONDUCTIVITY
! QUARTZ' CONDUCTIVITY
    THKQTZ = 7.7

! UNFROZEN FRACTION (FROM 1., i.e., 100%LIQUID, TO 0. (100% FROZEN))
    THKS = (THKQTZ ** QUARTZ)* (THKO ** (1. - QUARTZ))

! UNFROZEN VOLUME FOR SATURATION (POROSITY*XUNFROZ)
    XUNFROZ = SH2O / SMC
! SATURATED THERMAL CONDUCTIVITY
    XU = XUNFROZ * SMCMAX

! DRY DENSITY IN KG/M3
    THKSAT = THKS ** (1. - SMCMAX)* TKICE ** (SMCMAX - XU)* THKW **   &
         (XU)

! DRY THERMAL CONDUCTIVITY IN W.M-1.K-1
    GAMMD = (1. - SMCMAX)*2700.

    THKDRY = (0.135* GAMMD+ 64.7)/ (2700. - 0.947* GAMMD)
! FROZEN
    IF ( (SH2O + 0.0005) <  SMC ) THEN
       AKE = SATRATIO
! UNFROZEN
! RANGE OF VALIDITY FOR THE KERSTEN NUMBER (AKE)
    ELSE

! KERSTEN NUMBER (USING "FINE" FORMULA, VALID FOR SOILS CONTAINING AT
! LEAST 5% OF PARTICLES WITH DIAMETER LESS THAN 2.E-6 METERS.)
! (FOR "COARSE" FORMULA, SEE PETERS-LIDARD ET AL., 1998).

       IF ( SATRATIO >  0.1 ) THEN

          AKE = LOG10 (SATRATIO) + 1.0

! USE K = KDRY
       ELSE

          AKE = 0.0
       END IF
```

```
!  THERMAL CONDUCTIVITY

    END IF

    DF = AKE * (THKSAT - THKDRY) + THKDRY

  end subroutine TDFCND
! ==================================================================================================
  SUBROUTINE RADIATION (VEGTYP  ,IST     ,ISC     ,ICE     ,NSOIL   , & !in
                        SNEQVO  ,SNEQV   ,DT      ,COSZ    ,SNOWH   , & !in
                        TG      ,TV      ,FSNO    ,QSNOW   ,FWET    , & !in
                        ELAI    ,ESAI    ,SMC     ,SOLAD   ,SOLAI   , & !in
                        FVEG    ,ipoint  ,                            & !in
                        ALBOLD  ,                                     & !inout
                        FSUN    ,LAISUN  ,LAISHA  ,PARSUN  ,PARSHA  , & !out
                        SAV     ,SAG     ,FSR     ,FSA     )           !out
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input
  INTEGER, INTENT(IN)                    :: ipoint !
  INTEGER, INTENT(IN)                    :: VEGTYP !vegetation type
  INTEGER, INTENT(IN)                    :: IST    !surface type
  INTEGER, INTENT(IN)                    :: ISC    !soil color type (1-lighest; 8-darkest)
  INTEGER, INTENT(IN)                    :: ICE    !ice (ice = 1)
  INTEGER, INTENT(IN)                    :: NSOIL  !number of soil layers

  REAL, INTENT(IN)                       :: DT     !time step [s]
  REAL, INTENT(IN)                       :: QSNOW  !snowfall (mm/s)
  REAL, INTENT(IN)                       :: SNEQVO !snow mass at last time step(mm)
  REAL, INTENT(IN)                       :: SNEQV  !snow mass (mm)
  REAL, INTENT(IN)                       :: SNOWH  !snow height (mm)
  REAL, INTENT(IN)                       :: COSZ   !cosine solar zenith angle (0-1)
  REAL, INTENT(IN)                       :: TG     !ground temperature (k)
  REAL, INTENT(IN)                       :: TV     !vegetation temperature (k)
  REAL, INTENT(IN)                       :: ELAI   !LAI, one-sided, adjusted for burying by snow
  REAL, INTENT(IN)                       :: ESAI   !SAI, one-sided, adjusted for burying by snow
  REAL, INTENT(IN)                       :: FWET   !fraction of canopy that is wet
  REAL, DIMENSION(1:NSOIL), INTENT(IN)   :: SMC    !volumetric soil water [m3/m3]
  REAL, DIMENSION(1:2)    , INTENT(IN)   :: SOLAD  !incoming direct solar radiation (w/m2)
  REAL, DIMENSION(1:2)    , INTENT(IN)   :: SOLAI  !incoming diffuse solar radiation (w/m2)
  REAL, INTENT(IN)                       :: FSNO   !snow cover fraction (-)
  REAL, INTENT(IN)                       :: FVEG   !green vegetation fraction [0.0-1.0]

! inout
  REAL,                   INTENT(INOUT)  :: ALBOLD !snow albedo at last time step (CLASS type)

! output
  REAL, INTENT(OUT)                      :: FSUN   !sunlit fraction of canopy (-)
  REAL, INTENT(OUT)                      :: LAISUN !sunlit leaf area (-)
  REAL, INTENT(OUT)                      :: LAISHA !shaded leaf area (-)
  REAL, INTENT(OUT)                      :: PARSUN !average absorbed par for sunlit leaves (w/m2)
  REAL, INTENT(OUT)                      :: PARSHA !average absorbed par for shaded leaves (w/m2)
  REAL, INTENT(OUT)                      :: SAV    !solar radiation absorbed by vegetation (w/m2)
  REAL, INTENT(OUT)                      :: SAG    !solar radiation absorbed by ground (w/m2)
  REAL, INTENT(OUT)                      :: FSA    !total absorbed solar radiation (w/m2)
  REAL, INTENT(OUT)                      :: FSR    !total reflected solar radiation (w/m2)

! local
  REAL                                   :: FAGE   !snow age function (0 - new snow)
  REAL, DIMENSION(1:2)                   :: ALBGRD !ground albedo (direct)
  REAL, DIMENSION(1:2)                   :: ALBGRI !ground albedo (diffuse)
  REAL, DIMENSION(1:2)                   :: ALBD   !surface albedo (direct)
  REAL, DIMENSION(1:2)                   :: ALBI   !surface albedo (diffuse)
  REAL, DIMENSION(1:2)                   :: FABD   !flux abs by veg (per unit direct flux)
  REAL, DIMENSION(1:2)                   :: FABI   !flux abs by veg (per unit diffuse flux)
  REAL, DIMENSION(1:2)                   :: FTDD   !down direct flux below veg (per unit dir flux)
  REAL, DIMENSION(1:2)                   :: FTID   !down diffuse flux below veg (per unit dir flux)
  REAL, DIMENSION(1:2)                   :: FTII   !down diffuse flux below veg (per unit dif flux)
```

```fortran
  REAL                                :: FSHA   !shaded fraction of canopy
  REAL                                :: VAI    !total LAI + stem area index, one sided

  REAL,PARAMETER :: MPE = 1.E-6
  LOGICAL VEG  !true: vegetated for surface temperature calculation

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

! surface abeldo

   CALL ALBEDO (VEGTYP ,IST    ,ISC    ,ICE    ,NSOIL  , & !in
                DT     ,COSZ   ,FAGE   ,ELAI   ,ESAI   , & !in
                TG     ,TV     ,SNOWH  ,FSNO   ,FWET   , & !in
                SMC    ,SNEQVO ,SNEQV  ,QSNOW  ,FVEG   , & !in
                ipoint ,                                 & !in
                ALBOLD ,                                 & !inout
                ALBGRD ,ALBGRI ,ALBD   ,ALBI   ,FABD   , & !out
                FABI   ,FTDD   ,FTID   ,FTII   ,FSUN   )   !out

! surface radiation

     FSHA = 1.-FSUN
     LAISUN = ELAI*FSUN
     LAISHA = ELAI*FSHA
     VAI = ELAI+ ESAI
     IF (VAI .GT. 0.) THEN
        VEG = .TRUE.
     ELSE
        VEG = .FALSE.
     END IF

   CALL SURRAD (MPE    ,FSUN   ,FSHA   ,ELAI   ,VAI    , & !in
                LAISUN ,LAISHA ,SOLAD  ,SOLAI  ,FABD   , & !in
                FABI   ,FTDD   ,FTID   ,FTII   ,ALBGRD , & !in
                ALBGRI ,ALBD   ,ALBI   ,ipoint ,         & !in
                PARSUN ,PARSHA ,SAV    ,SAG    ,FSA    , & !out
                FSR    )                                   !out

  END SUBROUTINE RADIATION
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE ALBEDO (VEGTYP ,IST    ,ISC    ,ICE    ,NSOIL  , & !in
                     DT     ,COSZ   ,FAGE   ,ELAI   ,ESAI   , & !in
                     TG     ,TV     ,SNOWH  ,FSNO   ,FWET   , & !in
                     SMC    ,SNEQVO ,SNEQV  ,QSNOW  ,FVEG   , & !in
                     ipoint ,                                 & !in
                     ALBOLD ,                                 & !inout
                     ALBGRD ,ALBGRI ,ALBD   ,ALBI   ,FABD   , & !out
                     FABI   ,FTDD   ,FTID   ,FTII   ,FSUN   )   !out
! --------------------------------------------------------------------------------------------------
! surface albedos. also fluxes (per unit incoming direct and diffuse
! radiation) reflected, transmitted, and absorbed by vegetation.
! also sunlit fraction of the canopy.
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input
  INTEGER,                  INTENT(IN)  :: ipoint !
  INTEGER,                  INTENT(IN)  :: NSOIL  !number of soil layers
  INTEGER,                  INTENT(IN)  :: VEGTYP !vegetation type
  INTEGER,                  INTENT(IN)  :: IST    !surface type
  INTEGER,                  INTENT(IN)  :: ISC    !soil color type (1-lighest; 8-darkest)
  INTEGER,                  INTENT(IN)  :: ICE    !ice (ice = 1)

  REAL,                     INTENT(IN)  :: DT     !time step [sec]
  REAL,                     INTENT(IN)  :: QSNOW  !snowfall
```

```
  REAL,                        INTENT(IN)   :: COSZ   !cosine solar zenith angle for next time step
  REAL,                        INTENT(IN)   :: SNOWH  !snow height (mm)
  REAL,                        INTENT(IN)   :: TG     !ground temperature (k)
  REAL,                        INTENT(IN)   :: TV     !vegetation temperature (k)
  REAL,                        INTENT(IN)   :: ELAI   !LAI, one-sided, adjusted for burying by snow
  REAL,                        INTENT(IN)   :: ESAI   !SAI, one-sided, adjusted for burying by snow
  REAL,                        INTENT(IN)   :: FSNO   !fraction of grid covered by snow
  REAL,                        INTENT(IN)   :: FWET   !fraction of canopy that is wet
  REAL,                        INTENT(IN)   :: SNEQVO !snow mass at last time step(mm)
  REAL,                        INTENT(IN)   :: SNEQV  !snow mass (mm)
  REAL,                        INTENT(IN)   :: FVEG   !green vegetation fraction [0.0-1.0]
  REAL, DIMENSION(1:NSOIL),    INTENT(IN)   :: SMC    !volumetric soil water (m3/m3)

! inout
  REAL,                     INTENT(INOUT)   :: ALBOLD !snow albedo at last time step (CLASS type)

! output
  REAL, DIMENSION(1:     2), INTENT(OUT) :: ALBGRD !ground albedo (direct)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: ALBGRI !ground albedo (diffuse)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: ALBD   !surface albedo (direct)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: ALBI   !surface albedo (diffuse)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: FABD   !flux abs by veg (per unit direct flux)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: FABI   !flux abs by veg (per unit diffuse flux)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: FTDD   !down direct flux below veg (per unit dir flux)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: FTID   !down diffuse flux below veg (per unit dir flux)
  REAL, DIMENSION(1:     2), INTENT(OUT) :: FTII   !down diffuse flux below veg (per unit dif flux)
  REAL,                      INTENT(OUT) :: FSUN   !sunlit fraction of canopy (-)

! ------------------------------------------------------------------------
! ------------------------ local variables ----------------------------------
! local
  REAL                  :: FAGE     !snow age function
  REAL                  :: ALB
  INTEGER               :: IB       !indices
  INTEGER               :: NBAND    !number of solar radiation wave bands
  INTEGER               :: IC       !direct beam: ic=0; diffuse: ic=1

  REAL                  :: WL       !fraction of LAI+SAI that is LAI
  REAL                  :: WS       !fraction of LAI+SAI that is SAI
  REAL                  :: MPE      !prevents overflow for division by zero

  REAL, DIMENSION(1:2) :: RHO      !leaf/stem reflectance weighted by fraction LAI and SAI
  REAL, DIMENSION(1:2) :: TAU      !leaf/stem transmittance weighted by fraction LAI and SAI
  REAL, DIMENSION(1:2) :: FTDI     !down direct flux below veg per unit dif flux = 0
  REAL, DIMENSION(1:2) :: ALBSND   !snow albedo (direct)
  REAL, DIMENSION(1:2) :: ALBSNI   !snow albedo (diffuse)

  REAL                  :: VAI      !ELAI+ESAI
  REAL                  :: GDIR     !average projected leaf/stem area in solar direction
  REAL                  :: EXT      !optical depth direct beam per unit leaf + stem area

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

  NBAND = 2
  MPE = 1.E-06

! initialize output because solar radiation only done if COSZ > 0

  DO IB = 1, NBAND
    ALBD(IB) = 0.
    ALBI(IB) = 0.
    ALBGRD(IB) = 0.
    ALBGRI(IB) = 0.
    FABD(IB) = 0.
    FABI(IB) = 0.
    FTDD(IB) = 0.
    FTID(IB) = 0.
    FTII(IB) = 0.
    IF (IB.EQ.1) FSUN = 0.
```

```
  END DO

  IF(COSZ <= 0) GOTO 100

! weight reflectance/transmittance by LAI and SAI

  DO IB = 1, NBAND
    VAI = ELAI + ESAI
    WL  = ELAI / MAX(VAI,MPE)
    WS  = ESAI / MAX(VAI,MPE)
    RHO(IB) = MAX(RHOL(VEGTYP,IB)*WL+RHOS(VEGTYP,IB)*WS, MPE)
    TAU(IB) = MAX(TAUL(VEGTYP,IB)*WL+TAUS(VEGTYP,IB)*WS, MPE)
  END DO

! snow age

   CALL SNOW_AGE (DT,TG,SNEQVO,SNEQV,FAGE)

! snow albedos: only if COSZ > 0 and FSNO > 0

  IF(OPT_ALB == 1) &
     CALL SNOWALB_BATS (NBAND, FSNO,COSZ,FAGE,ALBSND,ALBSNI)
  IF(OPT_ALB == 2) THEN
     CALL SNOWALB_CLASS (NBAND,QSNOW,DT,ALB,ALBOLD,ALBSND,ALBSNI,ipoint)
     ALBOLD = ALB
  END IF

! ground surface albedo

  CALL GROUNDALB (NSOIL   ,NBAND   ,ICE     ,IST     ,ISC     , & !in
                  FSNO    ,SMC     ,ALBSND  ,ALBSNI  ,COSZ    , & !in
                  TG      ,ipoint  ,                            & !in
                  ALBGRD  ,ALBGRI  )                              !out

! loop over NBAND wavebands to calculate surface albedos and solar
! fluxes for unit incoming direct (IC=0) and diffuse flux (IC=1)

  DO IB = 1, NBAND
      IC = 0      ! direct
      CALL TWOSTREAM (IB     ,IC      ,VEGTYP  ,COSZ    ,VAI    , & !in
                      FWET   ,TV      ,ALBGRD  ,ALBGRI  ,RHO    , & !in
                      TAU    ,FVEG    ,IST     ,ipoint  ,         & !in
                      FABD   ,ALBD    ,FTDD    ,FTID    ,GDIR   )   !out
      IC = 1      ! diffuse
      CALL TWOSTREAM (IB     ,IC      ,VEGTYP  ,COSZ    ,VAI    , & !in
                      FWET   ,TV      ,ALBGRD  ,ALBGRI  ,RHO    , & !in
                      TAU    ,FVEG    ,IST     ,ipoint  ,         & !in
                      FABI   ,ALBI    ,FTDI    ,FTII    ,GDIR   )   !out
  END DO

! sunlit fraction of canopy. set FSUN = 0 if FSUN < 0.01.

  EXT = GDIR/COSZ * SQRT(1.-RHO(1)-TAU(1))
  FSUN = (1.-EXP(-EXT*VAI)) / MAX(EXT*VAI,MPE)
  EXT = FSUN

  IF (EXT .LT. 0.01) THEN
     WL = 0.
  ELSE
     WL = EXT
  END IF
  FSUN = WL

100 CONTINUE

  END SUBROUTINE ALBEDO
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE SURRAD (MPE     ,FSUN    ,FSHA    ,ELAI    ,VAI     , & !in
```

```
                         LAISUN  ,LAISHA  ,SOLAD   ,SOLAI   ,FABD    ,  & !in
                         FABI    ,FTDD    ,FTID    ,FTII    ,ALBGRD  ,  & !in
                         ALBGRI  ,ALBD    ,ALBI    ,ipoint  ,           & !in
                         PARSUN  ,PARSHA  ,SAV     ,SAG     ,FSA     ,  & !out
                         FSR     )                                        !out
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input

  INTEGER, INTENT(IN)                :: ipoint !
  REAL, INTENT(IN)                   :: MPE     !prevents underflow errors if division by zero

  REAL, INTENT(IN)                   :: FSUN    !sunlit fraction of canopy
  REAL, INTENT(IN)                   :: FSHA    !shaded fraction of canopy
  REAL, INTENT(IN)                   :: ELAI    !leaf area, one-sided
  REAL, INTENT(IN)                   :: VAI     !leaf + stem area, one-sided
  REAL, INTENT(IN)                   :: LAISUN  !sunlit leaf area index, one-sided
  REAL, INTENT(IN)                   :: LAISHA  !shaded leaf area index, one-sided

  REAL, DIMENSION(1:2), INTENT(IN) :: SOLAD   !incoming direct solar radiation (w/m2)
  REAL, DIMENSION(1:2), INTENT(IN) :: SOLAI   !incoming diffuse solar radiation (w/m2)
  REAL, DIMENSION(1:2), INTENT(IN) :: FABD    !flux abs by veg (per unit incoming direct flux)
  REAL, DIMENSION(1:2), INTENT(IN) :: FABI    !flux abs by veg (per unit incoming diffuse flux)
  REAL, DIMENSION(1:2), INTENT(IN) :: FTDD    !down dir flux below veg (per incoming dir flux)
  REAL, DIMENSION(1:2), INTENT(IN) :: FTID    !down dif flux below veg (per incoming dir flux)
  REAL, DIMENSION(1:2), INTENT(IN) :: FTII    !down dif flux below veg (per incoming dif flux)
  REAL, DIMENSION(1:2), INTENT(IN) :: ALBGRD  !ground albedo (direct)
  REAL, DIMENSION(1:2), INTENT(IN) :: ALBGRI  !ground albedo (diffuse)
  REAL, DIMENSION(1:2), INTENT(IN) :: ALBD    !overall surface albedo (direct)
  REAL, DIMENSION(1:2), INTENT(IN) :: ALBI    !overall surface albedo (diffuse)

! output

  REAL, INTENT(OUT)                  :: PARSUN  !average absorbed par for sunlit leaves (w/m2)
  REAL, INTENT(OUT)                  :: PARSHA  !average absorbed par for shaded leaves (w/m2)
  REAL, INTENT(OUT)                  :: SAV     !solar radiation absorbed by vegetation (w/m2)
  REAL, INTENT(OUT)                  :: SAG     !solar radiation absorbed by ground (w/m2)
  REAL, INTENT(OUT)                  :: FSA     !total absorbed solar radiation (w/m2)
  REAL, INTENT(OUT)                  :: FSR     !total reflected solar radiation (w/m2)

! ------------------------ local variables ----------------------------------------------------
  INTEGER                            :: IB      !waveband number (1=vis, 2=nir)
  INTEGER                            :: NBAND   !number of solar radiation waveband classes

  REAL                               :: ABS     !absorbed solar radiation (w/m2)
  REAL                               :: RNIR    !reflected solar radiation [nir] (w/m2)
  REAL                               :: RVIS    !reflected solar radiation [vis] (w/m2)
  REAL                               :: LAIFRA  !leaf area fraction of canopy
  REAL                               :: TRD     !transmitted solar radiation: direct (w/m2)
  REAL                               :: TRI     !transmitted solar radiation: diffuse (w/m2)
  REAL, DIMENSION(1:2)               :: CAD     !direct beam absorbed by canopy (w/m2)
  REAL, DIMENSION(1:2)               :: CAI     !diffuse radiation absorbed by canopy (w/m2)
! ---------------------------------------------------------------------------------------------
   NBAND = 2

! zero summed solar fluxes

    SAG = 0.
    SAV = 0.
    FSA = 0.

! loop over nband wavebands

  DO IB = 1, NBAND

! absorbed by canopy

    CAD(IB)  = SOLAD(IB)*FABD(IB)
```

```
    CAI(IB) = SOLAI(IB)*FABI(IB)
    SAV     = SAV + CAD(IB) + CAI(IB)
    FSA     = FSA + CAD(IB) + CAI(IB)

! transmitted solar fluxes incident on ground

    TRD = SOLAD(IB)*FTDD(IB)
    TRI = SOLAD(IB)*FTID(IB) + SOLAI(IB)*FTII(IB)

! solar radiation absorbed by ground surface

    ABS = TRD*(1.-ALBGRD(IB)) + TRI*(1.-ALBGRI(IB))
    SAG = SAG + ABS
    FSA = FSA + ABS
  END DO

! partition visible canopy absorption to sunlit and shaded fractions
! to get average absorbed par for sunlit and shaded leaves

     LAIFRA = ELAI / MAX(VAI,MPE)
     IF (FSUN .GT. 0.) THEN
        PARSUN = (CAD(1)+FSUN*CAI(1)) * LAIFRA / MAX(LAISUN,MPE)
        PARSHA = (FSHA*CAI(1))*LAIFRA / MAX(LAISHA,MPE)
     ELSE
        PARSUN = 0.
        PARSHA = (CAD(1)+CAI(1))*LAIFRA /MAX(LAISHA,MPE)
     ENDIF

! reflected solar radiation

     RVIS = ALBD(1)*SOLAD(1) + ALBI(1)*SOLAI(1)
     RNIR = ALBD(2)*SOLAD(2) + ALBI(2)*SOLAI(2)
     FSR  = RVIS + RNIR

  END SUBROUTINE SURRAD
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE SNOW_AGE (DT,TG,SNEQVO,SNEQV,FAGE)
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! ------------------------ code history ------------------------------------------------------------
! from BATS
! ------------------------ input/output variables --------------------------------------------------
!input
   REAL, INTENT(IN) :: DT        !main time step (s)
   REAL, INTENT(IN) :: TG        !ground temperature (k)
   REAL, INTENT(IN) :: SNEQVO    !snow mass at last time step(mm)
   REAL, INTENT(IN) :: SNEQV     !snow water per unit ground area (mm)

!output
   REAL, INTENT(OUT) :: FAGE     !snow age

!local
   REAL            :: TAUSS      !non-dimensional snow age
   REAL            :: TAGE       !total aging effects
   REAL            :: AGE1       !effects of grain growth due to vapor diffusion
   REAL            :: AGE2       !effects of grain growth at freezing of melt water
   REAL            :: AGE3       !effects of soot
   REAL            :: DELA       !temporary variable
   REAL            :: SGE        !temporary variable
   REAL            :: DELS       !temporary variable
   REAL            :: DELA0      !temporary variable
   REAL            :: ARG        !temporary variable
! See Yang et al. (1997) J.of Climate for detail.
!---------------------------------------------------------------------------------------------------

   IF(SNEQV.LE.0.0) THEN
          TAUSS = 0.
   ELSE IF (SNEQV.GT.800.) THEN
```

```
          TAUSS = 0.
   ELSE
          DELA0 = 1.E-6*DT
          ARG   = 5.E3*(1./TFRZ-1./TG)
          AGE1  = EXP(ARG)
          AGE2  = EXP(AMIN1(0.,10.*ARG))
          AGE3  = 0.3
          TAGE  = AGE1+AGE2+AGE3
          DELA  = DELA0*TAGE
          DELS  = AMAX1(0.0,SNEQV-SNEQVO) / SWEMX
          SGE   = (TAUSS+DELA)*(1.0-DELS)
          TAUSS = AMAX1(0.,SGE)
   ENDIF

   FAGE= TAUSS/(TAUSS+1.)

  END SUBROUTINE SNOW_AGE
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE SNOWALB_BATS (NBAND,FSNO,COSZ,FAGE,ALBSND,ALBSNI)
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input

  INTEGER,INTENT(IN) :: NBAND  !number of waveband classes

  REAL,INTENT(IN) :: COSZ    !cosine solar zenith angle
  REAL,INTENT(IN) :: FSNO    !snow cover fraction (-)
  REAL,INTENT(IN) :: FAGE    !snow age correction

! output

  REAL, DIMENSION(1:2),INTENT(OUT) :: ALBSND !snow albedo for direct(1=vis, 2=nir)
  REAL, DIMENSION(1:2),INTENT(OUT) :: ALBSNI !snow albedo for diffuse
! ---------------------------------------------------------------------------------------------

! ------------------------ local variables ----------------------------------------------------
  INTEGER :: IB          !waveband class

  REAL :: FZEN                  !zenith angle correction
  REAL :: CF1                   !temperary variable
  REAL :: SL2                   !2.*SL
  REAL :: SL1                   !1/SL
  REAL :: SL                    !adjustable parameter
  REAL, PARAMETER :: C1 = 0.2   !default in BATS
  REAL, PARAMETER :: C2 = 0.5   !default in BATS
!  REAL, PARAMETER :: C1 = 0.2 * 2. ! double the default to match Sleepers River's
!  REAL, PARAMETER :: C2 = 0.5 * 2. ! snow surface albedo (double aging effects)
! ---------------------------------------------------------------------------------------------
! zero albedos for all points

        ALBSND(1: NBAND) = 0.
        ALBSNI(1: NBAND) = 0.

! when cosz > 0

        SL=2.0
        SL1=1./SL
        SL2=2.*SL
        CF1=((1.+SL1)/(1.+SL2*COSZ)-SL1)
        FZEN=AMAX1(CF1,0.)

        ALBSNI(1)=0.95*(1.-C1*FAGE)
        ALBSNI(2)=0.65*(1.-C2*FAGE)

        ALBSND(1)=ALBSNI(1)+0.4*FZEN*(1.-ALBSNI(1))    !  vis direct
        ALBSND(2)=ALBSNI(2)+0.4*FZEN*(1.-ALBSNI(2))    !  nir direct
```

```
  END SUBROUTINE SNOWALB_BATS
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE SNOWALB_CLASS (NBAND,QSNOW,DT,ALB,ALBOLD,ALBSND,ALBSNI,ipoint)
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input

  INTEGER,INTENT(IN) :: ipoint !grid index
  INTEGER,INTENT(IN) :: NBAND  !number of waveband classes

  REAL,INTENT(IN) :: QSNOW     !snowfall (mm/s)
  REAL,INTENT(IN) :: DT        !time step (sec)
  REAL,INTENT(IN) :: ALBOLD    !snow albedo at last time step

! in & out

  REAL,                INTENT(INOUT) :: ALB        !
! output

  REAL, DIMENSION(1:2),INTENT(OUT) :: ALBSND !snow albedo for direct(1=vis, 2=nir)
  REAL, DIMENSION(1:2),INTENT(OUT) :: ALBSNI !snow albedo for diffuse
! ---------------------------------------------------------------------------------------------

! ------------------------ local variables ------------------------------------------------
  INTEGER :: IB          !waveband class

! ---------------------------------------------------------------------------------------------
! zero albedos for all points

        ALBSND(1: NBAND) = 0.
        ALBSNI(1: NBAND) = 0.

! when cosz > 0

         ALB = 0.55 + (ALBOLD-0.55) * EXP(-0.01*DT/3600.)

! 1 mm fresh snow(SWE) -- 10mm snow depth, assumed the fresh snow density 100kg/m3
! here assume 1cm snow depth will fully cover the old snow

         IF (QSNOW > 0.) then
           ALB = ALB + MIN(QSNOW*DT,SWEMX) * (0.84-ALB)/(SWEMX)
         ENDIF

         ALBSNI(1)= ALB         ! vis diffuse
         ALBSNI(2)= ALB         ! nir diffuse
         ALBSND(1)= ALB         ! vis direct
         ALBSND(2)= ALB         ! nir direct

  END SUBROUTINE SNOWALB_CLASS
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE GROUNDALB (NSOIL   ,NBAND   ,ICE     ,IST     ,ISC     , & !in
                        FSNO    ,SMC     ,ALBSND  ,ALBSNI  ,COSZ    , & !in
                        TG      ,ipoint  ,                            & !in
                        ALBGRD  ,ALBGRI  )                              !out
! --------------------------------------------------------------------------------------------------
  USE RAD_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
!input

  INTEGER,                  INTENT(IN)  :: ipoint !
  INTEGER,                  INTENT(IN)  :: NSOIL  !number of soil layers
  INTEGER,                  INTENT(IN)  :: NBAND  !number of solar radiation waveband classes
  INTEGER,                  INTENT(IN)  :: ICE    !value of ist for land ice
  INTEGER,                  INTENT(IN)  :: IST    !surface type
```

```
  INTEGER,                  INTENT(IN)  :: ISC    !soil color class (1-lighest; 8-darkest)
  REAL,                     INTENT(IN)  :: FSNO   !fraction of surface covered with snow (-)
  REAL,                     INTENT(IN)  :: TG     !ground temperature (k)
  REAL,                     INTENT(IN)  :: COSZ   !cosine solar zenith angle (0-1)
  REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: SMC    !volumetric soil water content (m3/m3)
  REAL, DIMENSION(1:    2), INTENT(IN)  :: ALBSND !direct beam snow albedo (vis, nir)
  REAL, DIMENSION(1:    2), INTENT(IN)  :: ALBSNI !diffuse snow albedo (vis, nir)

!output

  REAL, DIMENSION(1:    2), INTENT(OUT) :: ALBGRD !ground albedo (direct beam: vis, nir)
  REAL, DIMENSION(1:    2), INTENT(OUT) :: ALBGRI !ground albedo (diffuse: vis, nir)

!local

  INTEGER                               :: IB     !waveband number (1=vis, 2=nir)
  REAL                                  :: INC    !soil water correction factor for soil albedo
  REAL                                  :: ALBSOD !soil albedo (direct)
  REAL                                  :: ALBSOI !soil albedo (diffuse)
! --------------------------------------------------------------------------------------------------

  DO IB = 1, NBAND
        INC = MAX(0.11-0.40*SMC(1), 0.)
        IF (IST .EQ. 1)  THEN                     !soil
           ALBSOD = MIN(ALBSAT(ISC,IB)+INC,ALBDRY(ISC,IB))
           ALBSOI = ALBSOD
        ELSE IF (TG .GT. TFRZ) THEN               !unfrozen lake, wetland
           ALBSOD = 0.06/(MAX(0.01,COSZ)**1.7 + 0.15)
           ALBSOI = 0.06
        ELSE                                      !frozen lake, wetland
           ALBSOD = ALBLAK(IB)
           ALBSOI = ALBSOD
        END IF

! increase desert and semi-desert albedos

        IF (IST .EQ. 1 .AND. ISC .EQ. 9) THEN
           ALBSOD = ALBSOD + 0.10
           ALBSOI = ALBSOI + 0.10
        end if

        ALBGRD(IB) = ALBSOD*(1.-FSNO) + ALBSND(IB)*FSNO
        ALBGRI(IB) = ALBSOI*(1.-FSNO) + ALBSNI(IB)*FSNO
  END DO

  END SUBROUTINE GROUNDALB
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE TWOSTREAM (IB     ,IC      ,VEGTYP  ,COSZ    ,VAI    ,  & !in
                        FWET   ,T       ,ALBGRD  ,ALBGRI  ,RHO    ,  & !in
                        TAU    ,FVEG    ,IST     ,ipoint  ,          & !in
                        FAB    ,FRE     ,FTD     ,FTI     ,GDIR   )    !out
! --------------------------------------------------------------------------------------------------
! use two-stream approximation of Dickinson (1983) Adv Geophysics
! 25:305-353 and Sellers (1985) Int J Remote Sensing 6:1335-1372
! to calculate fluxes absorbed by vegetation, reflected by vegetation,
! and transmitted through vegetation for unit incoming direct or diffuse
! flux given an underlying surface with known albedo.
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
  USE RAD_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input

   INTEGER,              INTENT(IN)  :: ipoint !
   INTEGER,              INTENT(IN)  :: IST    !surface type
   INTEGER,              INTENT(IN)  :: IB      !waveband number
```

```
    INTEGER,              INTENT(IN)   :: IC      !0=unit incoming direct; 1=unit incoming diffuse
    INTEGER,              INTENT(IN)   :: VEGTYP  !vegetation type

    REAL,                 INTENT(IN)   :: COSZ    !cosine of direct zenith angle (0-1)
    REAL,                 INTENT(IN)   :: VAI     !one-sided leaf+stem area index (m2/m2)
    REAL,                 INTENT(IN)   :: FWET    !fraction of lai, sai that is wetted (-)
    REAL,                 INTENT(IN)   :: T       !surface temperature (k)

    REAL, DIMENSION(1:2), INTENT(IN)   :: ALBGRD  !direct  albedo of underlying surface (-)
    REAL, DIMENSION(1:2), INTENT(IN)   :: ALBGRI  !diffuse albedo of underlying surface (-)
    REAL, DIMENSION(1:2), INTENT(IN)   :: RHO     !leaf+stem reflectance
    REAL, DIMENSION(1:2), INTENT(IN)   :: TAU     !leaf+stem transmittance
    REAL,                 INTENT(IN)   :: FVEG    !green vegetation fraction [0.0-1.0]

! output

    REAL, DIMENSION(1:2), INTENT(OUT)  :: FAB     !flux abs by veg layer (per unit incoming flux)
    REAL, DIMENSION(1:2), INTENT(OUT)  :: FRE     !flux refl above veg layer (per unit incoming flux)
    REAL, DIMENSION(1:2), INTENT(OUT)  :: FTD     !down dir flux below veg layer (per unit in flux)
    REAL, DIMENSION(1:2), INTENT(OUT)  :: FTI     !down dif flux below veg layer (per unit in flux)
    REAL,                 INTENT(OUT)  :: GDIR    !projected leaf+stem area in solar direction

! local
    REAL                               :: OMEGA   !fraction of intercepted radiation that is scattered
    REAL                               :: OMEGAL  !omega for leaves
    REAL                               :: BETAI   !upscatter parameter for diffuse radiation
    REAL                               :: BETAIL  !betai for leaves
    REAL                               :: BETAD   !upscatter parameter for direct beam radiation
    REAL                               :: BETADL  !betad for leaves
    REAL                               :: EXT     !optical depth of direct beam per unit leaf area
    REAL                               :: AVMU    !average diffuse optical depth

    REAL                               :: COSZI   !0.001 <= cosz <= 1.000
    REAL                               :: ASU     !single scattering albedo
    REAL                               :: CHIL    ! -0.4 <= xl <= 0.6

    REAL                               :: TMP0,TMP1,TMP2,TMP3,TMP4,TMP5,TMP6,TMP7,TMP8,TMP9
    REAL                               :: P1,P2,P3,P4,S1,S2,U1,U2,U3
    REAL                               :: B,C,D,D1,D2,F,H,H1,H2,H3,H4,H5,H6,H7,H8,H9,H10
    REAL                               :: PHI1,PHI2,SIGMA
    REAL                               :: FTDS,FTIS,FRES

!  variables for the modified two-stream scheme
!  Niu and Yang (2004), JGR

    REAL, PARAMETER :: PAI = 3.14159265
    REAL :: HD       !crown depth (m)
    REAL :: BB       !vertical crown radius (m)
    REAL :: THETAP   !angle conversion from SZA
    REAL :: FA       !foliage volume density (m-1)
    REAL :: NEWVAI   !effective LSAI (-)
    REAL :: BGAP     !between canopy gap fraction for beam (-)
    REAL :: WGAP     !within canopy gap fraction for beam (-)
    REAL :: KOPEN    !gap fraction for diffue light (-)
    REAL :: GAP      !total gap fraction for beam ( <=1-shafac )

! ----------------------------------------------------------------------
! compute within and between gaps

     if(VAI == 0.0) THEN
         GAP     = 1.0
         KOPEN   = 1.0
     ELSE
         IF(OPT_RAD == 1) THEN
           HD      = HVT(VEGTYP) - HVB(VEGTYP)
           BB      = 0.5 * HD
           THETAP  = ATAN(BB/RC(VEGTYP) * TAN(ACOS(MAX(0.01,COSZ))) )
           BGAP    = EXP(-DEN(VEGTYP) * PAI * RC(VEGTYP)**2/COS(THETAP) )
           FA      = VAI/(1.33 * PAI * RC(VEGTYP)**3.0 *(BB/RC(VEGTYP))*DEN(VEGTYP))
```

```
            NEWVAI   = HD*FA
            WGAP     = (1.0-BGAP) * EXP(-0.5*NEWVAI/COSZ)
            GAP      = MIN(1.0-FVEG, BGAP+WGAP)
            KOPEN    = 0.05
          END IF

          IF(OPT_RAD == 2) THEN
            GAP      = 0.0
            KOPEN    = 0.0
          END IF

          IF(OPT_RAD == 3) THEN
            GAP      = 1.0-FVEG
            KOPEN    = 0.0
          END IF
      end if

! calculate two-stream parameters OMEGA, BETAD, BETAI, AVMU, GDIR, EXT.
! OMEGA, BETAD, BETAI are adjusted for snow. values for OMEGA*BETAD
! and OMEGA*BETAI are calculated and then divided by the new OMEGA
! because the product OMEGA*BETAI, OMEGA*BETAD is used in solution.
! also, the transmittances and reflectances (TAU, RHO) are linear
! weights of leaf and stem values.

      COSZI  = MAX(0.001, COSZ)
      CHIL   = MIN( MAX(XL(VEGTYP), -0.4), 0.6)
      IF (ABS(CHIL) .LE. 0.01) CHIL = 0.01
      PHI1   = 0.5 - 0.633*CHIL - 0.330*CHIL*CHIL
      PHI2   = 0.877 * (1.-2.*PHI1)
      GDIR   = PHI1 + PHI2*COSZI
      EXT    = GDIR/COSZI
      AVMU   = ( 1. - PHI1/PHI2 * LOG((PHI1+PHI2)/PHI1) ) / PHI2
      OMEGAL = RHO(IB) + TAU(IB)
      TMP0   = GDIR + PHI2*COSZI
      TMP1   = PHI1*COSZI
      ASU    = 0.5*OMEGAL*GDIR/TMP0 * ( 1.-TMP1/TMP0*LOG((TMP1+TMP0)/TMP1) )
      BETADL = (1.+AVMU*EXT)/(OMEGAL*AVMU*EXT)*ASU
      BETAIL = 0.5 * ( RHO(IB)+TAU(IB) + (RHO(IB)-TAU(IB))   &
             * ((1.+CHIL)/2.)**2 ) / OMEGAL

! adjust omega, betad, and betai for intercepted snow

      IF (T .GT. TFRZ) THEN                                !no snow
         TMP0 = OMEGAL
         TMP1 = BETADL
         TMP2 = BETAIL
      ELSE
         TMP0 =   (1.-FWET)*OMEGAL        + FWET*OMEGAS(IB)
         TMP1 = ( (1.-FWET)*OMEGAL*BETADL + FWET*OMEGAS(IB)*BETADS ) / TMP0
         TMP2 = ( (1.-FWET)*OMEGAL*BETAIL + FWET*OMEGAS(IB)*BETAIS ) / TMP0
      END IF

      OMEGA = TMP0
      BETAD = TMP1
      BETAI = TMP2

! absorbed, reflected, transmitted fluxes per unit incoming radiation

      B = 1. - OMEGA + OMEGA*BETAI
      C = OMEGA*BETAI
      TMP0 = AVMU*EXT
      D = TMP0 * OMEGA*BETAD
      F = TMP0 * OMEGA*(1.-BETAD)
      TMP1 = B*B - C*C
      H = SQRT(TMP1) / AVMU
      SIGMA = TMP0*TMP0 - TMP1
      if(SIGMA == 0.) SIGMA = 1.e-6
      P1 = B + AVMU*H
      P2 = B - AVMU*H
```

```
     P3 = B + TMP0
     P4 = B - TMP0
     S1 = EXP(-H*VAI)
     S2 = EXP(-EXT*VAI)
     IF (IC .EQ. 0) THEN
        U1 = B - C/ALBGRD(IB)
        U2 = B - C*ALBGRD(IB)
        U3 = F + C*ALBGRD(IB)
     ELSE
        U1 = B - C/ALBGRI(IB)
        U2 = B - C*ALBGRI(IB)
        U3 = F + C*ALBGRI(IB)
     END IF
     TMP2 = U1 - AVMU*H
     TMP3 = U1 + AVMU*H
     D1 = P1*TMP2/S1 - P2*TMP3*S1
     TMP4 = U2 + AVMU*H
     TMP5 = U2 - AVMU*H
     D2 = TMP4/S1 - TMP5*S1
     H1 = -D*P4 - C*F
     TMP6 = D - H1*P3/SIGMA
     TMP7 = ( D - C - H1/SIGMA*(U1+TMP0) ) * S2
     H2 = ( TMP6*TMP2/S1 - P2*TMP7 ) / D1
     H3 = - ( TMP6*TMP3*S1 - P1*TMP7 ) / D1
     H4 = -F*P3 - C*D
     TMP8 = H4/SIGMA
     TMP9 = ( U3 - TMP8*(U2-TMP0) ) * S2
     H5 = - ( TMP8*TMP4/S1 + TMP9 ) / D2
     H6 = ( TMP8*TMP5*S1 + TMP9 ) / D2
     H7 = (C*TMP2) / (D1*S1)
     H8 = (-C*TMP3*S1) / D1
     H9 = TMP4 / (D2*S1)
     H10 = (-TMP5*S1) / D2

! downward direct and diffuse fluxes below vegetation
! Niu and Yang (2004), JGR.

     IF (IC .EQ. 0) THEN
        FTDS = S2                           *(1.0-GAP) + GAP
        FTIS = (H4*S2/SIGMA + H5*S1 + H6/S1)*(1.0-GAP)
     ELSE
        FTDS = 0.
        FTIS = (H9*S1 + H10/S1)*(1.0-KOPEN) + KOPEN
     END IF
     FTD(IB) = FTDS
     FTI(IB) = FTIS

! flux reflected by the surface (veg. and ground)

     IF (IC .EQ. 0) THEN
        FRES = (H1/SIGMA + H2 + H3)*(1.0-GAP  ) + ALBGRD(IB)*GAP
     ELSE
        FRES = (H7 + H8)           *(1.0-KOPEN) + ALBGRI(IB)*KOPEN
     END IF
     FRE(IB) = FRES

! flux absorbed by vegetation

     FAB(IB) = 1. - FRE(IB) - (1.-ALBGRD(IB))*FTD(IB) &
                            - (1.-ALBGRI(IB))*FTI(IB)

  END SUBROUTINE TWOSTREAM
! ==================================================================================================
  SUBROUTINE VEGE_FLUX(NSNOW   ,NSOIL   ,ISNOW   ,VEGTYP  ,VEG     , & !in
                       DT      ,SAV     ,SAG     ,LWDN    ,UR      , & !in
                       UU      ,VV      ,SFCTMP  ,THAIR   ,QAIR    , & !in
                       EAIR    ,RHOAIR  ,SNOWH   ,VAI     ,GAMMA   , & !in
                       FWET    ,LAISUN  ,LAISHA  ,CWP     ,DZSNSO  , & !in
                       HTOP    ,ZLVL    ,ZPD     ,ZOM     ,FVEG    , & !in
```

```
                        ZOMG    ,EMV    ,EMG    ,CANLIQ  ,               & !in
                        CANICE  ,STC    ,DF     ,RSSUN   ,RSSHA   ,      & !in
                        RSURF   ,LATHEA ,PARSUN ,PARSHA  ,IGS     ,      & !in
                        FOLN    ,CO2AIR ,O2AIR  ,BTRAN   ,SFCPRS  ,      & !in
                        RHSUR   ,ipoint ,                                & !in
                        EAH     ,TAH    ,TV     ,TG      ,CM      ,      & !inout
                        CH      ,                                        & !inout
                        TAUXV   ,TAUYV  ,IRG    ,IRC     ,SHG     ,      & !out
                        SHC     ,EVG    ,EVC    ,TR      ,GH      ,      & !out
                        T2MV    ,PSNSUN ,PSNSHA )                          !out

! --------------------------------------------------------------------------------------------------
! use newton-raphson iteration to solve for vegetation (tv) and
! ground (tg) temperatures that balance the surface energy budgets

! vegetated:
! -SAV + IRC[TV] + SHC[TV] + EVC[TV] + TR[TV] = 0
! -SAG + IRG[TG] + SHG[TG] + EVG[TG] + GH[TG] = 0
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! input
  INTEGER,                         INTENT(IN) :: ipoint
  LOGICAL,                         INTENT(IN) :: VEG    !true if vegetated surface
  INTEGER,                         INTENT(IN) :: NSNOW  !maximum no. of snow layers
  INTEGER,                         INTENT(IN) :: NSOIL  !number of soil layers
  INTEGER,                         INTENT(IN) :: ISNOW  !actual no. of snow layers
  INTEGER,                         INTENT(IN) :: VEGTYP !vegetation physiology type
  REAL,                            INTENT(IN) :: FVEG   !greeness vegetation fraction (-)
  REAL,                            INTENT(IN) :: SAV    !solar rad absorbed by veg (w/m2)
  REAL,                            INTENT(IN) :: SAG    !solar rad absorbed by ground (w/m)
  REAL,                            INTENT(IN) :: LWDN   !atmospheric longwave radiation (w/m2)
  REAL,                            INTENT(IN) :: UR     !wind speed at height zlvl (m/s)
  REAL,                            INTENT(IN) :: UU     !wind speed in eastward dir (m/s)
  REAL,                            INTENT(IN) :: VV     !wind speed in northward dir (m/s)
  REAL,                            INTENT(IN) :: SFCTMP !air temperature at reference height (k)
  REAL,                            INTENT(IN) :: THAIR  !potential temp at reference height (k)
  REAL,                            INTENT(IN) :: EAIR   !vapor pressure air at zlvl (pa)
  REAL,                            INTENT(IN) :: QAIR   !specific humidity at zlvl (kg/kg)
  REAL,                            INTENT(IN) :: RHOAIR !density air (kg/m**3)
  REAL,                            INTENT(IN) :: DT     !time step (s)

  REAL,                            INTENT(IN) :: SNOWH  !actual snow depth [m]
  REAL,                            INTENT(IN) :: FWET   !wetted fraction of canopy
  REAL,                            INTENT(IN) :: HTOP   !top of canopy layer (m)
  REAL,                            INTENT(IN) :: CWP    !canopy wind parameter

  REAL,                            INTENT(IN) :: VAI    !total leaf area index + stem area index
  REAL,                            INTENT(IN) :: LAISUN !sunlit leaf area index, one-sided (m2/m2)
  REAL,                            INTENT(IN) :: LAISHA !shaded leaf area index, one-sided (m2/m2)
  REAL,                            INTENT(IN) :: ZLVL   !reference height (m)
  REAL,                            INTENT(IN) :: ZPD    !zero plane displacement (m)
  REAL,                            INTENT(IN) :: Z0M    !roughness length, momentum (m)
  REAL,                            INTENT(IN) :: Z0MG   !roughness length, momentum, ground (m)
  REAL,                            INTENT(IN) :: EMV    !vegetation emissivity
  REAL,                            INTENT(IN) :: EMG    !ground emissivity

  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: STC    !soil/snow temperature (k)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DF     !thermal conductivity of snow/soil (w/m/k)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !thinkness of snow/soil layers (m)
  REAL,                            INTENT(IN) :: CANLIQ !intercepted liquid water (mm)
  REAL,                            INTENT(IN) :: CANICE !intercepted ice mass (mm)
  REAL,                            INTENT(IN) :: RSURF  !ground surface resistance (s/m)
  REAL,                            INTENT(IN) :: GAMMA  !psychrometric constant (pa/K)
  REAL,                            INTENT(IN) :: LATHEA !latent heat of vaporization/subli (j/kg)
  REAL,                            INTENT(IN) :: PARSUN !par absorbed per unit sunlit lai (w/m2)
  REAL,                            INTENT(IN) :: PARSHA !par absorbed per unit shaded lai (w/m2)
```

```
  REAL,                        INTENT(IN) :: FOLN   !foliage nitrogen (%)
  REAL,                        INTENT(IN) :: CO2AIR !atmospheric co2 concentration (pa)
  REAL,                        INTENT(IN) :: O2AIR  !atmospheric o2 concentration (pa)
  REAL,                        INTENT(IN) :: IGS    !growing season index (0=off, 1=on)
  REAL,                        INTENT(IN) :: SFCPRS !pressure (pa)
  REAL,                        INTENT(IN) :: BTRAN  !soil water transpiration factor (0 to 1)
  REAL,                        INTENT(IN) :: RHSUR  !raltive humidity in surface soil/snow air space (-)

! input/output
  REAL,                     INTENT(INOUT) :: EAH    !canopy air vapor pressure (pa)
  REAL,                     INTENT(INOUT) :: TAH    !canopy air temperature (k)
  REAL,                     INTENT(INOUT) :: TV     !vegetation temperature (k)
  REAL,                     INTENT(INOUT) :: TG     !ground temperature (k)
  REAL,                     INTENT(INOUT) :: CM     !momentum drag coefficient
  REAL,                     INTENT(INOUT) :: CH     !sensible heat exchange coefficient

! output
! -FSA + FIRA + FSH + (FCEV + FCTR + FGEV) + FCST + SSOIL = 0
  REAL,                       INTENT(OUT) :: TAUXV  !wind stress: e-w (n/m2)
  REAL,                       INTENT(OUT) :: TAUYV  !wind stress: n-s (n/m2)
  REAL,                       INTENT(OUT) :: IRC    !net longwave radiation (w/m2) [+= to atm]
  REAL,                       INTENT(OUT) :: SHC    !sensible heat flux (w/m2)     [+= to atm]
  REAL,                       INTENT(OUT) :: EVC    !evaporation heat flux (w/m2)  [+= to atm]
  REAL,                       INTENT(OUT) :: IRG    !net longwave radiation (w/m2) [+= to atm]
  REAL,                       INTENT(OUT) :: SHG    !sensible heat flux (w/m2)     [+= to atm]
  REAL,                       INTENT(OUT) :: EVG    !evaporation heat flux (w/m2)  [+= to atm]
  REAL,                       INTENT(OUT) :: TR     !transpiration heat flux (w/m2)[+= to atm]
  REAL,                       INTENT(OUT) :: GH     !ground heat (w/m2) [+ = to soil]
  REAL,                       INTENT(OUT) :: T2MV   !2 m height air temperature (k)
  REAL,                       INTENT(OUT) :: PSNSUN !sunlit leaf photosynthesis (umolco2/m2/s)
  REAL,                       INTENT(OUT) :: PSNSHA !shaded leaf photosynthesis (umolco2/m2/s)

! ------------------------ local variables ----------------------------------------------------
  REAL :: CW            !water vapor exchange coefficient
  REAL :: FV            !friction velocity (m/s)
  REAL :: WSTAR         !friction velocity n vertical direction (m/s) (only for SFCDIF2)
  REAL :: Z0H           !roughness length, sensible heat (m)
  REAL :: Z0HG          !roughness length, sensible heat (m)
  REAL :: RB            !bulk leaf boundary layer resistance (s/m)
  REAL :: RAMC          !aerodynamic resistance for momentum (s/m)
  REAL :: RAHC          !aerodynamic resistance for sensible heat (s/m)
  REAL :: RAWC          !aerodynamic resistance for water vapor (s/m)
  REAL :: RAMG          !aerodynamic resistance for momentum (s/m)
  REAL :: RAHG          !aerodynamic resistance for sensible heat (s/m)
  REAL :: RAWG          !aerodynamic resistance for water vapor (s/m)
  REAL :: RSSUN         !sunlit leaf stomatal resistance (s/m)
  REAL :: RSSHA         !shaded leaf stomatal resistance (s/m)
  REAL :: MOL           !Monin-Obukhov length (m)
  REAL :: DTV           !change in tv, last iteration (k)
  REAL :: DTG           !change in tg, last iteration (k)

  REAL :: AIR,CIR       !coefficients for ir as function of ts**4
  REAL :: CSH           !coefficients for sh as function of ts
  REAL :: CEV           !coefficients for ev as function of esat[ts]
  REAL :: CGH           !coefficients for st as function of ts
  REAL :: ATR,CTR       !coefficients for tr as function of esat[ts]
  REAL :: ATA,BTA       !coefficients for tah as function of ts
  REAL :: AEA,BEA       !coefficients for eah as function of esat[ts]

  REAL :: ESTV          !saturation vapor pressure at ts (pa)
  REAL :: ESTG          !saturation vapor pressure at tg (pa)
  REAL :: DESTV         !d(es)/dt at ts (pa/k)
  REAL :: DESTG         !d(es)/dt at tg (pa/k)
  REAL :: ESATW         !es for water
  REAL :: ESATI         !es for ice
  REAL :: DSATW         !d(es)/dt at tg (pa/k) for water
  REAL :: DSATI         !d(es)/dt at tg (pa/k) for ice

  REAL :: FM            !momentum stability correction, weighted by prior iters
```

```
  REAL :: FH           !sen heat stability correction, weighted by prior iters
  REAL :: FHG          !sen heat stability correction, ground
  REAL :: HCAN         !canopy height (m) [note: hcan >= z0mg]

  REAL :: A            !temporary calculation
  REAL :: B            !temporary calculation
  REAL :: CAH          !sensible heat conductance, canopy air to ZLVL air (m/s)
  REAL :: CVH          !sensible heat conductance, leaf surface to canopy air (m/s)
  REAL :: CAW          !latent heat conductance, canopy air ZLVL air (m/s)
  REAL :: CTW          !transpiration conductance, leaf to canopy air (m/s)
  REAL :: CEW          !evaporation conductance, leaf to canopy air (m/s)
  REAL :: CGW          !latent heat conductance, ground to canopy air (m/s)
  REAL :: COND         !sum of conductances (s/m)
  REAL :: UC           !wind speed at top of canopy (m/s)
  REAL :: KH           !turbulent transfer coefficient, sensible heat, (m2/s)
  REAL :: H            !temporary sensible heat flux (w/m2)
  REAL :: HG           !temporary sensible heat flux (w/m2)
  REAL :: MOZ          !Monin-Obukhov stability parameter
  REAL :: MOZG         !Monin-Obukhov stability parameter
  REAL :: MOZOLD       !Monin-Obukhov stability parameter from prior iteration

  REAL :: VAIE         !total leaf area index + stem area index,effective
  REAL :: LAISUNE      !sunlit leaf area index, one-sided (m2/m2),effective
  REAL :: LAISHAE      !shaded leaf area index, one-sided (m2/m2),effective

  INTEGER :: K         !index
  INTEGER :: ITER      !iteration index
  INTEGER :: NITERC    !number of iterations for surface temperature
  INTEGER :: NITERG    !number of iterations for ground temperature
  INTEGER :: MOZSGN    !number of times MOZ changes sign
  REAL    :: MPE       !prevents overflow error if division by zero

  DATA NITERC,NITERG /5,3/
  SAVE NITERC,NITERG
  REAL :: T, TDC       !Kelvin to degree Celsius with limit -50 to +50
  TDC(T)   = MIN( 50., MAX(-50.,(T-TFRZ)) )
! ----------------------------------------------------------------------------------------------
        MPE = 1E-6
! ----------------------------------------------------------------------------------------------
! initialization variables that do not depend on stability iteration
! ----------------------------------------------------------------------------------------------
        DTV = 0.
        DTG = 0.
        MOZSGN = 0
        MOZOLD = 0.
        HG     = 0.
        H      = 0.

! convert grid-cell LAI to the fractional vegetated area (FVEG)

        VAIE    = MIN(6.,VAI    / FVEG)
        LAISUNE = MIN(6.,LAISUN / FVEG)
        LAISHAE = MIN(6.,LAISHA / FVEG)

! saturation vapor pressure at ground temperature

        T = TDC(TG)
        CALL ESAT(T, ESATW, ESATI, DSATW, DSATI)
        IF (T .GT. 0.) THEN
           ESTG = ESATW
        ELSE
           ESTG = ESATI
        END IF

! canopy height

        HCAN = HTOP
        UC = UR*LOG(HCAN/Z0M)/LOG(ZLVL/Z0M)
        IF((HCAN-ZPD) <= 0.) THEN
```

```
          WRITE(*,*) "CRITICAL PROBLEM: HCAN <= ZPD"
          WRITE(*,*) 'ipoint=',ipoint
          WRITE(*,*) 'HCAN  =',HCAN
          WRITE(*,*) 'ZPD   =',ZPD
          WRITE(*,*) 'SNOWH =',SNOWH
          STOP
        END IF

! prepare for longwave rad.

        AIR = -EMV*(1.+(1.-EMV)*(1.-EMG))*LWDN - EMV*EMG*SB*TG**4
        CIR = (2.-EMV*(1.-EMG))*EMV*SB
! ---------------------------------------------------------------------------------------------
      DO ITER = 1, NITERC    !  begin stability iteration

       IF(ITER == 1) THEN
            Z0H  = Z0M
            Z0HG = Z0MG
       ELSE
            Z0H  = Z0M    !* EXP(-CZIL*0.4*258.2*SQRT(FV*Z0M))
            Z0HG = Z0MG   !* EXP(-CZIL*0.4*258.2*SQRT(FV*Z0MG))
       END IF

! aerodyn resistances between heights zlvl and d+z0v

       IF(OPT_SFC == 1) THEN
          CALL SFCDIF1(ITER    ,SFCTMP ,RHOAIR ,H      ,QAIR   , & !in
                       ZLVL    ,ZPD    ,Z0M    ,Z0H    ,UR     , & !in
                       MPE     ,ipoint ,                         & !in
                       MOZ     ,MOZSGN ,FM     ,FH     ,         & !inout
                       CM      ,CH     ,FV     )                   !out

       ENDIF

       IF(OPT_SFC == 2) THEN
          CALL SFCDIF2(ITER    ,Z0M    ,TAH    ,THAIR  ,UR     , & !in
                       CZIL    ,ZLVL   ,ipoint ,                 & !in
                       CM      ,CH     ,MOZ    ,WSTAR  ,         & !inout
                       FV      )                                   !out
       ENDIF

       RAMC = MAX(1.,1./(CM*UR))
       RAHC = MAX(1.,1./(CH*UR))
       RAWC = RAHC

! aerodyn resistance between heights z0g and d+z0v, RAG, and leaf
! boundary layer resistance, RB

       CALL RAGRB(ITER   ,VAIE   ,RHOAIR ,HG     ,TAH    , & !in
                  ZPD    ,Z0MG   ,Z0HG   ,HCAN   ,UC     , & !in
                  Z0H    ,FV     ,CWP    ,VEGTYP ,MPE    , & !in
                  TV     ,MOZG   ,FHG    ,ipoint ,         & !inout
                  RAMG   ,RAHG   ,RAWG   ,RB     )           !out

! es and d(es)/dt evaluated at tv

        T = TDC(TV)
        CALL ESAT(T, ESATW, ESATI, DSATW, DSATI)
        IF (T .GT. 0.) THEN
           ESTV  = ESATW
           DESTV = DSATW
        ELSE
           ESTV  = ESATI
           DESTV = DSATI
        END IF

! stomatal resistance

     IF(ITER == 1 ) THEN
```

```
        IF (OPT_CRS == 1) then  ! Ball-Berry
         CALL STOMATA (VEGTYP,MPE   ,PARSUN ,FOLN  ,ipoint, & !in
                       TV    ,ESTV  ,EAH    ,SFCTMP,SFCPRS, & !in
                       O2AIR ,CO2AIR,IGS    ,BTRAN ,RB    , & !in
                       RSSUN ,PSNSUN)                         !out

         CALL STOMATA (VEGTYP,MPE   ,PARSHA ,FOLN  ,ipoint, & !in
                       TV    ,ESTV  ,EAH    ,SFCTMP,SFCPRS, & !in
                       O2AIR ,CO2AIR,IGS    ,BTRAN ,RB    , & !in
                       RSSHA ,PSNSHA)                         !out
        END IF

        IF (OPT_CRS == 2) then  ! Jarvis
         CALL  CANRES (PARSUN,TV    ,BTRAN ,EAH    ,SFCPRS, & !in
                       RSSUN ,PSNSUN,ipoint)                  !out

         CALL  CANRES (PARSHA,TV    ,BTRAN ,EAH    ,SFCPRS, & !in
                       RSSHA ,PSNSHA,ipoint)                  !out
        END IF
     END IF

! prepare for sensible heat flux above veg.

        CAH  = 1./RAHC
        CVH  = 2.*VAIE/RB
        CGH  = 1./RAHG
        COND = CAH + CVH + CGH
        ATA  = (THAIR*CAH + TG*CGH) / COND
        BTA  = CVH/COND
        CSH  = (1.-BTA)*RHOAIR*CPAIR*CVH

! prepare for latent heat flux above veg.

        CAW  = 1./RAWC
        CEW  = FWET*VAIE/RB
        CTW  = (1.-FWET)*(LAISUNE/(RB+RSSUN) + LAISHAE/(RB+RSSHA))
        CGW  = 1./(RAWG+RSURF)
        COND = CAW + CEW + CTW + CGW
        AEA  = (EAIR*CAW + ESTG*CGW) / COND
        BEA  = (CEW+CTW)/COND
        CEV  = (1.-BEA)*CEW*RHOAIR*CPAIR/GAMMA
        CTR  = (1.-BEA)*CTW*RHOAIR*CPAIR/GAMMA

! evaluate surface fluxes with current temperature and solve for dts

        TAH = ATA + BTA*TV               ! canopy air T.
        EAH = AEA + BEA*ESTV             ! canopy air e

        IRC = FVEG*(AIR + CIR*TV**4)
        SHC = FVEG*RHOAIR*CPAIR*CVH * (  TV-TAH)
        EVC = FVEG*RHOAIR*CPAIR*CEW * (ESTV-EAH) / GAMMA
        TR  = FVEG*RHOAIR*CPAIR*CTW * (ESTV-EAH) / GAMMA
        EVC = MIN(CANLIQ*LATHEA/DT,EVC)

        B   = SAV-IRC-SHC-EVC-TR
        A   = FVEG*(4.*CIR*TV**3 + CSH + (CEV+CTR)*DESTV)
        DTV = B/A

        IRC = IRC + FVEG*4.*CIR*TV**3*DTV
        SHC = SHC + FVEG*CSH*DTV
        EVC = EVC + FVEG*CEV*DESTV*DTV
        TR  = TR  + FVEG*CTR*DESTV*DTV

! update vegetation surface temperature

        TV  = TV + DTV

! for computing M-O length in the next iteration
```

```
        H  = RHOAIR*CPAIR*(TAH - THAIR) /RAHC
        HG = RHOAIR*CPAIR*(TG  - TAH)   /RAHG

     END DO    ! end stability iteration

! under-canopy fluxes and tg

        AIR = - EMG*(1.-EMV)*LWDN - EMG*EMV*SB*TV**4
        CIR = EMG*SB
        CSH = RHOAIR*CPAIR/RAHG
        CEV = RHOAIR*CPAIR / (GAMMA*(RAWG+RSURF))
        CGH = 2.*DF(ISNOW+1)/DZSNSO(ISNOW+1)

     DO ITER = 1, NITERG

        T = TDC(TG)
        CALL ESAT(T, ESATW, ESATI, DSATW, DSATI)
        IF (T .GT. 0.) THEN
            ESTG  = ESATW
            DESTG = DSATW
        ELSE
            ESTG  = ESATI
            DESTG = DSATI
        END IF

        IRG = CIR*TG**4 + AIR
        SHG = CSH * (TG          - TAH          )
        EVG = CEV * (ESTG*RHSUR - EAH          )
        GH  = CGH * (TG          - STC(ISNOW+1))

        B = SAG-IRG-SHG-EVG-GH
        A = 4.*CIR*TG**3+CSH+CEV*DESTG+CGH
        DTG = B/A

        IRG = IRG + 4.*CIR*TG**3*DTG
        SHG = SHG + CSH*DTG
        EVG = EVG + CEV*DESTG*DTG
        GH  = GH  + CGH*DTG
        TG  = TG  + DTG
      END DO

! if snow on ground and TG > TFRZ: reset TG = TFRZ. reevaluate ground fluxes.

     IF(OPT_STC == 1) THEN
     IF (SNOWH > 0.05 .AND. TG > TFRZ) THEN
        TG  = TFRZ
        IRG = CIR*TG**4 - EMG*(1.-EMV)*LWDN - EMG*EMV*SB*TV**4
        SHG = CSH * (TG          - TAH)
        EVG = CEV * (ESTG*RHSUR - EAH)
        GH  = SAG - (IRG+SHG+EVG)
     END IF
     END IF

! wind stresses

     TAUXV = -RHOAIR*CM*UR*UU
     TAUYV = -RHOAIR*CM*UR*VV

! 2 m height air temperature

     T2MV = TAH - (SHG+SHC)/(RHOAIR*CPAIR*FV) * 1./VKC * LOG((2.+Z0H)/Z0H)

  END SUBROUTINE VEGE_FLUX
! ==================================================================================================
  SUBROUTINE BARE_FLUX (NSNOW    ,NSOIL    ,ISNOW    ,DT       ,SAG      ,  & !in
                        LWDN     ,UR       ,UU       ,VV       ,SFCTMP   ,  & !in
                        THAIR    ,QAIR     ,EAIR     ,RHOAIR   ,SNOWH    ,  & !in
                        DZSNSO   ,ZLVL     ,ZPD      ,Z0M      ,             & !in
                        EMG      ,STC      ,DF       ,RSURF    ,LATHEA   ,  & !in
```

```
                     GAMMA   ,RHSUR   ,ipoint  ,                    & !in
                     TGB     ,CM      ,CH      ,                    & !inout
                     TAUXB   ,TAUYB   ,IRB     ,SHB     ,EVB     ,  & !out
                     GHB     ,T2MB    )                               !out
! --------------------------------------------------------------------------------------------------
! use newton-raphson iteration to solve ground (tg) temperature
! that balances the surface energy budgets for bare soil fraction.

! bare soil:
! -SAB + IRB[TG] + SHB[TG] + EVB[TG] + GHB[TG] = 0
! ----------------------------------------------------------------------
  USE VEG_PARAMETERS
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! input
  integer                          , INTENT(IN)    :: ipoint
  INTEGER,                           INTENT(IN) :: NSNOW  !maximum no. of snow layers
  INTEGER,                           INTENT(IN) :: NSOIL  !number of soil layers
  INTEGER,                           INTENT(IN) :: ISNOW  !actual no. of snow layers
  REAL,                              INTENT(IN) :: DT     !time step (s)
  REAL,                              INTENT(IN) :: SAG    !solar radiation absorbed by ground (w/m2)
  REAL,                              INTENT(IN) :: LWDN   !atmospheric longwave radiation (w/m2)
  REAL,                              INTENT(IN) :: UR     !wind speed at height zlvl (m/s)
  REAL,                              INTENT(IN) :: UU     !wind speed in eastward dir (m/s)
  REAL,                              INTENT(IN) :: VV     !wind speed in northward dir (m/s)
  REAL,                              INTENT(IN) :: SFCTMP !air temperature at reference height (k)
  REAL,                              INTENT(IN) :: THAIR  !potential temperature at height zlvl (k)
  REAL,                              INTENT(IN) :: QAIR   !specific humidity at height zlvl (kg/kg)
  REAL,                              INTENT(IN) :: EAIR   !vapor pressure air at height (pa)
  REAL,                              INTENT(IN) :: RHOAIR !density air (kg/m3)
  REAL,                              INTENT(IN) :: SNOWH  !actual snow depth [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL),   INTENT(IN) :: DZSNSO !thickness of snow/soil layers (m)
  REAL,                              INTENT(IN) :: ZLVL   !reference height (m)
  REAL,                              INTENT(IN) :: ZPD    !zero plane displacement (m)
  REAL,                              INTENT(IN) :: Z0M    !roughness length, momentum, ground (m)
  REAL,                              INTENT(IN) :: EMG    !ground emissivity
  REAL, DIMENSION(-NSNOW+1:NSOIL),   INTENT(IN) :: STC    !soil/snow temperature (k)
  REAL, DIMENSION(-NSNOW+1:NSOIL),   INTENT(IN) :: DF     !thermal conductivity of snow/soil (w/m/k)
  REAL,                              INTENT(IN) :: RSURF  !ground surface resistance (s/m)
  REAL,                              INTENT(IN) :: LATHEA !latent heat of vaporization/subli (j/kg)
  REAL,                              INTENT(IN) :: GAMMA  !psychrometric constant (pa/k)
  REAL,                              INTENT(IN) :: RHSUR  !raltive humidity in surface soil/snow air space (-)

! input/output
  REAL,                           INTENT(INOUT) :: TGB    !ground temperature (k)
  REAL,                           INTENT(INOUT) :: CM     !momentum drag coefficient
  REAL,                           INTENT(INOUT) :: CH     !sensible heat exchange coefficient

! output
! -SAB + IRB[TG] + SHB[TG] + EVB[TG] + GHB[TG] = 0

  REAL,                             INTENT(OUT) :: TAUXB  !wind stress: e-w (n/m2)
  REAL,                             INTENT(OUT) :: TAUYB  !wind stress: n-s (n/m2)
  REAL,                             INTENT(OUT) :: IRB    !net longwave rad (w/m2)   [+ to atm]
  REAL,                             INTENT(OUT) :: SHB    !sensible heat flux (w/m2) [+ to atm]
  REAL,                             INTENT(OUT) :: EVB    !latent heat flux (w/m2)   [+ to atm]
  REAL,                             INTENT(OUT) :: GHB    !ground heat flux (w/m2)  [+ to soil]
  REAL,                             INTENT(OUT) :: T2MB   !2 m height air temperature (k)

! local variables

  REAL :: TAUX       !wind stress: e-w (n/m2)
  REAL :: TAUY       !wind stress: n-s (n/m2)
  REAL :: FIRA       !total net longwave rad (w/m2)      [+ to atm]
  REAL :: FSH        !total sensible heat flux (w/m2)    [+ to atm]
  REAL :: FGEV       !ground evaporation heat flux (w/m2)[+ to atm]
  REAL :: SSOIL      !soil heat flux (w/m2)             [+ to soil]
  REAL :: FIRE       !emitted ir (w/m2)
```

```
    REAL :: TRAD        !radiative temperature (k)
    REAL :: TAH         !"surface" temperature at height z0h+zpd (k)

    REAL :: CW          !water vapor exchange coefficient
    REAL :: FV          !friction velocity (m/s)
    REAL :: WSTAR       !friction velocity n vertical direction (m/s) (only for SFCDIF2)
    REAL :: Z0H         !roughness length, sensible heat, ground (m)
    REAL :: RB          !bulk leaf boundary layer resistance (s/m)
    REAL :: RAMB        !aerodynamic resistance for momentum (s/m)
    REAL :: RAHB        !aerodynamic resistance for sensible heat (s/m)
    REAL :: RAWB        !aerodynamic resistance for water vapor (s/m)
    REAL :: MOL         !Monin-Obukhov length (m)
    REAL :: DTG         !change in tg, last iteration (k)

    REAL :: CIR         !coefficients for ir as function of ts**4
    REAL :: CSH         !coefficients for sh as function of ts
    REAL :: CEV         !coefficients for ev as function of esat[ts]
    REAL :: CGH         !coefficients for st as function of ts

    REAL :: ESTG        !saturation vapor pressure at tg (pa)
    REAL :: DESTG       !d(es)/dt at tg (pa/K)
    REAL :: ESATW       !es for water
    REAL :: ESATI       !es for ice
    REAL :: DSATW       !d(es)/dt at tg (pa/K) for water
    REAL :: DSATI       !d(es)/dt at tg (pa/K) for ice

    REAL :: A           !temporary calculation
    REAL :: B           !temporary calculation
    REAL :: H           !temporary sensible heat flux (w/m2)
    REAL :: MOZ         !Monin-Obukhov stability parameter
    REAL :: MOZOLD      !Monin-Obukhov stability parameter from prior iteration
    REAL :: FM          !momentum stability correction, weighted by prior iters
    REAL :: FH          !sen heat stability correction, weighted by prior iters
    INTEGER :: MOZSGN   !number of times MOZ changes sign

    INTEGER :: ITER     !iteration index
    INTEGER :: NITERB   !number of iterations for surface temperature
    REAL    :: MPE      !prevents overflow error if division by zero

    DATA NITERB /3/
    SAVE NITERB
    REAL :: T, TDC      !Kelvin to degree Celsius with limit -50 to +50
    TDC(T)   = MIN( 50., MAX(-50.,(T-TFRZ)) )

! ------------------------------------------------------------------
! initialization variables that do not depend on stability iteration
! ------------------------------------------------------------------
        MPE = 1E-6
        DTG = 0.
        MOZSGN = 0
        MOZOLD = 0.
        H      = 0.

        CIR = EMG*SB
        CGH = 2.*DF(ISNOW+1)/DZSNSO(ISNOW+1)

! ------------------------------------------------------------------
      DO ITER = 1, NITERB  ! begin stability iteration

        IF(ITER == 1) THEN
            Z0H = Z0M
        ELSE
            Z0H = Z0M !* EXP(-CZIL*0.4*258.2*SQRT(FV*Z0M))
        END IF

        IF(OPT_SFC == 1) THEN
          CALL SFCDIF1(ITER   ,SFCTMP ,RHOAIR ,H      ,QAIR   , & !in
                       ZLVL   ,ZPD    ,Z0M    ,Z0H    ,UR     , & !in
                       MPE    ,ipoint ,                         & !in
```

```
                       MOZ     ,MOZSGN ,FM     ,FH     ,          & !inout
                       CM      ,CH     ,FV     )                    !out
        ENDIF

        IF(OPT_SFC == 2) THEN
          CALL SFCDIF2(ITER    ,ZOM    ,TGB    ,THAIR  ,UR     ,  & !in
                       CZIL    ,ZLVL   ,ipoint ,                  & !in
                       CM      ,CH     ,MOZ    ,WSTAR  ,          & !inout
                       FV      )                                    !out
          IF(SNOWH > 0.) THEN
             CM = MIN(0.01,CM)   ! CM & CH are too large, causing
             CH = MIN(0.01,CH)   ! computational instability
          END IF

        ENDIF

        RAMB = MAX(1.,1./(CM*UR))
        RAHB = MAX(1.,1./(CH*UR))
        RAWB = RAHB

! es and d(es)/dt evaluated at tg

        T = TDC(TGB)
        CALL ESAT(T, ESATW, ESATI, DSATW, DSATI)
        IF (T .GT. 0.) THEN
            ESTG  = ESATW
            DESTG = DSATW
        ELSE
            ESTG  = ESATI
            DESTG = DSATI
        END IF

        CSH = RHOAIR*CPAIR/RAHB
        CEV = RHOAIR*CPAIR/GAMMA/(RSURF+RAWB)

! surface fluxes and dtg

        IRB   = CIR * TGB**4 - EMG*LWDN
        SHB   = CSH * (TGB         - THAIR        )
        EVB   = CEV * (ESTG*RHSUR - EAIR          )
        GHB   = CGH * (TGB         - STC(ISNOW+1))

        B     = SAG-IRB-SHB-EVB-GHB
        A     = 4.*CIR*TGB**3 + CSH + CEV*DESTG + CGH
        DTG   = B/A

        IRB = IRB + 4.*CIR*TGB**3*DTG
        SHB = SHB + CSH*DTG
        EVB = EVB + CEV*DESTG*DTG
        GHB = GHB + CGH*DTG

! update ground surface temperature

        TGB = TGB + DTG

! for M-O length

        H = CSH * (TGB - THAIR)

     END DO ! end stability iteration
! -----------------------------------------------------------------

! if snow on ground and TG > TFRZ: reset TG = TFRZ. reevaluate ground fluxes.

     IF(OPT_STC == 1) THEN
     IF (SNOWH > 0.05 .AND. TGB > TFRZ) THEN
          TGB = TFRZ
          IRB = CIR * TGB**4 - EMG*LWDN
          SHB = CSH * (TGB         - THAIR)
```

```
          EVB = CEV * (ESTG*RHSUR - EAIR )          !ESTG reevaluate ?
          GHB = SAG - (IRB+SHB+EVB)
     END IF
     END IF

! wind stresses

     TAUXB = -RHOAIR*CM*UR*UU
     TAUYB = -RHOAIR*CM*UR*VV

! 2 m height air temperature

     T2MB = TAH - FSH/(RHOAIR*CPAIR*FV) * 1./VKC * LOG((2.+Z0H)/Z0H)

  END SUBROUTINE BARE_FLUX
! ==================================================================================================
  SUBROUTINE RAGRB(ITER   ,VAI    ,RHOAIR ,HG     ,TAH    , & !in
                   ZPD    ,Z0MG   ,Z0HG   ,HCAN   ,UC     , & !in
                   Z0H    ,FV     ,CWP    ,VEGTYP ,MPE    , & !in
                   TV     ,MOZG   ,FHG    ,ipoint ,         & !inout
                   RAMG   ,RAHG   ,RAWG   ,RB     )           !out
! --------------------------------------------------------------------------------------------------
! compute under-canopy aerodynamic resistance RAG and leaf boundary layer
! resistance RB
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs

  INTEGER,              INTENT(IN) :: ipoint !
  INTEGER,              INTENT(IN) :: ITER   !iteration index
  INTEGER,              INTENT(IN) :: VEGTYP !vegetation physiology type
  REAL,                 INTENT(IN) :: VAI    !total LAI + stem area index, one sided
  REAL,                 INTENT(IN) :: RHOAIR !density air (kg/m3)
  REAL,                 INTENT(IN) :: HG     !ground sensible heat flux (w/m2)
  REAL,                 INTENT(IN) :: TV     !vegetation temperature (k)
  REAL,                 INTENT(IN) :: TAH    !air temperature at height z0h+zpd (k)
  REAL,                 INTENT(IN) :: ZPD    !zero plane displacement (m)
  REAL,                 INTENT(IN) :: Z0MG   !roughness length, momentum, ground (m)
  REAL,                 INTENT(IN) :: HCAN   !canopy height (m) [note: hcan >= z0mg]
  REAL,                 INTENT(IN) :: UC     !wind speed at top of canopy (m/s)
  REAL,                 INTENT(IN) :: Z0H    !roughness length, sensible heat (m)
  REAL,                 INTENT(IN) :: Z0HG   !roughness length, sensible heat, ground (m)
  REAL,                 INTENT(IN) :: FV     !friction velocity (m/s)
  REAL,                 INTENT(IN) :: CWP    !canopy wind parameter
  REAL,                 INTENT(IN) :: MPE    !prevents overflow error if division by zero

! in & out

  REAL,              INTENT(INOUT) :: MOZG   !Monin-Obukhov stability parameter
  REAL,              INTENT(INOUT) :: FHG    !stability correction

! outputs
  REAL                             :: RAMG   !aerodynamic resistance for momentum (s/m)
  REAL                             :: RAHG   !aerodynamic resistance for sensible heat (s/m)
  REAL                             :: RAWG   !aerodynamic resistance for water vapor (s/m)
  REAL                             :: RB     !bulk leaf boundary layer resistance (s/m)


  REAL :: KH           !turbulent transfer coefficient, sensible heat, (m2/s)
  REAL :: TMP1         !temporary calculation
  REAL :: TMP2         !temporary calculation
  REAL :: TMPRAH2      !temporary calculation for aerodynamic resistances
  REAL :: TMPRB        !temporary calculation for rb
  real :: MOLG,FHGNEW,CWPC
! --------------------------------------------------------------------------------------------------
! stability correction to below canopy resistance
```

```
       MOZG = 0.
       MOLG = 0.

       IF(ITER > 1) THEN
        TMP1 = VKC * (GRAV/TAH) * HG/(RHOAIR*CPAIR)
        IF (ABS(TMP1) .LE. MPE) TMP1 = MPE
        MOLG = -1. * FV**3 / TMP1
        MOZG = MIN( (ZPD-Z0MG)/MOLG, 1.)
       END IF

       IF (MOZG < 0.) THEN
          FHGNEW  = (1. - 15.*MOZG)**(-0.25)
       ELSE
          FHGNEW  = 1.+ 4.7*MOZG
       ENDIF

       IF (ITER == 1) THEN
          FHG = FHGNEW
       ELSE
          FHG = 0.5 * (FHG+FHGNEW)
       ENDIF

       CWPC = CWP * FHG**0.5

       TMP1 = EXP( -CWPC*Z0HG/HCAN )
       TMP2 = EXP( -CWPC*(Z0H+ZPD)/HCAN )
       TMPRAH2 = HCAN*EXP(CWPC) / CWPC * (TMP1-TMP2)

! aerodynamic resistances raw and rah between heights zpd+z0h and z0hg.

       KH  = MAX ( VKC*FV*(HCAN-ZPD), MPE )
       RAMG = 0.
       RAHG = TMPRAH2 / KH
       RAWG = RAHG

! leaf boundary layer resistance

       TMPRB  = CWPC*50. / (1. - EXP(-CWPC/2.))
       RB     = TMPRB * SQRT(DLEAF(VEGTYP)/UC)

  END SUBROUTINE RAGRB
! ==================================================================================================
  SUBROUTINE SFCDIF1(ITER   ,SFCTMP ,RHOAIR ,H      ,QAIR   , & !in
                     ZLVL   ,ZPD    ,Z0M    ,Z0H    ,UR     , & !in
                     MPE    ,ipoint ,                           & !in
                     MOZ    ,MOZSGN ,FM     ,FH     ,           & !inout
                     CM     ,CH     ,FV     )                     !out
! -------------------------------------------------------------------------------------------------
! computing surface drag coefficient CM for momentum and CH for heat
! -------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! -------------------------------------------------------------------------------------------------
! inputs

  integer              , INTENT(IN) :: ipoint
  INTEGER,               INTENT(IN) :: ITER   !iteration index
  REAL,                  INTENT(IN) :: SFCTMP !temperature at reference height (k)
  REAL,                  INTENT(IN) :: RHOAIR !density air (kg/m**3)
  REAL,                  INTENT(IN) :: H      !sensible heat flux (w/m2) [+ to atm]
  REAL,                  INTENT(IN) :: QAIR   !specific humidity at reference height (kg/kg)
  REAL,                  INTENT(IN) :: ZLVL   !reference height  (m)
  REAL,                  INTENT(IN) :: ZPD    !zero plane displacement (m)
  REAL,                  INTENT(IN) :: Z0H    !roughness length, sensible heat, ground (m)
  REAL,                  INTENT(IN) :: Z0M    !roughness length, momentum, ground (m)
  REAL,                  INTENT(IN) :: UR     !wind speed (m/s)
  REAL,                  INTENT(IN) :: MPE    !prevents overflow error if division by zero
! in & out
```

```
  INTEGER,            INTENT(INOUT) :: MOZSGN !number of times moz changes sign
  REAL,               INTENT(INOUT) :: MOZ    !Monin-Obukhov stability (z/L)
  REAL,               INTENT(INOUT) :: FM     !momentum stability correction, weighted by prior iters
  REAL,               INTENT(INOUT) :: FH     !sen heat stability correction, weighted by prior iters

! outputs

  REAL,                 INTENT(OUT) :: CM     !drag coefficient for momentum
  REAL,                 INTENT(OUT) :: CH     !drag coefficient for heat
  REAL,                 INTENT(OUT) :: FV     !friction velocity (m/s)

! locals
  REAL    :: MOL                      !Monin-Obukhov length (m)
  REAL    :: TMPCM                    !temporary calculation for CM
  REAL    :: TMPCH                    !temporary calculation for CH
  REAL    :: FMNEW                    !stability correction factor, momentum, for current moz
  REAL    :: FHNEW                    !stability correction factor, sen heat, for current moz
  REAL    :: MOZOLD                   !Monin-Obukhov stability parameter from prior iteration
  REAL    :: TMP1,TMP2,TMP3,TMP4,TMP5 !temporary calculation
  REAL    :: TVIR                     !temporary virtual temperature (k)

  REAL    :: CMFM, CHFH
! -------------------------------------------------------------------------------------------------
! Monin-Obukhov stability parameter moz for next iteration

        MOZOLD = MOZ

        IF(ZLVL <= ZPD) THEN
        write(*,*) 'critical problem: ZLVL <= ZPD; model stops'
        STOP
        END IF

        TMPCM = LOG((ZLVL-ZPD) / Z0M)
        TMPCH = LOG((ZLVL-ZPD) / Z0H)

        IF(ITER == 1) THEN
          FV   = 0.0
          MOZ  = 0.0
          MOL  = 0.0
        ELSE
          TVIR = (1. + 0.61*QAIR) * SFCTMP
          TMP1 = VKC * (GRAV/TVIR) * H/(RHOAIR*CPAIR)
          IF (ABS(TMP1) .LE. MPE) TMP1 = MPE
          MOL  = -1. * FV**3 / TMP1
          MOZ  = MIN( (ZLVL-ZPD)/MOL, 1.)
        END IF

! accumulate number of times moz changes sign.

        IF (MOZOLD*MOZ .LT. 0.) MOZSGN = MOZSGN+1
        IF (MOZSGN .GE. 2) THEN
           MOZ = 0.
           FM = 0.
           FH = 0.
        END IF

! evaluate stability-dependent variables using moz from prior iteration
        IF (MOZ .LT. 0.) THEN
             TMP1 = (1. - 16.*MOZ)**0.25
             TMP2 = LOG((1.+TMP1*TMP1)/2.)
             TMP3 = LOG((1.+TMP1)/2.)
             FMNEW = 2.*TMP3 + TMP2 - 2.*ATAN(TMP1) + 1.5707963
             FHNEW = 2*TMP2
        ELSE
             FMNEW = -5.*MOZ
             FHNEW = FMNEW
        ENDIF

! except for first iteration, weight stability factors for previous
```

```
! iteration to help avoid flip-flops from one iteration to the next

        IF (ITER == 1) THEN
             FM = FMNEW
             FH = FHNEW
        ELSE
             FM = 0.5 * (FM+FMNEW)
             FH = 0.5 * (FH+FHNEW)
        ENDIF

! exchange coefficients

        CMFM = TMPCM-FM
        CHFH = TMPCH-FH
        IF(ABS(CMFM) <= MPE) CMFM = MPE
        IF(ABS(CHFH) <= MPE) CHFH = MPE
        CM  = VKC*VKC/(CMFM*CMFM)
        CH  = VKC*VKC/(CMFM*CHFH)

! friction velocity

        FV = UR * SQRT(CM)
! -------------------------------------------------------------------------------------------------
  END SUBROUTINE SFCDIF1
! ==========================================================================================================
  SUBROUTINE SFCDIF2(ITER   ,Z0     ,THZ0   ,THLM   ,SFCSPD ,  & !in
                     CZIL   ,ZLM    ,ipoint ,                  & !in
                     AKMS   ,AKHS   ,RLMO   ,WSTAR2 ,          & !inout
                     USTAR  )                                    !out

! -------------------------------------------------------------------------------------------------------
! SUBROUTINE SFCDIF (renamed SFCDIF_off to avoid clash with Eta PBL)
! -------------------------------------------------------------------------------------------------------
! CALCULATE SURFACE LAYER EXCHANGE COEFFICIENTS VIA ITERATIVE PROCESS.
! SEE CHEN ET AL (1997, BLM)
! -------------------------------------------------------------------------------------------------------

    IMPLICIT NONE
    INTEGER, INTENT(IN) :: ipoint
    INTEGER, INTENT(IN) :: ITER
    REAL,    INTENT(IN) :: ZLM, Z0, THZ0, THLM, SFCSPD, CZIL
    REAL, intent(INOUT) :: AKMS
    REAL, intent(INOUT) :: AKHS
    REAL, intent(INOUT) :: RLMO
    REAL, intent(INOUT) :: WSTAR2
    REAL,   intent(OUT) :: USTAR

    REAL     WWST, WWST2, VKRM, EXCM, BETA, BTG, ELFC, WOLD, WNEW
    REAL     PIHF, EPSU2, EPSUST, EPSIT, EPSA, ZTMIN, ZTMAX, HPBL,      &
     & SQVISC
    REAL     RIC, RRIC, FHNEU, RFC, RFAC, ZZ, PSLMU, PSLMS, PSLHU,      &
     & PSLHS
    REAL     XX, PSPMU, YY, PSPMS, PSPHU, PSPHS
    REAL     ZILFC, ZU, ZT, RDZ, CXCH
    REAL     DTHV, DU2, BTGH, ZSLU, ZSLT, RLOGU, RLOGT
    REAL     ZETALT, ZETALU, ZETAU, ZETAT, XLU4, XLT4, XU4, XT4

    REAL     XLU, XLT, XU, XT, PSMZ, SIMM, PSHZ, SIMH, USTARK, RLMN,   &
     &          RLMA

    INTEGER  ITRMX, ILECH, ITR
    PARAMETER                                                          &
     &         (WWST = 1.2,WWST2 = WWST * WWST,VKRM = 0.40,       &
     &          EXCM = 0.001                                            &
     &         ,BETA = 1./270.,BTG = BETA * GRAV,ELFC = VKRM * BTG        &
     &                    ,WOLD =.15,WNEW = 1.  - WOLD,ITRMX = 05,      &
     &                     PIHF = 3.14159265/2.)
    PARAMETER                                                          &
     &          (EPSU2 = 1.E-4,EPSUST = 0.07,EPSIT = 1.E-4,EPSA = 1.E-8  &
```

```
         &          ,ZTMIN = -5.,ZTMAX = 1.,HPBL = 1000.0                    &
         &           ,SQVISC = 258.2)
    PARAMETER                                                          &
         &        (RIC = 0.183,RRIC = 1.0/ RIC,FHNEU = 0.8,RFC = 0.191     &
         &         ,RFAC = RIC / (FHNEU * RFC * RFC))
! ----------------------------------------------------------------------
! NOTE: THE TWO CODE BLOCKS BELOW DEFINE FUNCTIONS
! ----------------------------------------------------------------------
! LECH'S SURFACE FUNCTIONS
    PSLMU (ZZ)= -0.96* log (1.0-4.5* ZZ)
    PSLMS (ZZ)= ZZ * RRIC -2.076* (1. -1./ (ZZ +1.))
    PSLHU (ZZ)= -0.96* log (1.0-4.5* ZZ)
    PSLHS (ZZ)= ZZ * RFAC -2.076* (1. -1./ (ZZ +1.))
! PAULSON'S SURFACE FUNCTIONS
    PSPMU (XX)= -2.* log ( (XX +1.)*0.5) - log ( (XX * XX +1.)*0.5)    &
         &          +2.* ATAN (XX)                                      &
         &- PIHF
    PSPMS (YY)= 5.* YY
    PSPHU (XX)= -2.* log ( (XX * XX +1.)*0.5)
    PSPHS (YY)= 5.* YY

! THIS ROUTINE SFCDIF CAN HANDLE BOTH OVER OPEN WATER (SEA, OCEAN) AND
! OVER SOLID SURFACE (LAND, SEA-ICE).
! ----------------------------------------------------------------------
!     ZTFC: RATIO OF ZOH/ZOM  LESS OR EQUAL THAN 1
!     C......ZTFC=0.1
!     CZIL: CONSTANT C IN Zilitinkevich, S. S.1995,:NOTE ABOUT ZT
! ----------------------------------------------------------------------
    ILECH = 0

! ----------------------------------------------------------------------
    ZILFC = - CZIL * VKRM * SQVISC
    ZU = Z0
    RDZ = 1./ ZLM
    CXCH = EXCM * RDZ
    DTHV = THLM - THZ0

! BELJARS CORRECTION OF USTAR
    DU2 = MAX (SFCSPD * SFCSPD,EPSU2)
    BTGH = BTG * HPBL

    IF(ITER == 1) THEN
        IF (BTGH * AKHS * DTHV .ne. 0.0) THEN
           WSTAR2 = WWST2* ABS (BTGH * AKHS * DTHV)** (2./3.)
        ELSE
           WSTAR2 = 0.0
        END IF
        USTAR = MAX (SQRT (AKMS * SQRT (DU2+ WSTAR2)),EPSUST)
        RLMO = ELFC * AKHS * DTHV / USTAR **3
    END IF

! ZILITINKEVITCH APPROACH FOR ZT
    ZT = MAX(1.E-6,EXP (ZILFC * SQRT (USTAR * Z0))* Z0)
    ZSLU = ZLM + ZU
    ZSLT = ZLM + ZT
    RLOGU = log (ZSLU / ZU)
    RLOGT = log (ZSLT / ZT)

! ----------------------------------------------------------------------
! 1./MONIN-OBUKKHOV LENGTH-SCALE
! ----------------------------------------------------------------------
!    DO ITR = 1,ITRMX
       ZETALT = MAX (ZSLT * RLMO,ZTMIN)
       RLMO = ZETALT / ZSLT
       ZETALU = ZSLU * RLMO
       ZETAU = ZU * RLMO
       ZETAT = ZT * RLMO

       IF (ILECH .eq. 0) THEN
```

```
          IF (RLMO .lt. 0.)THEN
             XLU4 = 1. -16.* ZETALU
             XLT4 = 1. -16.* ZETALT
             XU4  = 1. -16.* ZETAU
             XT4  = 1. -16.* ZETAT
             XLU  = SQRT (SQRT (XLU4))
             XLT  = SQRT (SQRT (XLT4))
             XU   = SQRT (SQRT (XU4))

             XT = SQRT (SQRT (XT4))
             PSMZ = PSPMU (XU)
             SIMM = PSPMU (XLU) - PSMZ + RLOGU
             PSHZ = PSPHU (XT)
             SIMH = PSPHU (XLT) - PSHZ + RLOGT
          ELSE
             ZETALU = MIN (ZETALU,ZTMAX)
             ZETALT = MIN (ZETALT,ZTMAX)
             PSMZ = PSPMS (ZETAU)
             SIMM = PSPMS (ZETALU) - PSMZ + RLOGU
             PSHZ = PSPHS (ZETAT)
             SIMH = PSPHS (ZETALT) - PSHZ + RLOGT
          END IF
! ----------------------------------------------------------------------
! LECH'S FUNCTIONS
! ----------------------------------------------------------------------
       ELSE
          IF (RLMO .lt. 0.)THEN
             PSMZ = PSLMU (ZETAU)
             SIMM = PSLMU (ZETALU) - PSMZ + RLOGU
             PSHZ = PSLHU (ZETAT)
             SIMH = PSLHU (ZETALT) - PSHZ + RLOGT
          ELSE
             ZETALU = MIN (ZETALU,ZTMAX)
             ZETALT = MIN (ZETALT,ZTMAX)
             PSMZ = PSLMS (ZETAU)
             SIMM = PSLMS (ZETALU) - PSMZ + RLOGU
             PSHZ = PSLHS (ZETAT)
             SIMH = PSLHS (ZETALT) - PSHZ + RLOGT
          END IF
! ----------------------------------------------------------------------
       END IF

! ----------------------------------------------------------------------
! BELJAARS CORRECTION FOR USTAR
! ----------------------------------------------------------------------
       USTAR = MAX (SQRT (AKMS * SQRT (DU2+ WSTAR2)),EPSUST)

! ZILITINKEVITCH FIX FOR ZT
       ZT = MAX(1.E-6,EXP (ZILFC * SQRT (USTAR * Z0))* Z0)
       ZSLT = ZLM + ZT
!-----------------------------------------------------------------------
       RLOGT = log (ZSLT / ZT)
       USTARK = USTAR * VKRM
       AKMS = MAX (USTARK / SIMM,CXCH)
!-----------------------------------------------------------------------
! IF STATEMENTS TO AVOID TANGENT LINEAR PROBLEMS NEAR ZERO
!-----------------------------------------------------------------------
       AKHS = MAX (USTARK / SIMH,CXCH)

       IF (BTGH * AKHS * DTHV .ne. 0.0) THEN
          WSTAR2 = WWST2* ABS (BTGH * AKHS * DTHV)** (2./3.)
       ELSE
          WSTAR2 = 0.0
       END IF
!-----------------------------------------------------------------------
       RLMN = ELFC * AKHS * DTHV / USTAR **3
!-----------------------------------------------------------------------
!     IF(ABS((RLMN-RLMO)/RLMA).LT.EPSIT)    GO TO 110
!-----------------------------------------------------------------------
```

```
       RLMA = RLMO * WOLD+ RLMN * WNEW
!-----------------------------------------------------------------------
       RLMO = RLMA

!       write(*,'(a20,10f15.6)')'SFCDIF: RLMO=',RLMO,RLMN,ELFC , AKHS , DTHV , USTAR
!    END DO
! ----------------------------------------------------------------------
  END SUBROUTINE SFCDIF2
! ==================================================================================
  SUBROUTINE ESAT(T, ESW, ESI, DESW, DESI)
!-----------------------------------------------------------------------------------
! use polynomials to calculate saturation vapor pressure and derivative with
! respect to temperature: over water when t > 0 c and over ice when t <= 0 c
  IMPLICIT NONE
!-----------------------------------------------------------------------------------
! in

  REAL, intent(in)  :: T              !temperature

!out

  REAL, intent(out) :: ESW            !saturation vapor pressure over water (pa)
  REAL, intent(out) :: ESI            !saturation vapor pressure over ice (pa)
  REAL, intent(out) :: DESW           !d(esat)/dt over water (pa/K)
  REAL, intent(out) :: DESI           !d(esat)/dt over ice (pa/K)

! local

  REAL :: A0,A1,A2,A3,A4,A5,A6  !coefficients for esat over water
  REAL :: B0,B1,B2,B3,B4,B5,B6  !coefficients for esat over ice
  REAL :: C0,C1,C2,C3,C4,C5,C6  !coefficients for dsat over water
  REAL :: D0,D1,D2,D3,D4,D5,D6  !coefficients for dsat over ice

  PARAMETER (A0=6.107799961    , A1=4.436518521E-01,  &
             A2=1.428945805E-02, A3=2.650648471E-04,  &
             A4=3.031240396E-06, A5=2.034080948E-08,  &
             A6=6.136820929E-11)

  PARAMETER (B0=6.109177956    , B1=5.034698970E-01,  &
             B2=1.886013408E-02, B3=4.176223716E-04,  &
             B4=5.824720280E-06, B5=4.838803174E-08,  &
             B6=1.838826904E-10)

  PARAMETER (C0= 4.438099984E-01, C1=2.857002636E-02,  &
             C2= 7.938054040E-04, C3=1.215215065E-05,  &
             C4= 1.036561403E-07, C5=3.532421810e-10,  &
             C6=-7.090244804E-13)

  PARAMETER (D0=5.030305237E-01, D1=3.773255020E-02,  &
             D2=1.267995369E-03, D3=2.477563108E-05,  &
             D4=3.005693132E-07, D5=2.158542548E-09,  &
             D6=7.131097725E-12)

  ESW  = 100.*(A0+T*(A1+T*(A2+T*(A3+T*(A4+T*(A5+T*A6))))))
  ESI  = 100.*(B0+T*(B1+T*(B2+T*(B3+T*(B4+T*(B5+T*B6))))))
  DESW = 100.*(C0+T*(C1+T*(C2+T*(C3+T*(C4+T*(C5+T*C6))))))
  DESI = 100.*(D0+T*(D1+T*(D2+T*(D3+T*(D4+T*(D5+T*D6))))))

  END SUBROUTINE ESAT
! ==================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE STOMATA (VEGTYP  ,MPE     ,APAR    ,FOLN    ,ipoint  , & !in
                      TV      ,EI      ,EA      ,SFCTMP  ,SFCPRS  , & !in
                      O2      ,CO2     ,IGS     ,BTRAN   ,RB      , & !in
                      RS      ,PSN     )                              !out
! ---------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! ---------------------------------------------------------------------------------
  IMPLICIT NONE
```

```
! ----------------------------------------------------------------------------------------------------
! input
      INTEGER,INTENT(IN)  :: ipoint
      INTEGER,INTENT(IN)  :: VEGTYP !vegetation physiology type

      REAL, INTENT(IN)    :: IGS    !growing season index (0=off, 1=on)
      REAL, INTENT(IN)    :: MPE    !prevents division by zero errors

      REAL, INTENT(IN)    :: TV     !foliage temperature (k)
      REAL, INTENT(IN)    :: EI     !vapor pressure inside leaf (sat vapor press at tv) (pa)
      REAL, INTENT(IN)    :: EA     !vapor pressure of canopy air (pa)
      REAL, INTENT(IN)    :: APAR   !par absorbed per unit lai (w/m2)
      REAL, INTENT(IN)    :: O2     !atmospheric o2 concentration (pa)
      REAL, INTENT(IN)    :: CO2    !atmospheric co2 concentration (pa)
      REAL, INTENT(IN)    :: SFCPRS !air pressure at reference height (pa)
      REAL, INTENT(IN)    :: SFCTMP !air temperature at reference height (k)
      REAL, INTENT(IN)    :: BTRAN  !soil water transpiration factor (0 to 1)
      REAL, INTENT(IN)    :: FOLN   !foliage nitrogen concentration (%)
      REAL, INTENT(IN)    :: RB     !boundary layer resistance (s/m)

! output
      REAL, INTENT(OUT)   :: RS     !leaf stomatal resistance (s/m)
      REAL, INTENT(OUT)   :: PSN    !foliage photosynthesis (umol co2 /m2/ s) [always +]

! in&out
      REAL                :: RLB    !boundary layer resistance (s m2 / umol)
! ---------------------------------------------------------------------------------------------

! ------------------------ local variables ----------------------------------------------------
      INTEGER :: ITER     !iteration index
      INTEGER :: NITER    !number of iterations

      DATA NITER /3/
      SAVE NITER

      REAL :: AB          !used in statement functions
      REAL :: BC          !used in statement functions
      REAL :: F1          !generic temperature response (statement function)
      REAL :: F2          !generic temperature inhibition (statement function)
      REAL :: TC          !foliage temperature (degree Celsius)
      REAL :: CS          !co2 concentration at leaf surface (pa)
      REAL :: KC          !co2 Michaelis-Menten constant (pa)
      REAL :: KO          !o2 Michaelis-Menten constant (pa)
      REAL :: A,B,C,Q     !intermediate calculations for RS
      REAL :: R1,R2       !roots for RS
      REAL :: FNF         !foliage nitrogen adjustment factor (0 to 1)
      REAL :: PPF         !absorb photosynthetic photon flux (umol photons/m2/s)
      REAL :: WC          !Rubisco limited photosynthesis (umol co2/m2/s)
      REAL :: WJ          !light limited photosynthesis (umol co2/m2/s)
      REAL :: WE          !export limited photosynthesis (umol co2/m2/s)
      REAL :: CP          !co2 compensation point (pa)
      REAL :: CI          !internal co2 (pa)
      REAL :: AWC         !intermediate calculation for wc
      REAL :: VCMX        !maximum rate of carbonylation (umol co2/m2/s)
      REAL :: J           !electron transport (umol co2/m2/s)
      REAL :: CEA         !constrain ea or else model blows up
      REAL :: CF          !s m2/umol -> s/m

      F1(AB,BC) = AB**((BC-25.)/10.)
      F2(AB) = 1. + EXP((-2.2E05+710.*(AB+273.16))/(8.314*(AB+273.16)))
      REAL :: T
! ---------------------------------------------------------------------------------------------

! initialize RS=RSMAX and PSN=0 because will only do calculations
! for APAR > 0, in which case RS <= RSMAX and PSN >= 0

         I = VEGTYP
         CF = SFCPRS/(8.314*SFCTMP)*1.e06
         RS = 1./BP(I) * CF
```

```
         PSN = 0.

         IF (APAR .LE. 0.) RETURN

         I = VEGTYP
         FNF = MIN( FOLN/MAX(MPE,FOLNMX(I)), 1.0 )
         TC  = TV-TFRZ
         PPF = 4.6*APAR
         J   = PPF*QE25(I)
         KC  = KC25(I) * F1(AKC(I),TC)
         KO  = KO25(I) * F1(AKO(I),TC)
         AWC = KC * (1.+O2/KO)
         CP  = 0.5*KC/KO*O2*0.21
         VCMX = VCMX25(I) / F2(TC) * FNF * BTRAN * F1(AVCMX(I),TC)

! first guess ci

         CI = 0.7*CO2*C3PSN(I) + 0.4*CO2*(1.-C3PSN(I))

! rb: s/m -> s m**2 / umol

         RLB = RB/CF

! constrain ea

         CEA = MAX(0.25*EI*C3PSN(I)+0.40*EI*(1.-C3PSN(I)), MIN(EA,EI) )

! ci iteration

       DO ITER = 1, NITER
            I = VEGTYP
            WJ = MAX(CI-CP,0.)*J/(CI+2.*CP)*C3PSN(I)  + J*(1.-C3PSN(I))
            WC = MAX(CI-CP,0.)*VCMX/(CI+AWC)*C3PSN(I) + VCMX*(1.-C3PSN(I))
            WE = 0.5*VCMX*C3PSN(I) + 4000.*VCMX*CI/SFCPRS*(1.-C3PSN(I))
            PSN = MIN(WJ,WC,WE) * IGS

            CS = MAX( CO2-1.37*RLB*SFCPRS*PSN, MPE )
            A = MP(I)*PSN*SFCPRS*CEA / (CS*EI) + BP(I)
            B = ( MP(I)*PSN*SFCPRS/CS + BP(I) ) * RLB - 1.
            C = -RLB
            IF (B .GE. 0.) THEN
               Q = -0.5*( B + SQRT(B*B-4.*A*C) )
            ELSE
               Q = -0.5*( B - SQRT(B*B-4.*A*C) )
            END IF
            R1 = Q/A
            R2 = C/Q
            RS = MAX(R1,R2)
            CI = MAX( CS-PSN*SFCPRS*1.65*RS, 0. )
       END DO

! rs, rb:  s m**2 / umol -> s/m

         RS = RS*CF

  END SUBROUTINE STOMATA
! ==================================================================================================
  SUBROUTINE CANRES (PAR   ,SFCTMP,RCSOIL ,EAH   ,SFCPRS , & !in
                     RC    ,PSN   ,ipoint )                  !out

! --------------------------------------------------------------------------------------------------
! calculate canopy resistance which depends on incoming solar radiation,
! air temperature, atmospheric water vapor pressure deficit at the
! lowest model level, and soil moisture (preferably unfrozen soil
! moisture rather than total)
! --------------------------------------------------------------------------------------------------
! source:  Jarvis (1976), Noilhan and Planton (1989, MWR), Jacquemin and
! Noilhan (1990, BLM). Chen et al (1996, JGR, Vol 101(D3), 7251-7268),
! eqns 12-14 and table 2 of sec. 3.1.2
```

```
! ----------------------------------------------------------------------------------------------------
    USE module_Noahlsm_utility
! ----------------------------------------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------------------------------------
! inputs

    integer,                  INTENT(IN)  :: ipoint
    REAL,                     INTENT(IN)  :: PAR    !par absorbed per unit sunlit lai (w/m2)
    REAL,                     INTENT(IN)  :: SFCTMP !canopy air temperature
    REAL,                     INTENT(IN)  :: SFCPRS !surface pressure (pa)
    REAL,                     INTENT(IN)  :: EAH    !water vapor pressure (pa)
    REAL,                     INTENT(IN)  :: RCSOIL !soil moisture stress factor

!outputs

    REAL,                     INTENT(OUT) :: RC     !canopy resistance per unit LAI
    REAL,                     INTENT(OUT) :: PSN    !foliage photosynthesis (umolco2/m2/s)

!local

    REAL                                  :: RCQ
    REAL                                  :: RCS
    REAL                                  :: RCT
    REAL                                  :: FF
    REAL                                  :: Q2     !water vapor mixing ratio (kg/kg)
    REAL                                  :: Q2SAT  !saturation Q2
    REAL                                  :: DQSDT2 !d(Q2SAT)/d(T)

! RSMIN, RSMAX, TOPT, RGL, HS are canopy stress parameters set in REDPRM
! ----------------------------------------------------------------------
! initialize canopy resistance multiplier terms.
! ----------------------------------------------------------------------
    RC     = 0.0
    RCS    = 0.0
    RCT    = 0.0
    RCQ    = 0.0

!  compute Q2 and Q2SAT

    Q2 = 0.622 *  EAH  / (SFCPRS - 0.378 * EAH) !specific humidity [kg/kg]
    Q2 = Q2 / (1.0 + Q2)                        !mixing ratio [kg/kg]

    CALL CALHUM(SFCTMP, SFCPRS, Q2SAT, DQSDT2)

! contribution due to incoming solar radiation

    FF  = 2.0 * PAR / RGL
    RCS = (FF + RSMIN / RSMAX) / (1.0+ FF)
    RCS = MAX (RCS,0.0001)

! contribution due to air temperature

    RCT = 1.0- 0.0016* ( (TOPT - SFCTMP)**2.0)
    RCT = MAX (RCT,0.0001)

! contribution due to vapor pressure deficit

    RCQ = 1.0/ (1.0+ HS * MAX(0.,Q2SAT-Q2))
    RCQ = MAX (RCQ,0.01)

! determine canopy resistance due to all factors

    RC  = RSMIN / (RCS * RCT * RCQ * RCSOIL)
    PSN = -999.99       ! PSN not applied for dynamic carbon

  END SUBROUTINE CANRES
! ====================================================================================================
  SUBROUTINE TSNOSOI (ICE     ,NSOIL   ,NSNOW   ,ISNOW   ,IST     , & !in
```

```
                     TBOT    ,ZSNSO   ,SSOIL   ,DF      ,HCPCT   ,  & !in
                     ZBOT    ,SAG     ,DT      ,SNOWH   ,DZSNSO  ,  & !in
                     TG      ,ipoint  ,                              & !in
                     STC     )                                         !inout
! ----------------------------------------------------------------------------------------------------
! Compute snow (up to 3L) and soil (4L) temperature. Note that snow temperatures
! during melting season may exceed melting point (TFRZ) but later in PHASECHANGE
! subroutine the snow temperatures are reset to TFRZ for melting snow.
! ----------------------------------------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------------------------------------
!input

    integer,                         INTENT(IN)  :: ipoint
    INTEGER,                         INTENT(IN)  :: ICE    !
    INTEGER,                         INTENT(IN)  :: NSOIL  !no of soil layers (4)
    INTEGER,                         INTENT(IN)  :: NSNOW  !maximum no of snow layers (3)
    INTEGER,                         INTENT(IN)  :: ISNOW  !actual no of snow layers
    INTEGER,                         INTENT(IN)  :: IST    !surface type

    REAL,                            INTENT(IN)  :: DT     !time step (s)
    REAL,                            INTENT(IN)  :: TBOT   !
    REAL,                            INTENT(IN)  :: SSOIL  !ground heat flux (w/m2)
    REAL,                            INTENT(IN)  :: SAG    !solar rad. absorbed by ground (w/m2)
    REAL,                            INTENT(IN)  :: SNOWH  !snow depth (m)
    REAL,                            INTENT(IN)  :: ZBOT   !from soil surface (m)
    REAL,                            INTENT(IN)  :: TG     !ground temperature (k)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: ZSNSO  !layer-bot. depth from snow surf.(m)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: DZSNSO !snow/soil layer thickness (m)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: DF     !thermal conductivity
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: HCPCT  !heat capacity (J/m3/k)

!input and output

    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC

!local

    INTEGER                                      :: IZ
    REAL                                         :: ZBOTSNO   !ZBOT from snow surface
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: AI, BI, CI, RHSTS
    REAL                                         :: EFLXB !energy influx from soil bottom (w/m2)
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: PHI   !light through water (w/m2)

    REAL                          :: BEG_EST !heat storage of snow/soil before updating STC (J/m2)
    REAL                          :: END_EST !heat storage of snow/soil after updating STC (J/m2)
    REAL                          :: ERR_EST !heat storage error  (w/m2)
    REAL                          :: SSOIL2  !ground heat flux (w/m2) (for energy check)
    REAL                          :: EFLXB2  !heat flux from the bottom (w/m2) (for energy check)
! ----------------------------------------------------------------------
! compute solar penetration through water, needs more work

    PHI(ISNOW+1:NSOIL) = 0.

! adjust ZBOT from soil surface to ZBOTSNO from snow surface

    ZBOTSNO = ZBOT - SNOWH    !from snow surface

! snow/soil heat storage for energy balance check

    BEG_EST = 0.
    DO IZ = ISNOW+1, NSOIL
      BEG_EST = BEG_EST  + STC(IZ)*DZSNSO(IZ)*HCPCT(IZ)
    END DO

! compute soil temperatures

      CALL HRT   (NSNOW     ,NSOIL     ,ISNOW     ,ZSNSO     ,  &
                  STC       ,TBOT      ,ZBOTSNO   ,DT        ,  &
```

```
                  DF         ,HCPCT      ,SSOIL      ,PHI        ,  &
                  AI         ,BI         ,CI         ,RHSTS      ,  &
                  EFLXB      )

      CALL HSTEP  (NSNOW      ,NSOIL      ,ISNOW      ,DT         ,  &
                  AI         ,BI         ,CI         ,RHSTS      ,  &
                  STC        )

    END_EST = 0.
    DO IZ = ISNOW+1, NSOIL
      END_EST = END_EST  + STC(IZ)*DZSNSO(IZ)*HCPCT(IZ)-PHI(IZ)*DT
    END DO

! update ground heat flux just for energy check, but not for final output
! otherwise, it would break the surface energy balance

    SSOIL2 = DF(ISNOW+1)*(TG-STC(ISNOW+1))/(0.5*DZSNSO(ISNOW+1))   !M. Barlage

    IF(OPT_TBOT == 1) THEN
       EFLXB2  = 0.
    END IF
    IF(OPT_TBOT == 2) THEN
       EFLXB2  = DF(NSOIL)*(TBOT-STC(NSOIL)) / &
                (0.5*(ZSNSO(NSOIL-1)+ZSNSO(NSOIL)) - ZBOTSNO)
    END IF

! energy balance check

    IF(OPT_STC == 1) THEN   ! semi-implicit
      ERR_EST = (END_EST-BEG_EST) - (SSOIL +EFLXB )*DT
    ELSE                    ! full-implicit
      ERR_EST = (END_EST-BEG_EST) - (SSOIL2+EFLXB2)*DT
    END IF

    ERR_EST = ERR_EST / DT

    IF (ERR_EST > 1.) THEN    ! W/m2
      WRITE(*,*) 'TSNOSOI is losing(-)/gaining(+) false energy',ERR_EST,' W/m2'
      WRITE(*,'(i6,i3,F8.3,2F20.3,5F10.2)') &
      ipoint, IST,ERR_EST,END_EST,BEG_EST,SSOIL,SNOWH,TG,STC(ISNOW+1),EFLXB
      STOP
    END IF

  END SUBROUTINE TSNOSOI
! ==================================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE HRT (NSNOW      ,NSOIL      ,ISNOW      ,ZSNSO      ,  &
                  STC        ,TBOT       ,ZBOT       ,DT         ,  &
                  DF         ,HCPCT      ,SSOIL      ,PHI        ,  &
                  AI         ,BI         ,CI         ,RHSTS      ,  &
                  BOTFLX     )
! ----------------------------------------------------------------------
! ----------------------------------------------------------------------
! calculate the right hand side of the time tendency term of the soil
! thermal diffusion equation.  also to compute ( prepare ) the matrix
! coefficients for the tri-diagonal matrix of the implicit time scheme.
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input

    INTEGER,                         INTENT(IN)  :: NSOIL  !no of soil layers (4)
    INTEGER,                         INTENT(IN)  :: NSNOW  !maximum no of snow layers (3)
    INTEGER,                         INTENT(IN)  :: ISNOW  !actual no of snow layers
    REAL,                            INTENT(IN)  :: TBOT   !bottom soil temp. at ZBOT (k)
    REAL,                            INTENT(IN)  :: ZBOT   !depth of lower boundary condition (m)
                                                           !from soil surface not snow surface
    REAL,                            INTENT(IN)  :: DT     !time step (s)
    REAL,                            INTENT(IN)  :: SSOIL  !ground heat flux (w/m2)
```

```
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: ZSNSO  !depth of layer-bottom of snow/soil (m)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: STC    !snow/soil temperature (k)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: DF     !thermal conductivity [w/m/k]
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: HCPCT  !heat capacity [j/m3/k]
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: PHI    !light through water (w/m2)

! output

    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: RHSTS  !right-hand side of the matrix
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: AI     !left-hand side coefficient
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: BI     !left-hand side coefficient
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: CI     !left-hand side coefficient
    REAL,                            INTENT(OUT) :: BOTFLX !energy influx from soil bottom (w/m2)

! local

    INTEGER                                      :: K
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: DDZ
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: DZ
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: DENOM
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: DTSDZ
    REAL, DIMENSION(-NSNOW+1:NSOIL)              :: EFLUX
    REAL                                         :: TEMP1
! ----------------------------------------------------------------------

    DO K = ISNOW+1, NSOIL
        IF (K == ISNOW+1) THEN
           DENOM(K)  = - ZSNSO(K) * HCPCT(K)
           TEMP1     = - ZSNSO(K+1)
           DDZ(K)    = 2.0 / TEMP1
           DTSDZ(K)  = 2.0 * (STC(K) - STC(K+1)) / TEMP1
           EFLUX(K)  = DF(K) * DTSDZ(K) - SSOIL - PHI(K)
        ELSE IF (K < NSOIL) THEN
           DENOM(K)  = (ZSNSO(K-1) - ZSNSO(K)) * HCPCT(K)
           TEMP1     = ZSNSO(K-1) - ZSNSO(K+1)
           DDZ(K)    = 2.0 / TEMP1
           DTSDZ(K)  = 2.0 * (STC(K) - STC(K+1)) / TEMP1
           EFLUX(K)  = (DF(K)*DTSDZ(K) - DF(K-1)*DTSDZ(K-1)) - PHI(K)
        ELSE IF (K == NSOIL) THEN
           DENOM(K)  = (ZSNSO(K-1) - ZSNSO(K)) * HCPCT(K)
           TEMP1     =  ZSNSO(K-1) - ZSNSO(K)
           IF(OPT_TBOT == 1) THEN
               BOTFLX     = 0.
           END IF
           IF(OPT_TBOT == 2) THEN
               DTSDZ(K)  = (STC(K) - TBOT) / ( 0.5*(ZSNSO(K-1)+ZSNSO(K)) - ZBOT)
               BOTFLX    = -DF(K) * DTSDZ(K)
           END IF
           EFLUX(K)  = (-BOTFLX - DF(K-1)*DTSDZ(K-1) ) - PHI(K)
        END IF
    END DO

    DO K = ISNOW+1, NSOIL
        IF (K == ISNOW+1) THEN
           AI(K)    =   0.0
           CI(K)    = - DF(K)   * DDZ(K) / DENOM(K)
           IF (OPT_STC == 1) THEN
              BI(K) = - CI(K)
           END IF
           IF (OPT_STC == 2) THEN
              BI(K) = - CI(K) + DF(K)/(0.5*ZSNSO(K)*ZSNSO(K)*HCPCT(K))
           END IF
        ELSE IF (K < NSOIL) THEN
           AI(K)    = - DF(K-1) * DDZ(K-1) / DENOM(K)
           CI(K)    = - DF(K  ) * DDZ(K  ) / DENOM(K)
           BI(K)    = - (AI(K) + CI (K))
        ELSE IF (K == NSOIL) THEN
           AI(K)    = - DF(K-1) * DDZ(K-1) / DENOM(K)
           CI(K)    = 0.0
```

```
          BI(K)    = - (AI(K) + CI(K))
       END IF
          RHSTS(K)  = EFLUX(K)/ (-DENOM(K))
    END DO

  END SUBROUTINE HRT
! ==================================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE HSTEP (NSNOW     ,NSOIL     ,ISNOW     ,DT        ,  &
                    AI        ,BI        ,CI        ,RHSTS     ,  &
                    STC       )
! ----------------------------------------------------------------------
! CALCULATE/UPDATE THE SOIL TEMPERATURE FIELD.
! ----------------------------------------------------------------------
    implicit none
! ----------------------------------------------------------------------
! input

    INTEGER,                         INTENT(IN)    :: NSOIL
    INTEGER,                         INTENT(IN)    :: NSNOW
    INTEGER,                         INTENT(IN)    :: ISNOW
    REAL,                            INTENT(IN)    :: DT

! output & input
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: RHSTS
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: AI
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: BI
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: CI
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC

! local
    INTEGER                                        :: K
    REAL, DIMENSION(-NSNOW+1:NSOIL)                :: RHSTSIN
    REAL, DIMENSION(-NSNOW+1:NSOIL)                :: CIIN
! ----------------------------------------------------------------------

    DO K = ISNOW+1,NSOIL
       RHSTS(K) =   RHSTS(K) * DT
       AI(K)    =      AI(K) * DT
       BI(K)    = 1. + BI(K) * DT
       CI(K)    =      CI(K) * DT
    END DO

! copy values for input variables before call to rosr12

    DO K = ISNOW+1,NSOIL
       RHSTSIN(K) = RHSTS(K)
       CIIN(K)    = CI(K)
    END DO

! solve the tri-diagonal matrix equation

    CALL ROSR12 (CI,AI,BI,CIIN,RHSTSIN,RHSTS,ISNOW+1,NSOIL,NSNOW)

! update snow & soil temperature

    DO K = ISNOW+1,NSOIL
       STC (K) = STC (K) + CI (K)
    END DO

  END SUBROUTINE HSTEP
! ==================================================================================================
  SUBROUTINE ROSR12 (P,A,B,C,D,DELTA,NTOP,NSOIL,NSNOW)
! ----------------------------------------------------------------------
! SUBROUTINE ROSR12
! ----------------------------------------------------------------------
! INVERT (SOLVE) THE TRI-DIAGONAL MATRIX PROBLEM SHOWN BELOW:
! ###                                            ### ###  ###   ###  ###
! #B(1), C(1),  0  ,  0  ,  0  ,   . . .  ,    0  # #      #   #      #
```

```
! #A(2), B(2), C(2),  0  ,  0  ,   . . . ,    0   # #      #   #     #
! # 0  , A(3), B(3), C(3),  0  ,   . . . ,    0   # #      #   # D(3) #
! # 0  ,  0  , A(4), B(4), C(4),   . . . ,    0   # # P(4) #   # D(4) #
! # 0  ,  0  ,  0  , A(5), B(5),   . . . ,    0   # # P(5) #   # D(5) #
! # .                                       .   # #  .   # = #   .  #
! # .                                       .   # #  .   #   #   .  #
! # .                                       .   # #  .   #   #   .  #
! # 0  , . . . , 0 , A(M-2), B(M-2), C(M-2),   0   # #P(M-2)#   #D(M-2)#
! # 0  , . . . , 0 ,   0   , A(M-1), B(M-1), C(M-1)# #P(M-1)#   #D(M-1)#
! # 0  , . . . , 0 ,   0   ,   0   ,  A(M) ,  B(M) # # P(M) #   # D(M) #
! ###                                          ### ###  ###   ###  ###
! ----------------------------------------------------------------------
    IMPLICIT NONE

    INTEGER, INTENT(IN)    :: NTOP
    INTEGER, INTENT(IN)    :: NSOIL,NSNOW
    INTEGER                :: K, KK

    REAL, DIMENSION(-NSNOW+1:NSOIL),INTENT(IN):: A, B, D
    REAL, DIMENSION(-NSNOW+1:NSOIL),INTENT(INOUT):: C,P,DELTA

! ----------------------------------------------------------------------
! INITIALIZE EQN COEF C FOR THE LOWEST SOIL LAYER
! ----------------------------------------------------------------------
    C (NSOIL) = 0.0
    P (NTOP) = - C (NTOP) / B (NTOP)
! ----------------------------------------------------------------------
! SOLVE THE COEFS FOR THE 1ST SOIL LAYER
! ----------------------------------------------------------------------
    DELTA (NTOP) = D (NTOP) / B (NTOP)
! ----------------------------------------------------------------------
! SOLVE THE COEFS FOR SOIL LAYERS 2 THRU NSOIL
! ----------------------------------------------------------------------
    DO K = NTOP+1,NSOIL
       P (K) = - C (K) * ( 1.0 / (B (K) + A (K) * P (K -1)) )
       DELTA (K) = (D (K) - A (K)* DELTA (K -1))* (1.0/ (B (K) + A (K)&
            * P (K -1)))
    END DO
! ----------------------------------------------------------------------
! SET P TO DELTA FOR LOWEST SOIL LAYER
! ----------------------------------------------------------------------
    P (NSOIL) = DELTA (NSOIL)
! ----------------------------------------------------------------------
! ADJUST P FOR SOIL LAYERS 2 THRU NSOIL
! ----------------------------------------------------------------------
    DO K = NTOP+1,NSOIL
       KK = NSOIL - K + (NTOP-1) + 1
       P (KK) = P (KK) * P (KK +1) + DELTA (KK)
    END DO
! ----------------------------------------------------------------------
  END SUBROUTINE ROSR12
! ----------------------------------------------------------------------
! ==================================================================================================
  SUBROUTINE PHASECHANGE (NSNOW   ,NSOIL   ,ISNOW   ,DT      ,FACT    , & !in
                          DZSNSO  ,HCPCT   ,IST     ,ipoint  ,          & !in
                          STC     ,SNICE   ,SNLIQ   ,SNEQV   ,SNOWH   , & !inout
                          SMC     ,SH2O    ,                            & !inout
                          QMELT   ,IMELT   ,PONDING )                     !out
! ----------------------------------------------------------------------
! melting/freezing of snow water and soil water
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! inputs

  INTEGER, INTENT(IN)                             :: ipoint !
  INTEGER, INTENT(IN)                             :: NSNOW  !maximum no. of snow layers [=3]
  INTEGER, INTENT(IN)                             :: NSOIL  !No. of soil layers [=4]
  INTEGER, INTENT(IN)                             :: ISNOW  !actual no. of snow layers [<=3]
```

```fortran
  INTEGER, INTENT(IN)                              :: IST     !surface type: 1->soil; 2->lake
  REAL, INTENT(IN)                                 :: DT      !land model time step (sec)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)      :: FACT    !temporary
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)      :: DZSNSO  !snow/soil layer thickness [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)      :: HCPCT   !heat capacity (J/m3/k)

! outputs
  INTEGER, DIMENSION(-NSNOW+1:NSOIL), INTENT(OUT) :: IMELT  !phase change index
  REAL,                               INTENT(OUT) :: QMELT  !snowmelt rate [mm/s]
  REAL,                               INTENT(OUT) :: PONDING!snowmelt when snow has no layer [mm]

! inputs and outputs

  REAL, INTENT(INOUT) :: SNEQV
  REAL, INTENT(INOUT) :: SNOWH
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT)   :: STC     !snow/soil layer temperature [k]
  REAL, DIMENSION(        1:NSOIL), INTENT(INOUT)  :: SH2O    !soil liquid water [m3/m3]
  REAL, DIMENSION(        1:NSOIL), INTENT(INOUT)  :: SMC     !total soil water [m3/m3]
  REAL, DIMENSION(-NSNOW+1:0)     , INTENT(INOUT)  :: SNICE   !snow layer ice [mm]
  REAL, DIMENSION(-NSNOW+1:0)     , INTENT(INOUT)  :: SNLIQ   !snow layer liquid water [mm]

! local

  INTEGER                         :: J         !do loop index
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: HM        !energy residual [w/m2]
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: XM        !melting or freezing water [kg/m2]
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: WMASS0
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: WICE0
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: WLIQ0
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: MICE      !soil/snow ice mass [mm]
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: MLIQ      !soil/snow liquid water mass [mm]
  REAL, DIMENSION(-NSNOW+1:NSOIL) :: SUPERCOOL !supercooled water in soil (kg/m2)
  REAL                            :: HEATR     !energy residual or loss after melting/freezing
  REAL                            :: TEMP1     !temporary variables [kg/m2]
  REAL                            :: PROPOR
  REAL                            :: SMP       !frozen water potential (mm)
  REAL                            :: XMF       !total latent heat of phase change

! ----------------------------------------------------------------------
! Initialization

    QMELT   = 0.
    PONDING = 0.
    XMF     = 0.

    DO J = -NSNOW+1, NSOIL
         SUPERCOOL(J) = 0.0
    END DO

    DO J = ISNOW+1,0       ! all layers
         MICE(J) = SNICE(J)
         MLIQ(J) = SNLIQ(J)
    END DO

    DO J = 1, NSOIL               ! soil
         MLIQ(J) =  SH2O(J)            * DZSNSO(J) * 1000.
         MICE(J) = (SMC(J) - SH2O(J))  * DZSNSO(J) * 1000.
    END DO

    DO J = ISNOW+1,NSOIL       ! all layers
         IMELT(J)    = 0
         HM(J)       = 0.
         XM(J)       = 0.
         WICE0(J)    = MICE(J)
         WLIQ0(J)    = MLIQ(J)
         WMASS0(J)   = MICE(J) + MLIQ(J)
    ENDDO

    if(ist == 1) then
```

```
       DO J = 1,NSOIL
          IF (OPT_FRZ == 1) THEN
             IF(STC(J) < TFRZ) THEN
                SMP = HFUS*(TFRZ-STC(J))/(GRAV*STC(J))              !(m)
                SUPERCOOL(J) = SMCMAX*(SMP/PSISAT)**(-1./BEXP)
                SUPERCOOL(J) = SUPERCOOL(J)*DZSNSO(J)*1000.        !(mm)
             END IF
          END IF
          IF (OPT_FRZ == 2) THEN
                CALL FRH2O (SUPERCOOL(J),STC(J),SMC(J),SH2O(J))
                SUPERCOOL(J) = SUPERCOOL(J)*DZSNSO(J)*1000.        !(mm)
          END IF
       ENDDO
     end if

     DO J = ISNOW+1,NSOIL
          IF (MICE(J) > 0. .AND. STC(J) >= TFRZ) THEN  !melting
              IMELT(J) = 1
          ENDIF
          IF (MLIQ(J) > SUPERCOOL(J) .AND. STC(J) < TFRZ) THEN
              IMELT(J) = 2
          ENDIF

          ! If snow exists, but its thickness is not enough to create a layer
          IF (ISNOW == 0 .AND. SNEQV > 0. .AND. J == 1) THEN
              IF (STC(J) >= TFRZ) THEN
                 IMELT(J) = 1
              ENDIF
          ENDIF
     ENDDO

! Calculate the energy surplus and loss for melting and freezing

     DO J = ISNOW+1,NSOIL
          IF (IMELT(J) > 0) THEN
              HM(J) = (STC(J)-TFRZ)/FACT(J)
              STC(J) = TFRZ
          ENDIF

          IF (IMELT(J) == 1 .AND. HM(J) < 0.) THEN
             HM(J) = 0.
             IMELT(J) = 0
          ENDIF
          IF (IMELT(J) == 2 .AND. HM(J) > 0.) THEN
             HM(J) = 0.
             IMELT(J) = 0
          ENDIF
          XM(J) = HM(J)*DT/HFUS
     ENDDO

! The rate of melting and freezing for snow without a layer, needs more work.

     IF (ISNOW == 0 .AND. SNEQV > 0. .AND. XM(1) > 0.) THEN
         TEMP1  = SNEQV
         SNEQV  = MAX(0.,TEMP1-XM(1))
         PROPOR = SNEQV/TEMP1
         SNOWH  = MAX(0.,PROPOR * SNOWH)
         HEATR  = HM(1) - HFUS*(TEMP1-SNEQV)/DT
         IF (HEATR > 0.) THEN
               XM(1) = HEATR*DT/HFUS
               HM(1) = HEATR
         ELSE
               XM(1) = 0.
               HM(1) = 0.
         ENDIF
         QMELT   = MAX(0.,(TEMP1-SNEQV))/DT
         XMF     = HFUS*QMELT
         PONDING = TEMP1-SNEQV
     ENDIF
```

```
! The rate of melting and freezing for snow and soil

    DO J = ISNOW+1,NSOIL
      IF (IMELT(J) > 0 .AND. ABS(HM(J)) > 0.) THEN

         HEATR = 0.
         IF (XM(J) > 0.) THEN
            MICE(J) = MAX(0., WICE0(J)-XM(J))
            HEATR = HM(J) - HFUS*(WICE0(J)-MICE(J))/DT
         ELSE IF (XM(J) < 0.) THEN
            IF (J <= 0) THEN                        ! snow
               MICE(J) = MIN(WMASS0(J), WICE0(J)-XM(J))
            ELSE                                    ! soil
               IF (WMASS0(J) < SUPERCOOL(J)) THEN
                  MICE(J) = 0.
               ELSE
                  MICE(J) = MIN(WMASS0(J) - SUPERCOOL(J),WICE0(J)-XM(J))
                  MICE(J) = MAX(MICE(J),0.0)
               ENDIF
            ENDIF
            HEATR = HM(J) - HFUS*(WICE0(J)-MICE(J))/DT
         ENDIF

         MLIQ(J) = MAX(0.,WMASS0(J)-MICE(J))

         IF (ABS(HEATR) > 0.) THEN
            STC(J) = STC(J) + FACT(J)*HEATR
            IF (J <= 0) THEN                        ! snow
               IF (MLIQ(J)*MICE(J)>0.) STC(J) = TFRZ
            END IF
         ENDIF

         XMF = XMF + HFUS * (WICE0(J)-MICE(J))/DT

         IF (J < 1) THEN
            QMELT = QMELT + MAX(0.,(WICE0(J)-MICE(J)))/DT
         ENDIF
      ENDIF
    ENDDO

    DO J = ISNOW+1,0             ! snow
       SNLIQ(J) = MLIQ(J)
       SNICE(J) = MICE(J)
    END DO

    DO J = 1, NSOIL              ! soil
       SH2O(J) =  MLIQ(J)            / (1000. * DZSNSO(J))
       SMC(J)  = (MLIQ(J) + MICE(J)) / (1000. * DZSNSO(J))
    END DO

  END SUBROUTINE PHASECHANGE
! ==================================================================================================
  SUBROUTINE FRH2O (FREE,TKELV,SMC,SH2O)

! ----------------------------------------------------------------------
! SUBROUTINE FRH2O
! ----------------------------------------------------------------------
! CALCULATE AMOUNT OF SUPERCOOLED LIQUID SOIL WATER CONTENT IF
! TEMPERATURE IS BELOW 273.15K (TFRZ).  REQUIRES NEWTON-TYPE ITERATION
! TO SOLVE THE NONLINEAR IMPLICIT EQUATION GIVEN IN EQN 17 OF KOREN ET AL
! (1999, JGR, VOL 104(D16), 19569-19585).
! ----------------------------------------------------------------------
! NEW VERSION (JUNE 2001): MUCH FASTER AND MORE ACCURATE NEWTON
! ITERATION ACHIEVED BY FIRST TAKING LOG OF EQN CITED ABOVE -- LESS THAN
! 4 (TYPICALLY 1 OR 2) ITERATIONS ACHIEVES CONVERGENCE.  ALSO, EXPLICIT
! 1-STEP SOLUTION OPTION FOR SPECIAL CASE OF PARAMETER CK=0, WHICH
! REDUCES THE ORIGINAL IMPLICIT EQUATION TO A SIMPLER EXPLICIT FORM,
! KNOWN AS THE "FLERCHINGER EQN". IMPROVED HANDLING OF SOLUTION IN THE
```

```
! LIMIT OF FREEZING POINT TEMPERATURE TFRZ.
! ----------------------------------------------------------------------
! INPUT:

!   TKELV.........TEMPERATURE (Kelvin)
!   SMC...........TOTAL SOIL MOISTURE CONTENT (VOLUMETRIC)
!   SH2O..........LIQUID SOIL MOISTURE CONTENT (VOLUMETRIC)
!   B.............SOIL TYPE "B" PARAMETER (FROM REDPRM)
!   PSISAT........SATURATED SOIL MATRIC POTENTIAL (FROM REDPRM)

! OUTPUT:
!   FREE..........SUPERCOOLED LIQUID WATER CONTENT [m3/m3]
! ----------------------------------------------------------------------
    IMPLICIT NONE
    REAL, INTENT(IN)     :: SH2O,SMC,TKELV
    REAL, INTENT(OUT)    :: FREE
    REAL                 :: BX,DENOM,DF,DSWL,FK,SWL,SWLK
    INTEGER              :: NLOG,KCOUNT
!      PARAMETER(CK = 0.0)
    REAL, PARAMETER      :: CK = 8.0, BLIM = 5.5, ERROR = 0.005,       &
         DICE = 920.0

! ----------------------------------------------------------------------
! LIMITS ON PARAMETER B: B < 5.5  (use parameter BLIM)
! SIMULATIONS SHOWED IF B > 5.5 UNFROZEN WATER CONTENT IS
! NON-REALISTICALLY HIGH AT VERY LOW TEMPERATURES.
! ----------------------------------------------------------------------
    BX = BEXP
! ----------------------------------------------------------------------
! INITIALIZING ITERATIONS COUNTER AND ITERATIVE SOLUTION FLAG.
! ----------------------------------------------------------------------

    IF (BEXP >  BLIM) BX = BLIM
    NLOG = 0

! ----------------------------------------------------------------------
!  IF TEMPERATURE NOT SIGNIFICANTLY BELOW FREEZING (TFRZ), SH2O = SMC
! ----------------------------------------------------------------------
    KCOUNT = 0
    IF (TKELV > (TFRZ- 1.E-3)) THEN
       FREE = SMC
    ELSE

! ----------------------------------------------------------------------
! OPTION 1: ITERATED SOLUTION IN KOREN ET AL, JGR, 1999, EQN 17
! ----------------------------------------------------------------------
! INITIAL GUESS FOR SWL (frozen content)
! ----------------------------------------------------------------------
       IF (CK /= 0.0) THEN
          SWL = SMC - SH2O
! ----------------------------------------------------------------------
! KEEP WITHIN BOUNDS.
! ----------------------------------------------------------------------
          IF (SWL > (SMC -0.02)) SWL = SMC -0.02
! ----------------------------------------------------------------------
!  START OF ITERATIONS
! ----------------------------------------------------------------------
          IF (SWL < 0.) SWL = 0.
1001      Continue
          IF (.NOT.( (NLOG < 10) .AND. (KCOUNT == 0)))   goto 1002
          NLOG = NLOG +1
          DF = ALOG ( ( PSISAT * GRAV / hfus ) * ( ( 1. + CK * SWL )**2.) * &
               ( SMCMAX / (SMC - SWL) )** BX) - ALOG ( - (                 &
               TKELV - TFRZ)/ TKELV)
          DENOM = 2. * CK / ( 1. + CK * SWL ) + BX / ( SMC - SWL )
          SWLK = SWL - DF / DENOM
! ----------------------------------------------------------------------
! BOUNDS USEFUL FOR MATHEMATICAL SOLUTION.
! ----------------------------------------------------------------------
```

```
          IF (SWLK > (SMC -0.02)) SWLK = SMC - 0.02
          IF (SWLK < 0.) SWLK = 0.

! ----------------------------------------------------------------------
! MATHEMATICAL SOLUTION BOUNDS APPLIED.
! ----------------------------------------------------------------------
          DSWL = ABS (SWLK - SWL)
! IF MORE THAN 10 ITERATIONS, USE EXPLICIT METHOD (CK=0 APPROX.)
! WHEN DSWL LESS OR EQ. ERROR, NO MORE ITERATIONS REQUIRED.
! ----------------------------------------------------------------------
          SWL = SWLK
          IF ( DSWL <= ERROR ) THEN
             KCOUNT = KCOUNT +1
          END IF
! ----------------------------------------------------------------------
!  END OF ITERATIONS
! ----------------------------------------------------------------------
! BOUNDS APPLIED WITHIN DO-BLOCK ARE VALID FOR PHYSICAL SOLUTION.
! ----------------------------------------------------------------------
          goto 1001
1002      continue
          FREE = SMC - SWL
       END IF
! ----------------------------------------------------------------------
! END OPTION 1
! ----------------------------------------------------------------------
! ----------------------------------------------------------------------
! OPTION 2: EXPLICIT SOLUTION FOR FLERCHINGER EQ. i.e. CK=0
! IN KOREN ET AL., JGR, 1999, EQN 17
! APPLY PHYSICAL BOUNDS TO FLERCHINGER SOLUTION
! ----------------------------------------------------------------------
       IF (KCOUNT == 0) THEN
          PRINT *,'Flerchinger USEd in NEW version. Iterations=',NLOG
          FK = ( ( (hfus / (GRAV * ( - PSISAT)))*                    &
               ( (TKELV - TFRZ)/ TKELV))** ( -1/ BX))* SMCMAX
          IF (FK < 0.02) FK = 0.02
          FREE = MIN (FK, SMC)
! ----------------------------------------------------------------------
! END OPTION 2
! ----------------------------------------------------------------------
       END IF
    END IF
! ----------------------------------------------------------------------
  END SUBROUTINE FRH2O
! ----------------------------------------------------------------------
! ==================================================================================================
! **********************End of energy subroutines***********************
! ==================================================================================================
  SUBROUTINE WATER (VEGTYP ,NSNOW  ,NSOIL  ,IMELT  ,DT     ,UU     , & !in
                    VV     ,FCEV   ,FCTR   ,QPRECC ,QPRECL ,ELAI   , & !in
                    ESAI   ,SFCTMP ,QVAP   ,QDEW   ,ZSOIL  ,BTRANI , & !in
                    FICEOLD,PONDING,TG     ,IST    ,FVEG   ,ipoint , & !in
                    ISNOW  ,CANLIQ ,CANICE ,TV     ,SNOWH  ,SNEQV  , & !inout
                    SNICE  ,SNLIQ  ,STC    ,ZSNSO  ,SH2O   ,SMC    , & !inout
                    SICE   ,ZWT    ,WA     ,WT     ,DZSNSO ,WSLAKE , & !inout
                    CMC    ,ECAN   ,ETRAN  ,FWET   ,RUNSRF ,RUNSUB , & !out
                    QIN    ,QDIS   ,QSNOW  )                           !out
! ----------------------------------------------------------------------
! Code history:
! Initial code: Guo-Yue Niu, Oct. 2007
! ----------------------------------------------------------------------
  implicit none
! ----------------------------------------------------------------------
! input
  INTEGER,                         INTENT(IN)    :: ipoint  !
  INTEGER,                         INTENT(IN)    :: VEGTYP  !vegetation type
  INTEGER,                         INTENT(IN)    :: NSNOW   !maximum no. of snow layers
  INTEGER                        , INTENT(IN)    :: IST     !surface type 1-soil; 2-lake
  INTEGER,                         INTENT(IN)    :: NSOIL   !no. of soil layers
```

```
  INTEGER, DIMENSION(-NSNOW+1:0) , INTENT(IN)     :: IMELT   !melting state index [1-melt; 2-freeze]
  REAL,                            INTENT(IN)     :: DT      !main time step (s)
  REAL,                            INTENT(IN)     :: UU      !u-direction wind speed [m/s]
  REAL,                            INTENT(IN)     :: VV      !v-direction wind speed [m/s]
  REAL,                            INTENT(IN)     :: FCEV    !canopy evaporation (w/m2) [+ to atm ]
  REAL,                            INTENT(IN)     :: FCTR    !transpiration (w/m2) [+ to atm]
  REAL,                            INTENT(IN)     :: QPRECC  !convective precipitation (mm/s)
  REAL,                            INTENT(IN)     :: QPRECL  !large-scale precipitation (mm/s)
  REAL,                            INTENT(IN)     :: ELAI    !leaf area index, after burying by snow
  REAL,                            INTENT(IN)     :: ESAI    !stem area index, after burying by snow
  REAL,                            INTENT(IN)     :: SFCTMP  !surface air temperature [k]
  REAL,                            INTENT(IN)     :: QVAP    !soil surface evaporation rate[mm/s]
  REAL,                            INTENT(IN)     :: QDEW    !soil surface dew rate[mm/s]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)     :: ZSOIL   !depth of layer-bottom from soil surface
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)     :: BTRANI  !soil water stress factor (0 to 1)
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(IN)     :: FICEOLD !ice fraction at last timestep
  REAL                           , INTENT(IN)     :: PONDING ![mm]
  REAL                           , INTENT(IN)     :: TG      !ground temperature (k)
  REAL                           , INTENT(IN)     :: FVEG    !greeness vegetation fraction (-)

! input/output
  INTEGER,                         INTENT(INOUT)  :: ISNOW   !actual no. of snow layers
  REAL,                            INTENT(INOUT)  :: CANLIQ  !intercepted liquid water (mm)
  REAL,                            INTENT(INOUT)  :: CANICE  !intercepted ice mass (mm)
  REAL,                            INTENT(INOUT)  :: TV      !vegetation temperature (k)
  REAL,                            INTENT(INOUT)  :: SNOWH   !snow height [m]
  REAL,                            INTENT(INOUT)  :: SNEQV   !snow water eqv. [mm]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT)  :: SNICE   !snow layer ice [mm]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT)  :: SNLIQ   !snow layer liquid water [mm]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT)  :: STC     !snow/soil layer temperature [k]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT)  :: ZSNSO   !depth of snow/soil layer-bottom
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT)  :: DZSNSO  !snow/soil layer thickness [m]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT)  :: SH2O    !soil liquid water content [m3/m3]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT)  :: SICE    !soil ice content [m3/m3]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT)  :: SMC     !total soil water content [m3/m3]
  REAL,                            INTENT(INOUT)  :: ZWT     !the depth to water table [m]
  REAL,                            INTENT(INOUT)  :: WA      !water storage in aquifer [mm]
  REAL,                            INTENT(INOUT)  :: WT      !water storage in aquifer
                                                             !+ stuarated soil [mm]
  REAL,                            INTENT(INOUT)  :: WSLAKE  !water storage in lake (can be -) (mm)

! output
  REAL,                            INTENT(OUT)    :: CMC     !intercepted water per ground area (mm)
  REAL,                            INTENT(OUT)    :: ECAN    !evap of intercepted water (mm/s) [+]
  REAL,                            INTENT(OUT)    :: ETRAN   !transpiration rate (mm/s) [+]
  REAL,                            INTENT(OUT)    :: FWET    !wetted/snowed fraction of canopy (-)
  REAL,                            INTENT(OUT)    :: RUNSRF  !surface runoff [mm/s]
  REAL,                            INTENT(OUT)    :: RUNSUB  !baseflow (sturation excess) [mm/s]
  REAL,                            INTENT(OUT)    :: QIN     !groundwater recharge [mm/s]
  REAL,                            INTENT(OUT)    :: QDIS    !groundwater discharge [mm/s]
  REAL,                            INTENT(OUT)    :: QSNOW   !snow at ground srf (mm/s) [+]

! local
  INTEGER                                         :: IZ
  REAL                                            :: QINSUR  !water input on soil surface [m/s]
  REAL                                            :: QRAIN   !rain at ground srf (mm) [+]
  REAL                                            :: QSNBOT  !melting water out of snow bottom [mm/s]
  REAL                                            :: QSEVA   !soil surface evap rate [mm/s]
  REAL                                            :: QSDEW   !soil surface dew rate [mm/s]
  REAL                                            :: QSNFRO  !snow surface frost rate[mm/s]
  REAL                                            :: QSNSUB  !snow surface sublimation rate [mm/s]
  REAL                                            :: SNOWHIN !snow depth increasing rate (m/s)
  REAL, DIMENSION(       1:NSOIL)                 :: ETRANI  !transpiration rate (mm/s) [+]
  REAL, DIMENSION(       1:NSOIL)                 :: WCND   !hydraulic conductivity (m/s)
  REAL                                            :: QDRAIN  !soil-bottom free drainage [mm/s]
  REAL                                            :: SNOFLOW !glacier flow [mm/s]
  REAL                                            :: FCRMAX !maximum of FCR (-)

  REAL, PARAMETER ::  WSLMAX = 5000.      !maximum lake water storage (mm)
```

```
! -------------------------------------------------------------------------
! initialize

   ETRANI(1:NSOIL) = 0.
   SNOFLOW         = 0.
   RUNSUB          = 0.
   QINSUR          = 0.

! canopy-intercepted snowfall/rainfall, drips, and throughfall

   CALL CANWATER (VEGTYP ,DT     ,SFCTMP ,UU     ,VV     , & !in
                  FCEV   ,FCTR   ,QPRECC ,QPRECL ,ELAI   , & !in
                  ESAI   ,IST    ,TG     ,FVEG   ,ipoint , & !in
                  CANLIQ ,CANICE ,TV     ,                 & !inout
                  CMC    ,ECAN   ,ETRAN  ,QRAIN  ,QSNOW  , & !out
                  SNOWHIN,FWET   )                           !out

! sublimation, frost, evaporation, and dew

     QSNSUB = 0.
     IF (SNEQV > 0.) THEN
       QSNSUB = MIN(QVAP, SNEQV/DT)
     ENDIF
     QSEVA = QVAP-QSNSUB

     QSNFRO = 0.
     IF (SNEQV > 0.) THEN
        QSNFRO = QDEW
     ENDIF
     QSDEW = QDEW - QSNFRO

   CALL SNOWWATER (NSNOW  ,NSOIL  ,IMELT  ,DT     ,ZSOIL  , & !in
                   SFCTMP ,SNOWHIN,QSNOW  ,QSNFRO ,QSNSUB , & !in
                   QRAIN  ,FICEOLD,ipoint ,                 & !in
                   ISNOW  ,SNOWH  ,SNEQV  ,SNICE  ,SNLIQ  , & !inout
                   SH2O   ,SICE   ,STC    ,ZSNSO  ,DZSNSO , & !inout
                   QSNBOT ,SNOFLOW)                           !out

! convert units (mm/s -> m/s)

    !PONDING: melting water from snow when there is no layer
    QINSUR = PONDING/DT * 0.001

    IF(ISNOW == 0) THEN
       QINSUR = QINSUR+(QSNBOT + QSDEW + QRAIN) * 0.001
    ELSE
       QINSUR = QINSUR+(QSNBOT + QSDEW) * 0.001
    ENDIF

    QSEVA  = QSEVA * 0.001

    DO IZ = 1, NROOT
       ETRANI(IZ) = ETRAN * BTRANI(IZ) * 0.001
    ENDDO

! lake/soil water balances

    IF (IST == 2) THEN                                    ! lake
       RUNSRF = 0.
       IF(WSLAKE >= WSLMAX) RUNSRF = QINSUR*1000.         !mm/s
       WSLAKE = WSLAKE + (QINSUR-QSEVA)*1000.*DT -RUNSRF*DT !mm
    ELSE                                                  ! soil
       CALL      SOILWATER (NSOIL  ,NSNOW  ,DT     ,ZSOIL  ,DZSNSO , & !in
                            QINSUR ,QSEVA  ,ETRANI ,SICE   ,ipoint , & !in
                            SH2O   ,SMC    ,ZWT    ,                 & !inout
                            RUNSRF ,QDRAIN ,RUNSUB ,WCND   ,FCRMAX )   !out

       IF(OPT_RUN == 1) THEN
          CALL GROUNDWATER (NSNOW  ,NSOIL  ,DT     ,SICE   ,ZSOIL  , & !in
```

is anyone doing this routine?

```
                           STC     ,WCND    ,FCRMAX ,ipoint ,           & !in
                           SH2O    ,ZWT     ,WA      ,WT      ,           & !inout
                           QIN     ,QDIS    )                               !out
          RUNSUB       = QDIS          !mm/s
       END IF

       IF(OPT_RUN == 3 .or. OPT_RUN == 4) THEN
          RUNSUB       = RUNSUB + QDRAIN        !mm/s
       END IF

       DO IZ = 1,NSOIL
           SMC(IZ) = SH2O(IZ) + SICE(IZ)
       ENDDO
    ENDIF

    RUNSUB       = RUNSUB + SNOFLOW         !mm/s

  END SUBROUTINE WATER
! ====================================================================================================
  SUBROUTINE CANWATER (VEGTYP ,DT      ,SFCTMP ,UU      ,VV      ,  & !in
                       FCEV    ,FCTR    ,QPRECC ,QPRECL ,ELAI    ,  & !in
                       ESAI    ,IST     ,TG      ,FVEG    ,ipoint ,  & !in
                       CANLIQ ,CANICE ,TV      ,                    & !inout
                       CMC     ,ECAN    ,ETRAN  ,QRAIN   ,QSNOW   ,  & !out
                       SNOWHIN,FWET    )                               !out

! ------------------------ code history ------------------------------
! canopy hydrology
! --------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------
  IMPLICIT NONE
! ------------------------ input/output variables --------------------
! input
  INTEGER,INTENT(IN)   :: ipoint
  INTEGER,INTENT(IN)   :: VEGTYP  !vegetation type
  REAL,    INTENT(IN)   :: DT      !main time step (s)
  REAL,    INTENT(IN)   :: SFCTMP  !air temperature (k)
  REAL,    INTENT(IN)   :: UU      !u-direction wind speed [m/s]
  REAL,    INTENT(IN)   :: VV      !v-direction wind speed [m/s]
  REAL,    INTENT(IN)   :: FCEV    !canopy evaporation (w/m2) [+ = to atm]
  REAL,    INTENT(IN)   :: FCTR    !transpiration (w/m2) [+ = to atm]
  REAL,    INTENT(IN)   :: QPRECC  !convective precipitation (mm/s)
  REAL,    INTENT(IN)   :: QPRECL  !large-scale precipitation (mm/s)
  REAL,    INTENT(IN)   :: ELAI    !leaf area index, after burying by snow
  REAL,    INTENT(IN)   :: ESAI    !stem area index, after burying by snow
  INTEGER,INTENT(IN)   :: IST     !surface type 1-soil; 2-lake
  REAL,    INTENT(IN)   :: TG      !ground temperature (k)
  REAL,    INTENT(IN)   :: FVEG    !greeness vegetation fraction (-)

! input & output
  REAL, INTENT(INOUT) :: CANLIQ  !intercepted liquid water (mm)
  REAL, INTENT(INOUT) :: CANICE  !intercepted ice mass (mm)
  REAL, INTENT(INOUT) :: TV      !vegetation temperature (k)

! output
  REAL, INTENT(OUT)   :: CMC     !intercepted water (mm)
  REAL, INTENT(OUT)   :: ECAN    !evaporation of intercepted water (mm/s) [+]
  REAL, INTENT(OUT)   :: ETRAN   !transpiration rate (mm/s) [+]
  REAL, INTENT(OUT)   :: QRAIN   !rain at ground srf (mm/s) [+]
  REAL, INTENT(OUT)   :: QSNOW   !snow at ground srf (mm/s) [+]
  REAL, INTENT(OUT)   :: SNOWHIN !snow depth increasing rate (m/s)
  REAL, INTENT(OUT)   :: FWET    !wetted or snowed fraction of the canopy (-)
! --------------------------------------------------------------------

! ------------------------ local variables ---------------------------
  REAL                 :: MAXSNO  !canopy capacity for snow interception (mm)
  REAL                 :: MAXLIQ  !canopy capacity for rain interception (mm)
  REAL                 :: FP      !fraction of the gridcell that receives precipitation
```

```
  REAL                 :: FPICE   !snow fraction in precipitation
  REAL                 :: BDFALL  !bulk density of snowfall (kg/m3)
  REAL                 :: QINTR   !interception rate for rain (mm/s)
  REAL                 :: QDRIPR  !drip rate for rain (mm/s)
  REAL                 :: QTHROR  !throughfall for rain (mm/s)
  REAL                 :: QINTS   !interception (loading) rate for snowfall (mm/s)
  REAL                 :: QDRIPS  !drip (unloading) rate for intercepted snow (mm/s)
  REAL                 :: QTHROS  !throughfall of snowfall (mm/s)
  REAL                 :: QEVAC   !evaporation rate (mm/s)
  REAL                 :: QDEWC   !dew rate (mm/s)
  REAL                 :: QFROC   !frost rate (mm/s)
  REAL                 :: QSUBC   !sublimation rate (mm/s)
  REAL                 :: FT      !temperature factor for unloading rate
  REAL                 :: FV      !wind factor for unloading rate
  REAL                 :: QMELTC  !melting rate of canopy snow (mm/s)
  REAL                 :: QFRZC   !refreezing rate of canopy liquid water (mm/s)
  REAL                 :: RAIN    !rainfall (mm/s)
  REAL                 :: SNOW    !snowfall (mm/s)
  REAL                 :: CANMAS  !total canopy mass (kg/m2)
! --------------------------------------------------------------------
! initialization

      FP      = 0.0
      RAIN    = 0.0
      SNOW    = 0.0
      QINTR   = 0.
      QDRIPR  = 0.
      QTHROR  = 0.
      QINTR   = 0.
      QINTS   = 0.
      QDRIPS  = 0.0
      QTHROS  = 0.
      QRAIN   = 0.0
      QSNOW   = 0.0
      SNOWHIN = 0.0
      ECAN    = 0.0
! --------------------------------------------------------------------
! partition precipitation into rain and snow.

! Jordan (1991)

     IF(OPT_SNF == 1) THEN
       IF(SFCTMP > TFRZ+2.5)THEN
           FPICE = 0.
       ELSE
         IF(SFCTMP <= TFRZ+0.5)THEN
           FPICE = 1.0
         ELSE IF(SFCTMP <= TFRZ+2.)THEN
           FPICE = 1.-(-54.632 + 0.2*SFCTMP)
         ELSE
           FPICE = 0.6
         ENDIF
       ENDIF
     ENDIF

     IF(OPT_SNF == 2) THEN
       IF(SFCTMP >= TFRZ+2.2) THEN
           FPICE = 0.
       ELSE
           FPICE = 1.0
       ENDIF
     ENDIF

     IF(OPT_SNF == 3) THEN
       IF(SFCTMP >= TFRZ) THEN
           FPICE = 0.
       ELSE
           FPICE = 1.0
       ENDIF
```

```
     ENDIF

! Hedstrom NR and JW Pomeroy (1998), Hydrol. Processes, 12, 1611-1625
! fresh snow density

     BDFALL = MAX(120.,67.92+51.25*EXP((SFCTMP-TFRZ)/2.59))

     RAIN   = (QPRECC + QPRECL) * (1.-FPICE)
     SNOW   = (QPRECC + QPRECL) * FPICE

! fractional area that receives precipitation (see, Niu et al. 2005)

    IF(QPRECC + QPRECL > 0.) &
       FP = (QPRECC + QPRECL) / (10.*QPRECC + QPRECL)

! ---------------------------- liquid water ------------------------------
! maximum canopy water

      MAXLIQ =  CH2OP(VEGTYP) * (ELAI+ ESAI)

! average interception and throughfall

      IF((ELAI+ ESAI).GT.0.) THEN
         QINTR  = FVEG * RAIN * FP  ! interception capability
         QINTR  = MIN(QINTR, (MAXLIQ - CANLIQ)/DT * (1.-EXP(-RAIN*DT/MAXLIQ)) )
         QINTR  = MAX(QINTR, 0.)
         QDRIPR = FVEG * RAIN - QINTR
         QTHROR = (1.-FVEG) * RAIN
      ELSE
         QINTR  = 0.
         QDRIPR = 0.
         QTHROR = RAIN
      END IF

! evaporation, transpiration, and dew

      IF (TV .GT. TFRZ) THEN
        ETRAN = MAX( FCTR/HVAP, 0. )
        QEVAC = MAX( FCEV/HVAP, 0. )
        QDEWC = ABS( MIN( FCEV/HVAP, 0. ) )
        QSUBC = 0.
        QFROC = 0.
      ELSE
        ETRAN = MAX( FCTR/HSUB, 0. )
        QEVAC = 0.
        QDEWC = 0.
        QSUBC = MAX( FCEV/HSUB, 0. )
        QFROC = ABS( MIN( FCEV/HSUB, 0. ) )
      ENDIF

! canopy water balance. for convenience allow dew to bring CANLIQ above
! maxh2o or else would have to re-adjust drip

       QEVAC = MIN(CANLIQ/DT,QEVAC)
       CANLIQ=MAX(0.,CANLIQ+(QINTR+QDEWC-QEVAC)*DT)
       IF(CANLIQ <= 1.E-03) CANLIQ = 0.0

! ---------------------------- canopy ice ------------------------------
! for canopy ice

      MAXSNO = 6.6*(0.27+46./BDFALL) * (ELAI+ ESAI)

      IF((ELAI+ ESAI).GT.0.) THEN
         QINTS = FVEG * SNOW * FP
         QINTS = MIN(QINTS, (MAXSNO - CANICE)/DT * (1.-EXP(-SNOW*DT/MAXSNO)) )
         QINTS = MAX(QINTS, 0.)
         FT = MAX(0.0,(TV - 270.15) / 1.87E5)
         FV = SQRT(UU*UU + VV*VV) / 1.56E5
         QDRIPS = MAX(0.,CANICE/DT) * (FV+FT)
```

```
         QTHROS = (1.0-FVEG) * SNOW + (FVEG * SNOW - QINTS)
      ELSE
         QINTS  = 0.
         QDRIPS = 0.
         QTHROS = SNOW
      ENDIF

      QSUBC = MIN(CANICE/DT,QSUBC)
      CANICE= MAX(0.,CANICE+(QINTS-QDRIPS)*DT + (QFROC-QSUBC)*DT)
      IF(CANICE.LE.1.E-3) CANICE = 0.

! wetted fraction of canopy

      IF(CANICE.GT.0.) THEN
           FWET = MAX(0.,CANICE) / MAX(MAXSNO,1.E-06)
      ELSE
           FWET = MAX(0.,CANLIQ) / MAX(MAXLIQ,1.E-06)
      ENDIF
      FWET = MIN(FWET, 1.) ** 0.667

! phase change

      QMELTC = 0.
      QFRZC = 0.

      IF(CANICE.GT.1.E-3.AND.TV.GT.TFRZ) THEN
         QMELTC = MIN(CANICE/DT,(TV-TFRZ)*CICE*CANICE/DENICE/(DT*HFUS))
         CANICE = MAX(0.,CANICE - QMELTC*DT)
         CANLIQ = MAX(0.,CANLIQ + QMELTC*DT)
         TV     = FWET*TFRZ + (1.-FWET)*TV
      ENDIF

      IF(CANLIQ.GT.1.E-3.AND.TV.LT.TFRZ) THEN
         QFRZC  = MIN(CANLIQ/DT,(TFRZ-TV)*CWAT*CANLIQ/DENH2O/(DT*HFUS))
         CANLIQ = MAX(0.,CANLIQ - QFRZC*DT)
         CANICE = MAX(0.,CANICE + QFRZC*DT)
         TV     = FWET*TFRZ + (1.-FWET)*TV
      ENDIF

! total canopy water

      CMC = CANLIQ + CANICE

! total canopy evaporation

      ECAN = QEVAC + QSUBC - QDEWC - QFROC

! rain or snow on the ground

      QRAIN   = QDRIPR + QTHROR
      QSNOW   = QDRIPS + QTHROS
      SNOWHIN = QSNOW/BDFALL


      IF (IST == 2 .AND. TG > TFRZ) THEN
         QSNOW   = 0.
         SNOWHIN = 0.
      END IF

  END SUBROUTINE CANWATER
! ==================================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE SNOWWATER (NSNOW  ,NSOIL  ,IMELT  ,DT     ,ZSOIL  , & !in
                        SFCTMP ,SNOWHIN,QSNOW  ,QSNFRO ,QSNSUB , & !in
                        QRAIN  ,FICEOLD,ipoint ,               & !in
                        ISNOW  ,SNOWH  ,SNEQV  ,SNICE  ,SNLIQ  , & !inout
                        SH2O   ,SICE   ,STC    ,ZSNSO  ,DZSNSO , & !inout
                        QSNBOT ,SNOFLOW)                         !out
! ----------------------------------------------------------------------
```

```
  IMPLICIT NONE
! --------------------------------------------------------------------
! input
  INTEGER,                         INTENT(IN)    :: ipoint
  INTEGER,                         INTENT(IN)    :: NSNOW  !maximum no. of snow layers
  INTEGER,                         INTENT(IN)    :: NSOIL  !no. of soil layers
  INTEGER, DIMENSION(-NSNOW+1:0) , INTENT(IN)    :: IMELT  !melting state index [0-no melt;1-melt]
  REAL,                            INTENT(IN)    :: DT     !time step (s)
  REAL, DIMENSION(       1:NSOIL), INTENT(IN)    :: ZSOIL  !depth of layer-bottom from soil surface
  REAL,                            INTENT(IN)    :: SFCTMP !surface air temperature [k]
  REAL,                            INTENT(IN)    :: SNOWHIN!snow depth increasing rate (m/s)
  REAL,                            INTENT(IN)    :: QSNOW  !snow at ground srf (mm/s) [+]
  REAL,                            INTENT(IN)    :: QSNFRO !snow surface frost rate[mm/s]
  REAL,                            INTENT(IN)    :: QSNSUB !snow surface sublimation rate[mm/s]
  REAL,                            INTENT(IN)    :: QRAIN  !snow surface rain rate[mm/s]
  REAL, DIMENSION(-NSNOW+1:0)    , INTENT(IN)    :: FICEOLD!ice fraction at last timestep

! input & output
  INTEGER,                         INTENT(INOUT) :: ISNOW  !actual no. of snow layers
  REAL,                            INTENT(INOUT) :: SNOWH  !snow height [m]
  REAL,                            INTENT(INOUT) :: SNEQV  !snow water eqv. [mm]
  REAL, DIMENSION(-NSNOW+1:     0), INTENT(INOUT) :: SNICE  !snow layer ice [mm]
  REAL, DIMENSION(-NSNOW+1:     0), INTENT(INOUT) :: SNLIQ  !snow layer liquid water [mm]
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SH2O   !soil liquid moisture (m3/m3)
  REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SICE   !soil ice moisture (m3/m3)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC    !snow layer temperature [k]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: ZSNSO  !depth of snow/soil layer-bottom
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: DZSNSO !snow/soil layer thickness [m]

! output
  REAL,                              INTENT(OUT) :: QSNBOT !melting water out of snow bottom [mm/s]
  REAL,                              INTENT(OUT) :: SNOFLOW!glacier flow [mm]

! local
  INTEGER :: IZ,i
  REAL    :: BDSNOW  !bulk density of snow (kg/m3)
! --------------------------------------------------------------------
   SNOFLOW = 0.0

   CALL SNOWFALL (NSOIL  ,NSNOW  ,DT     ,QSNOW  ,SNOWHIN, & !in
                  SFCTMP ,ipoint ,                         & !in
                  ISNOW  ,SNOWH  ,DZSNSO ,STC    ,SNICE  , & !inout
                  SNLIQ  ,SNEQV  )                           !inout

   if(isnow < 0) then        !when more than one layer
   CALL  COMPACT (NSNOW  ,NSOIL  ,DT     ,STC    ,SNICE  , & !in
                  SNLIQ  ,ZSOIL  ,IMELT  ,FICEOLD,ipoint , & !in
                  ISNOW  ,DZSNSO ,ZSNSO  )                   !inout

   CALL  COMBINE (NSNOW  ,NSOIL  ,ipoint ,                 & !in
                  ISNOW  ,SH2O   ,STC    ,SNICE  ,SNLIQ  , & !inout
                  DZSNSO ,SICE   ,SNOWH  ,SNEQV  )           !inout

   CALL   DIVIDE (NSNOW  ,NSOIL  ,                         & !in
                  ISNOW  ,STC    ,SNICE  ,SNLIQ  ,DZSNSO )   !inout
   end if

!set empty snow layers to zero

   do iz = -nsnow+1, isnow
        snice(iz) = 0.
        snliq(iz) = 0.
        stc(iz)   = 0.
        dzsnso(iz)= 0.
        zsnso(iz) = 0.
   enddo

   CALL  SNOWH2O (NSNOW  ,NSOIL  ,DT     ,QSNFRO ,QSNSUB , & !in
                  QRAIN  ,ipoint ,                                & !in
```

```
                  ISNOW   ,DZSNSO ,SNOWH   ,SNEQV   ,SNICE   ,  & !inout
                  SNLIQ   ,SH2O   ,SICE    ,STC     ,           & !inout
                  QSNBOT )                                          !out

!to obtain equilibrium state of snow in glacier region

   IF(SNEQV > 2000.) THEN   ! 2000 mm -> maximum water depth
      BDSNOW      = SNICE(0) / DZSNSO(0)
      SNOFLOW     = (SNEQV - 2000.)
      SNICE(0)    = SNICE(0)  - SNOFLOW
      DZSNSO(0)   = DZSNSO(0) - SNOFLOW/BDSNOW
      SNOFLOW     = SNOFLOW / DT
   END IF

! sum up snow mass for layered snow

   IF(ISNOW /= 0) THEN
       SNEQV = 0.
       DO IZ = ISNOW+1,0
             SNEQV = SNEQV + SNICE(IZ) + SNLIQ(IZ)
       ENDDO
   END IF

! Reset ZSNSO and layer thinkness DZSNSO

   DO IZ = ISNOW+1, 0
        DZSNSO(IZ) = -DZSNSO(IZ)
   END DO

   DZSNSO(1) = ZSOIL(1)
   DO IZ = 2,NSOIL
        DZSNSO(IZ) = (ZSOIL(IZ) - ZSOIL(IZ-1))
   END DO

   ZSNSO(ISNOW+1) = DZSNSO(ISNOW+1)
   DO IZ = ISNOW+2 ,NSOIL
       ZSNSO(IZ) = ZSNSO(IZ-1) + DZSNSO(IZ)
   ENDDO

   DO IZ = ISNOW+1 ,NSOIL
       DZSNSO(IZ) = -DZSNSO(IZ)
   END DO

  END SUBROUTINE SNOWWATER
! ==================================================================================================
  SUBROUTINE SNOWFALL (NSOIL  ,NSNOW   ,DT     ,QSNOW   ,SNOWHIN ,  & !in
                       SFCTMP ,ipoint ,                             & !in
                       ISNOW  ,SNOWH   ,DZSNSO ,STC     ,SNICE   ,  & !inout
                       SNLIQ  ,SNEQV   )                              !inout
! ----------------------------------------------------------------------
! snow depth and density to account for the new snowfall.
! new values of snow depth & density returned.
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input

  INTEGER,                            INTENT(IN) :: ipoint !
  INTEGER,                            INTENT(IN) :: NSOIL  !no. of soil layers
  INTEGER,                            INTENT(IN) :: NSNOW  !maximum no. of snow layers
  REAL,                               INTENT(IN) :: DT     !main time step (s)
  REAL,                               INTENT(IN) :: QSNOW  !snow at ground srf (mm/s) [+]
  REAL,                               INTENT(IN) :: SNOWHIN!snow depth increasing rate (m/s)
  REAL,                               INTENT(IN) :: SFCTMP !surface air temperature [k]

! input and output

  INTEGER,                         INTENT(INOUT) :: ISNOW  !actual no. of snow layers
  REAL,                            INTENT(INOUT) :: SNOWH  !snow depth [m]
```

```
  REAL,                              INTENT(INOUT) :: SNEQV  !swow water equivalent [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: DZSNSO !thickness of snow/soil layers (m)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC    !snow layer temperature [k]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNICE  !snow layer ice [mm]
  REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNLIQ  !snow layer liquid water [mm]

! local

  INTEGER :: NEWNODE            ! 0-no new layers, 1-creating new layers
! ----------------------------------------------------------------------
    NEWNODE  = 0

! shallow snow / no layer

    IF(ISNOW == 0 .and. QSNOW > 0.)  THEN
      SNOWH = SNOWH + SNOWHIN * DT
      SNEQV = SNEQV + QSNOW * DT
    END IF

! creating a new layer

    IF(ISNOW == 0  .AND. QSNOW>0. .AND. SNOWH >= 0.05) THEN
      ISNOW    = -1
      NEWNODE  =  1
      DZSNSO(0)= SNOWH
      SNOWH    = 0.
      STC(0)   = MIN(273.16, SFCTMP)   ! temporary setup
      SNICE(0) = SNEQV
      SNLIQ(0) = 0.
    END IF

! snow with layers

    IF(ISNOW <  0 .AND. NEWNODE == 0 .AND. QSNOW > 0.) then
         SNICE(ISNOW+1)  = SNICE(ISNOW+1)   + QSNOW   * DT
         DZSNSO(ISNOW+1) = DZSNSO(ISNOW+1)  + SNOWHIN * DT
    ENDIF

! ----------------------------------------------------------------------
  END SUBROUTINE SNOWFALL
! ==================================================================================================
  SUBROUTINE COMBINE (NSNOW  ,NSOIL  ,ipoint ,                 & !in
                      ISNOW  ,SH2O   ,STC    ,SNICE  ,SNLIQ  , & !inout
                      DZSNSO ,SICE   ,SNOWH  ,SNEQV  )           !inout
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input

  INTEGER,                           INTENT(IN)    :: ipoint
    INTEGER, INTENT(IN)      :: NSNOW                          !maximum no. of snow layers
    INTEGER, INTENT(IN)      :: NSOIL                          !no. of soil layers

! input and output

    INTEGER,                           INTENT(INOUT) :: ISNOW !actual no. of snow layers
    REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SH2O  !soil liquid moisture (m3/m3)
    REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SICE  !soil ice moisture (m3/m3)
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC   !snow layer temperature [k]
    REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNICE !snow layer ice [mm]
    REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNLIQ !snow layer liquid water [mm]
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: DZSNSO!snow layer depth [m]
    REAL,                              INTENT(INOUT) :: sneqv !snow water equivalent [m]
    REAL,                              INTENT(INOUT) :: snowh !snow depth [m]

! local variables:

    INTEGER :: I,J,K,L               ! node indices
    INTEGER :: ISNOW_OLD             ! number of top snow layer
```

```
    INTEGER :: MSSI                    ! node index
    INTEGER :: NEIBOR                  ! adjacent node selected for combination
    REAL    :: ZWICE                   ! total ice mass in snow
    REAL    :: ZWLIQ                   ! total liquid water in snow

    REAL    :: DZMIN(3)                ! minimum of top snow layer
    DATA DZMIN /0.045, 0.05, 0.2/
!-----------------------------------------------------------------------

       ISNOW_OLD = ISNOW

       DO J = ISNOW_OLD+1,0
          IF (SNICE(J) <= .1) THEN
             IF(J /= 0) THEN
                SNLIQ(J+1) = SNLIQ(J+1) + SNLIQ(J)
                SNICE(J+1) = SNICE(J+1) + SNICE(J)
             ELSE
                SH2O(1) = SH2O(1)+SNLIQ(J)/(DZSNSO(1)*1000.)
                SICE(1) = SICE(1)+SNICE(J)/(DZSNSO(1)*1000.)
             ENDIF

             ! shift all elements above this down by one.
             IF (J > ISNOW+1 .AND. ISNOW < -1) THEN
                DO I = J, ISNOW+2, -1
                   STC(I)   = STC(I-1)
                   SNLIQ(I) = SNLIQ(I-1)
                   SNICE(I) = SNICE(I-1)
                   DZSNSO(I)= DZSNSO(I-1)
                END DO
             END IF
             ISNOW = ISNOW + 1
          END IF
       END DO

! to conserve water in case of too large surface sublimation

       IF(SICE(1) < 0.) THEN
          SH2O(1) = SH2O(1) + SICE(1)
          SICE(1) = 0.
       END IF

       SNEQV  = 0.
       SNOWH  = 0.
       ZWICE  = 0.
       ZWLIQ  = 0.

       DO J = ISNOW+1,0
             SNEQV = SNEQV + SNICE(J) + SNLIQ(J)
             SNOWH = SNOWH + DZSNSO(J)
             ZWICE = ZWICE + SNICE(J)
             ZWLIQ = ZWLIQ + SNLIQ(J)
       END DO

! check the snow depth - all snow gone
! the liquid water assumes ponding on soil surface.

       IF (SNOWH < 0.05 ) THEN
          ISNOW  = 0
          SNEQV = ZWICE
          SH2O(1) = SH2O(1) + ZWLIQ / (DZSNSO(1) * 1000.)
          IF(SNEQV <= 0.) SNOWH = 0.
       END IF

! check the snow depth - snow layers combined

       IF (ISNOW < -1) THEN

          ISNOW_OLD = ISNOW
          MSSI     = 1
```

```
          DO I = ISNOW_OLD+1,0
             IF (DZSNSO(I) < DZMIN(MSSI)) THEN

                IF (I == ISNOW+1) THEN
                   NEIBOR = I + 1
                ELSE IF (I == 0) THEN
                   NEIBOR = I - 1
                ELSE
                   NEIBOR = I + 1
                   IF ((DZSNSO(I-1)+DZSNSO(I)) < (DZSNSO(I+1)+DZSNSO(I))) NEIBOR = I-1
                END IF

                ! Node l and j are combined and stored as node j.
                IF (NEIBOR > I) THEN
                   J = NEIBOR
                   L = I
                ELSE
                   J = I
                   L = NEIBOR
                END IF

                CALL COMBO (DZSNSO(J), SNLIQ(J), SNICE(J), &
                   STC(J), DZSNSO(L), SNLIQ(L), SNICE(L), STC(L) )

                ! Now shift all elements above this down one.
                IF (J-1 > ISNOW+1) THEN
                   DO K = J-1, ISNOW+2, -1
                      STC(K)   = STC(K-1)
                      SNICE(K) = SNICE(K-1)
                      SNLIQ(K) = SNLIQ(K-1)
                      DZSNSO(K) = DZSNSO(K-1)
                   END DO
                END IF

                ! Decrease the number of snow layers
                ISNOW = ISNOW + 1
                IF (ISNOW >= -1) EXIT
             ELSE

                ! The layer thickness is greater than the prescribed minimum value
                MSSI = MSSI + 1

             END IF
          END DO

       END IF

  END SUBROUTINE COMBINE
! ==================================================================================================
  SUBROUTINE DIVIDE (NSNOW  ,NSOIL  ,                          & !in
                     ISNOW  ,STC    ,SNICE  ,SNLIQ  ,DZSNSO  )  !inout
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input

    INTEGER, INTENT(IN)                            :: NSNOW !maximum no. of snow layers [ =3]
    INTEGER, INTENT(IN)                            :: NSOIL !no. of soil layers [ =4]

! input and output

    INTEGER                        , INTENT(INOUT) :: ISNOW !actual no. of snow layers
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC   !snow layer temperature [k]
    REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNICE !snow layer ice [mm]
    REAL, DIMENSION(-NSNOW+1:    0), INTENT(INOUT) :: SNLIQ !snow layer liquid water [mm]
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: DZSNSO!snow layer depth [m]

! local variables:
```

```
    INTEGER                                        :: J     !indices
    INTEGER                                        :: MSNO  !number of layer (top) to MSNO (bot)
    REAL                                           :: DRR   !thickness of the combined [m]
    REAL, DIMENSION(       1:NSNOW)                :: DZ    !snow layer thickness [m]
    REAL, DIMENSION(       1:NSNOW)                :: SWICE !partial volume of ice [m3/m3]
    REAL, DIMENSION(       1:NSNOW)                :: SWLIQ !partial volume of liquid water [m3/m3]
    REAL, DIMENSION(       1:NSNOW)                :: TSNO  !node temperature [k]
    REAL                                           :: ZWICE !temporary
    REAL                                           :: ZWLIQ !temporary
    REAL                                           :: PROPOR!temporary
    REAL                                           :: DTDZ  !temporary
! ----------------------------------------------------------------------

    DO J = 1,NSNOW
          IF (J <= ABS(ISNOW)) THEN
             DZ(J)     = DZSNSO(J+ISNOW)
             SWICE(J)  = SNICE(J+ISNOW)
             SWLIQ(J)  = SNLIQ(J+ISNOW)
             TSNO(J)   = STC(J+ISNOW)
          END IF
    END DO

       MSNO = ABS(ISNOW)

       IF (MSNO == 1) THEN
          ! Specify a new snow layer
          IF (DZ(1) > 0.05) THEN
             MSNO = 2
             DZ(1)     = DZ(1)/2.
             SWICE(1)  = SWICE(1)/2.
             SWLIQ(1)  = SWLIQ(1)/2.
             DZ(2)     = DZ(1)
             SWICE(2)  = SWICE(1)
             SWLIQ(2)  = SWLIQ(1)
             TSNO(2)   = TSNO(1)
          END IF
       END IF

       IF (MSNO > 1) THEN
          IF (DZ(1) > 0.05) THEN
             DRR       = DZ(1) - 0.05
             PROPOR    = DRR/DZ(1)
             ZWICE     = PROPOR*SWICE(1)
             ZWLIQ     = PROPOR*SWLIQ(1)
             PROPOR    = 0.05/DZ(1)
             SWICE(1)  = PROPOR*SWICE(1)
             SWLIQ(1)  = PROPOR*SWLIQ(1)
             DZ(1)     = 0.05

             CALL COMBO (DZ(2), SWLIQ(2), SWICE(2), TSNO(2), DRR, &
                  ZWLIQ, ZWICE, TSNO(1))

             ! subdivide a new layer
             IF (MSNO <= 2 .AND. DZ(2) > 0.10) THEN
                MSNO = 3
                DTDZ = (TSNO(1) - TSNO(2))/((DZ(1)+DZ(2))/2.)
                DZ(2)     = DZ(2)/2.
                SWICE(2)  = SWICE(2)/2.
                SWLIQ(2)  = SWLIQ(2)/2.
                DZ(3)     = DZ(2)
                SWICE(3)  = SWICE(2)
                SWLIQ(3)  = SWLIQ(2)
                TSNO(3) = TSNO(2) - DTDZ*DZ(2)/2.
                IF (TSNO(3) >= TFRZ) THEN
                   TSNO(3)   = TSNO(2)
                ELSE
                   TSNO(2) = TSNO(2) + DTDZ*DZ(2)/2.
                ENDIF
```

```
            END IF
         END IF
      END IF

      IF (MSNO > 2) THEN
         IF (DZ(2) > 0.2) THEN
            DRR = DZ(2) - 0.2
            PROPOR   = DRR/DZ(2)
            ZWICE    = PROPOR*SWICE(2)
            ZWLIQ    = PROPOR*SWLIQ(2)
            PROPOR   = 0.2/DZ(2)
            SWICE(2) = PROPOR*SWICE(2)
            SWLIQ(2) = PROPOR*SWLIQ(2)
            DZ(2)    = 0.2
            CALL COMBO (DZ(3), SWLIQ(3), SWICE(3), TSNO(3), DRR, &
                 ZWLIQ, ZWICE, TSNO(2))
         END IF
      END IF

      ISNOW = -MSNO

    DO J = ISNOW+1,0
            DZSNSO(J) = DZ(J-ISNOW)
            SNICE(J) = SWICE(J-ISNOW)
            SNLIQ(J) = SWLIQ(J-ISNOW)
            STC(J)   = TSNO(J-ISNOW)
    END DO


!    DO J = ISNOW+1,NSOIL
!    WRITE(*,'(I5,7F10.3)') J, DZSNSO(J), SNICE(J), SNLIQ(J),STC(J)
!    END DO

  END SUBROUTINE DIVIDE
! ==================================================================================================

! ----------------------------------------------------------------------
  SUBROUTINE COMBO(DZ,  WLIQ,  WICE, T, DZ2, WLIQ2, WICE2, T2)
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------

! ----------------------------------------------------------------------s
! input

    REAL, INTENT(IN)    :: DZ2   !nodal thickness of 2 elements being combined [m]
    REAL, INTENT(IN)    :: WLIQ2 !liquid water of element 2 [kg/m2]
    REAL, INTENT(IN)    :: WICE2 !ice of element 2 [kg/m2]
    REAL, INTENT(IN)    :: T2    !nodal temperature of element 2 [k]
    REAL, INTENT(INOUT) :: DZ    !nodal thickness of 1 elements being combined [m]
    REAL, INTENT(INOUT) :: WLIQ  !liquid water of element 1
    REAL, INTENT(INOUT) :: WICE  !ice of element 1 [kg/m2]
    REAL, INTENT(INOUT) :: T     !node temperature of element 1 [k]

! local

    REAL                :: DZC   !total thickness of nodes 1 and 2 (DZC=DZ+DZ2).
    REAL                :: WLIQC !combined liquid water [kg/m2]
    REAL                :: WICEC !combined ice [kg/m2]
    REAL                :: TC    !combined node temperature [k]
    REAL                :: H     !enthalpy of element 1 [J/m2]
    REAL                :: H2    !enthalpy of element 2 [J/m2]
    REAL                :: HC    !temporary

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

    DZC = DZ+DZ2
    WICEC = (WICE+WICE2)
```

```
     WLIQC = (WLIQ+WLIQ2)
     H = (CICE*WICE+CWAT*WLIQ) * (T-TFRZ)+HFUS*WLIQ
     H2= (CICE*WICE2+CWAT*WLIQ2) * (T2-TFRZ)+HFUS*WLIQ2

     HC = H + H2
     IF(HC < 0.)THEN
        TC = TFRZ + HC/(CICE*WICEC + CWAT*WLIQC)
     ELSE IF (HC.LE.HFUS*WLIQC) THEN
        TC = TFRZ
     ELSE
        TC = TFRZ + (HC - HFUS*WLIQC) / (CICE*WICEC + CWAT*WLIQC)
     END IF

     DZ = DZC
     WICE = WICEC
     WLIQ = WLIQC
     T = TC

  END SUBROUTINE COMBO
! ==================================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE COMPACT (NSNOW  ,NSOIL  ,DT     ,STC    ,SNICE  , & !in
                      SNLIQ  ,ZSOIL  ,IMELT  ,FICEOLD,ipoint , & !in
                      ISNOW  ,DZSNSO ,ZSNSO )                    !inout
! ----------------------------------------------------------------------
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! input
   INTEGER,                          INTENT(IN)    :: ipoint !
   INTEGER,                          INTENT(IN)    :: NSOIL  !no. of soil layers [ =4]
   INTEGER,                          INTENT(IN)    :: NSNOW  !maximum no. of snow layers [ =3]
   INTEGER, DIMENSION(-NSNOW+1:0) ,  INTENT(IN)    :: IMELT  !melting state index [0-no melt;1-melt]
   REAL,                             INTENT(IN)    :: DT     !time step (sec)
   REAL, DIMENSION(-NSNOW+1:NSOIL),  INTENT(IN)    :: STC    !snow layer temperature [k]
   REAL, DIMENSION(-NSNOW+1:    0),  INTENT(IN)    :: SNICE  !snow layer ice [mm]
   REAL, DIMENSION(-NSNOW+1:    0),  INTENT(IN)    :: SNLIQ  !snow layer liquid water [mm]
   REAL, DIMENSION(       1:NSOIL),  INTENT(IN)    :: ZSOIL  !depth of layer-bottom from soil srf
   REAL, DIMENSION(-NSNOW+1:    0),  INTENT(IN)    :: FICEOLD!ice fraction at last timestep

! input and output
   INTEGER,                          INTENT(INOUT) :: ISNOW  ! actual no. of snow layers
   REAL, DIMENSION(-NSNOW+1:NSOIL),  INTENT(INOUT) :: DZSNSO ! snow layer thickness [m]
   REAL, DIMENSION(-NSNOW+1:NSOIL),  INTENT(INOUT) :: ZSNSO  ! depth of snow/soil layer-bottom

! local
   REAL, PARAMETER     :: C2 = 21.e-3   ![m3/kg] ! default 21.e-3
   REAL, PARAMETER     :: C3 = 2.5e-6   ![1/s]
   REAL, PARAMETER     :: C4 = 0.04     ![1/k]
   REAL, PARAMETER     :: C5 = 2.0      !
   REAL, PARAMETER     :: DM = 100.0    !upper Limit on destructive metamorphism compaction [kg/m3]
   REAL, PARAMETER     :: ETA0 = 0.8e+6 !viscosity coefficient [kg-s/m2]
                                        !according to Anderson, it is between 0.52e6~1.38e6
   REAL :: BURDEN !pressure of overlying snow [kg/m2]
   REAL :: DDZ1   !rate of settling of snow pack due to destructive metamorphism.
   REAL :: DDZ2   !rate of compaction of snow pack due to overburden.
   REAL :: DDZ3   !rate of compaction of snow pack due to melt [1/s]
   REAL :: DEXPF  !EXPF=exp(-c4*(273.15-STC)).
   REAL :: TD     !STC - TFRZ [K]
   REAL :: PDZDTC !nodal rate of change in fractional-thickness due to compaction [fraction/s]
   REAL :: VOID   !void (1 - SNICE - SNLIQ)
   REAL :: WX     !water mass (ice + liquid) [kg/m2]
   REAL :: BI     !partial density of ice [kg/m3]
   REAL, DIMENSION(-NSNOW+1:0) :: FICE  !fraction of ice at current time step

   INTEGER  :: J

! ----------------------------------------------------------------------
     BURDEN = 0.0
```

```
    DO J = ISNOW+1, 0

        WX      = SNICE(J) + SNLIQ(J)
        FICE(J) = SNICE(J) / WX
        VOID    = 1. - (SNICE(J)/DENICE + SNLIQ(J)/DENH2O) / DZSNSO(J)

        ! Allow compaction only for non-saturated node and higher ice lens node.
        IF (VOID > 0.001 .AND. SNICE(J) > 0.1) THEN
           BI = SNICE(J) / DZSNSO(J)
           TD = MAX(0.,TFRZ-STC(J))
           DEXPF = EXP(-C4*TD)

           ! Settling as a result of destructive metamorphism

           DDZ1 = -C3*DEXPF

           IF (BI > DM) DDZ1 = DDZ1*EXP(-46.0E-3*(BI-DM))

           ! Liquid water term

           IF (SNLIQ(J) > 0.01*DZSNSO(J)) DDZ1=DDZ1*C5

           ! Compaction due to overburden

           DDZ2 = -(BURDEN+0.5*WX)*EXP(-0.08*TD-C2*BI)/ETA0 ! 0.5*WX -> self-burden

           ! Compaction occurring during melt

           IF (IMELT(J) == 1) THEN
              DDZ3 = MAX(0.,(FICEOLD(J) - FICE(J))/MAX(1.E-6,FICEOLD(J)))
              DDZ3 = - DDZ3/DT           ! sometimes too large
           ELSE
              DDZ3 = 0.
           END IF

           ! Time rate of fractional change in DZ (units of s-1)

           PDZDTC = (DDZ1 + DDZ2 + DDZ3)*DT
           PDZDTC = MAX(-0.5,PDZDTC)

           ! The change in DZ due to compaction

           DZSNSO(J) = DZSNSO(J)*(1.+PDZDTC)
        END IF

        ! Pressure of overlying snow

        BURDEN = BURDEN + WX

    END DO

  END SUBROUTINE COMPACT
! ==================================================================================================
  SUBROUTINE SNOWH2O (NSNOW  ,NSOIL  ,DT     ,QSNFRO ,QSNSUB , & !in
                      QRAIN  ,ipoint ,                         & !in
                      ISNOW  ,DZSNSO ,SNOWH  ,SNEQV  ,SNICE  , & !inout
                      SNLIQ  ,SH2O   ,SICE   ,STC    ,         & !inout
                      QSNBOT )                                   !out
! ----------------------------------------------------------------------
! Renew the mass of ice lens (SNICE) and liquid (SNLIQ) of the
! surface snow layer resulting from sublimation (frost) / evaporation (dew)
! ----------------------------------------------------------------------
   IMPLICIT NONE
! ----------------------------------------------------------------------
! input

   INTEGER,                         INTENT(IN) :: ipoint !
   INTEGER,                         INTENT(IN) :: NSNOW  !maximum no. of snow layers[=3]
```

```
   INTEGER,                       INTENT(IN) :: NSOIL  !No. of soil layers[=4]
   REAL,                          INTENT(IN) :: DT     !time step
   REAL,                          INTENT(IN) :: QSNFRO !snow surface frost rate[mm/s]
   REAL,                          INTENT(IN) :: QSNSUB !snow surface sublimation rate[mm/s]
   REAL,                          INTENT(IN) :: QRAIN  !snow surface rain rate[mm/s]

! output

   REAL, INTENT(OUT)                          :: QSNBOT !melting water out of snow bottom [mm/s]

! input and output

   INTEGER,                       INTENT(INOUT) :: ISNOW  !actual no. of snow layers
   REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: DZSNSO ! snow layer depth [m]
   REAL,                          INTENT(INOUT) :: SNOWH  !snow height [m]
   REAL,                          INTENT(INOUT) :: SNEQV  !snow water eqv. [mm]
   REAL, DIMENSION(-NSNOW+1:0),   INTENT(INOUT) :: SNICE  !snow layer ice [mm]
   REAL, DIMENSION(-NSNOW+1:0),   INTENT(INOUT) :: SNLIQ  !snow layer liquid water [mm]
   REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SH2O   !soil liquid moisture (m3/m3)
   REAL, DIMENSION(       1:NSOIL), INTENT(INOUT) :: SICE   !soil ice moisture (m3/m3)
   REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(INOUT) :: STC    !snow layer temperature [k]

! local variables:

   INTEGER                        :: J         !do loop/array indices
   REAL                           :: QIN       !water flow into the element (mm/s)
   REAL                           :: QOUT      !water flow out of the element (mm/s)
   REAL                           :: WGDIF     !ice mass after minus sublimation
   REAL, DIMENSION(-NSNOW+1:0)    :: VOL_LIQ   !partial volume of liquid water in layer
   REAL, DIMENSION(-NSNOW+1:0)    :: VOL_ICE   !partial volume of ice lens in layer
   REAL, DIMENSION(-NSNOW+1:0)    :: EPORE     !effective porosity = porosity - VOL_ICE
   REAL :: PROPOR, TEMP
! ----------------------------------------------------------------------

!for the case when SNEQV becomes '0' after 'COMBINE'

     IF(SNEQV == 0.) THEN
          SH2O(1) =  SH2O(1) + (QSNFRO-QSNSUB)*DT/(DZSNSO(1)*1000.)
     END IF

! for shallow snow without a layer
! snow surface sublimation may be larger than existing snow mass. To conserve water,
! excessive sublimation is used to reduce soil water. Smaller time steps would tend
! to aviod this problem.

     IF(ISNOW == 0. .and. SNEQV > 0.) THEN
        TEMP   = SNEQV
        SNEQV  = SNEQV - QSNSUB*DT + QSNFRO*DT
        PROPOR = SNEQV/TEMP
        SNOWH  = MAX(0.,PROPOR * SNOWH)

        IF(SNEQV < 0.) THEN
           SICE(1) = SICE(1) + SNEQV/(DZSNSO(1)*1000.)
           SNEQV   = 0.
        END IF
        IF(SICE(1) < 0.) THEN
           SH2O(1) = SH2O(1) + SICE(1)
           SICE(1) = 0.
        END IF
     END IF

      IF(SNOWH <= 1.E-8) SNOWH = 0.0
      IF(SNEQV <= 1.E-6) SNEQV = 0.0

! for deep snow

     WGDIF = SNICE(ISNOW+1) - QSNSUB*DT + QSNFRO*DT
     SNICE(ISNOW+1) = WGDIF
          IF (WGDIF < 1.e-6 .and. ISNOW <0) THEN
```

```
             CALL  COMBINE (NSNOW  ,NSOIL  ,ipoint,              & !in
                   ISNOW  ,SH2O   ,STC    ,SNICE  ,SNLIQ  ,  & !inout
                   DZSNSO ,SICE   ,SNOWH  ,SNEQV  )            !inout
           ENDIF
        SNLIQ(ISNOW+1) = SNLIQ(ISNOW+1) + QRAIN * DT
        SNLIQ(ISNOW+1) = MAX(0., SNLIQ(ISNOW+1))

! Porosity and partial volume

    DO J = -NSNOW+1, 0
          IF (J >= ISNOW+1) THEN
             VOL_ICE(J)      = MIN(1., SNICE(J)/(DZSNSO(J)*DENICE))
             EPORE(J)        = 1. - VOL_ICE(J)
             VOL_LIQ(J)      = MIN(EPORE(J),SNLIQ(J)/(DZSNSO(J)*DENH2O))
          END If
    END DO

    QIN = 0.
    QOUT = 0.

    DO J = -NSNOW+1, 0
          IF (J >= ISNOW+1) THEN
             SNLIQ(J) = SNLIQ(J) + QIN
             IF (J <= -1) THEN
                IF (EPORE(J) < 0.05 .OR. EPORE(J+1) < 0.05) THEN
                   QOUT = 0.
                ELSE
                   QOUT = MAX(0.,(VOL_LIQ(J)-SSI*EPORE(J))*DZSNSO(J))
                   QOUT = MIN(QOUT,(1.-VOL_ICE(J+1)-VOL_LIQ(J+1))*DZSNSO(J+1))
                END IF
             ELSE
                QOUT = MAX(0.,(VOL_LIQ(J) - SSI*EPORE(J))*DZSNSO(J))
             END IF
             QOUT = QOUT*1000.
             SNLIQ(J) = SNLIQ(J) - QOUT
             QIN = QOUT
          END IF
    end do

! Liquid water from snow bottom to soil

    QSNBOT = QOUT / DT           ! mm/s

  END SUBROUTINE SNOWH2O
! ==================================================================================================
  SUBROUTINE SOILWATER (NSOIL  ,NSNOW  ,DT     ,ZSOIL  ,DZSNSO , & !in
                        QINSUR ,QSEVA  ,ETRANI ,SICE   ,ipoint , & !in
                        SH2O   ,SMC    ,ZWT    ,                 & !inout
                        RUNSRF ,QDRAIN ,RUNSUB ,WCND   ,FCRMAX )   !out

! ----------------------------------------------------------------------
! calculate surface runoff and soil moisture.
! ----------------------------------------------------------------------
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! input
  INTEGER,                         INTENT(IN)  :: ipoint  !
  INTEGER,                         INTENT(IN)  :: NSOIL  !no. of soil layers
  INTEGER,                         INTENT(IN)  :: NSNOW  !maximum no. of snow layers
  REAL,                            INTENT(IN)  :: DT     !time step (sec)
  REAL, INTENT(IN)                             :: QINSUR !water input on soil surface [mm/s]
  REAL, INTENT(IN)                             :: QSEVA  !evap from soil surface [mm/s]
  REAL, DIMENSION(1:NSOIL),        INTENT(IN)  :: ZSOIL  !depth of soil layer-bottom [m]
  REAL, DIMENSION(1:NSOIL),        INTENT(IN)  :: ETRANI !evapotranspiration from soil layers [mm/s]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN)  :: DZSNSO !snow/soil layer depth [m]
  REAL, DIMENSION(1:NSOIL), INTENT(IN)    :: SICE   !soil ice content [m3/m3]

! input & output
```

```
  REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: SH2O   !soil liquid water content [m3/m3]
  REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: SMC    !total soil water content [m3/m3]
  REAL, INTENT(INOUT)                     :: ZWT    !water table depth [m]

! output
  REAL, INTENT(OUT)                       :: QDRAIN !soil-bottom free drainage [mm/s]
  REAL, INTENT(OUT)                       :: RUNSRF !surface runoff [mm/s]
  REAL, INTENT(OUT)                       :: RUNSUB !subsurface runoff [mm/s]
  REAL, INTENT(OUT)                       :: FCRMAX !maximum of FCR (-)
  REAL, DIMENSION(1:NSOIL), INTENT(OUT)   :: WCND   !hydraulic conductivity (m/s)

! local
  INTEGER                                 :: K,IZ   !do-loop index
  INTEGER                                 :: ITER   !iteration index
  REAl                                    :: DTFINE !fine time step (s)
  REAL, DIMENSION(1:NSOIL)                :: RHSTT  !right-hand side term of the matrix
  REAL, DIMENSION(1:NSOIL)                :: AI     !left-hand side term
  REAL, DIMENSION(1:NSOIL)                :: BI     !left-hand side term
  REAL, DIMENSION(1:NSOIL)                :: CI     !left-hand side term

  REAL                                    :: PDDUM  !infiltration rate at surface (m/s)
  REAL                                    :: FICE   !ice fraction in frozen soil
  REAL                                    :: WPLUS  !saturation excess of the total soil [m]
  REAL                                    :: RSAT   !accumulation of WPLUS (saturation excess) [m]
  REAL                                    :: SICEMAX!maximum soil ice content (m3/m3)
  REAL                                    :: SH2OMIN!minimum soil liquid water content (m3/m3)
  REAL                                    :: WTSUB  !sum of WCND(K)*DZSNSO(K)
  REAL                                    :: MH2O   !water mass removal (mm)
  REAL                                    :: FSAT   !fractional saturated area (-)
  REAL, DIMENSION(1:NSOIL)                :: MLIQ   !
  REAL                                    :: XS     !
  REAL                                    :: WATMIN !
  REAL                                    :: EPORE  !effective porosity [m3/m3]
  REAL, DIMENSION(1:NSOIL)                :: FCR    !impermeable fraction due to frozen soil
  INTEGER                                 :: NITER  !iteration times soil moisture (-)
  REAL                                    :: SMCTOT !2-m averaged soil moisture (m3/m3)
  REAL                                    :: DZTOT  !2-m soil depth (m)
  REAL, PARAMETER :: A = 4.0
! ----------------------------------------------------------------------
    RUNSRF = 0.0
    PDDUM  = 0.0
    RSAT   = 0.0

! for the case when snowmelt water is too large

    DO K = 1,NSOIL
       EPORE    = SMCMAX-SICE(K)
       RSAT     = RSAT + MAX(0.,SH2O(K)-EPORE)*DZSNSO(K)
       SH2O(K)  = MIN(EPORE,SH2O(K))
    END DO

!impermeable fraction due to frozen soil

    DO K = 1,NSOIL
       FICE     = MIN(1.0,SICE(K)/SMCMAX)
       FCR(K)   = MAX(0.0,EXP(-A*(1.-FICE))- EXP(-A)) /   &
                         (1.0              - EXP(-A))
    END DO

! maximum soil ice content and minimum liquid water of all layers

    SICEMAX = 0.0
    FCRMAX  = 0.0
    SH2OMIN = SMCMAX
    DO K = 1,NSOIL
       IF (SICE(K) > SICEMAX) SICEMAX = SICE(K)
       IF (FCR(K)  > FCRMAX)  FCRMAX  = FCR(K)
       IF (SH2O(K) < SH2OMIN) SH2OMIN = SH2O(K)
    END DO
```

```
!subsurface runoff for runoff scheme option 2

    IF(OPT_RUN == 2) THEN
       FFF   = 2.0
       RSBMX = 4.0
       CALL ZWTEQ (NSOIL  ,NSNOW  ,ZSOIL  ,DZSNSO ,SH2O   ,ZWT)
       RUNSUB = (1.0-FCRMAX) * RSBMX * EXP(-TIMEAN) * EXP(-FFF*ZWT)   ! mm/s
    END IF

!surface runoff and infiltration rate using different schemes

    IF(OPT_RUN == 1) THEN
      FSAT   = FSATMX*EXP(-0.5*FFF*(ZWT-2.0))
      IF(QINSUR > 0.) THEN
        FFF    = 6.0
        RUNSRF = QINSUR * ( (1.0-FCR(1))*FSAT + FCR(1) )
        PDDUM  = QINSUR - RUNSRF                          ! m/s
      END IF
    END IF

    IF(OPT_RUN == 2) THEN
      FFF   = 2.0
      FSAT   = FSATMX*EXP(-0.5*FFF*ZWT)
      IF(QINSUR > 0.) THEN
        RUNSRF = QINSUR * ( (1.0-FCR(1))*FSAT + FCR(1) )
        PDDUM  = QINSUR - RUNSRF                          ! m/s
      END IF
    END IF

    IF(OPT_RUN == 3) THEN
      CALL INFIL (NSOIL  ,DT     ,ZSOIL  ,SH2O   ,SICE   , & !in
                  SICEMAX,QINSUR ,                         & !in
                  PDDUM  ,RUNSRF )                           !out
    END IF

    IF(OPT_RUN == 4) THEN
      SMCTOT = 0.
      DZTOT  = 0.
      DO K = 1,NSOIL
         DZTOT   = DZTOT  + DZSNSO(K)
         IF(DZTOT >= 2.0) EXIT
         SMCTOT  = SMCTOT + SMC(K)*DZSNSO(K)
      END DO
      SMCTOT = SMCTOT/DZTOT
      FSAT   = MAX(0.01,SMCTOT/SMCMAX) ** 4.        !BATS

      IF(QINSUR > 0.) THEN
        RUNSRF = QINSUR * ((1.0-FCR(1))*FSAT+FCR(1))
        PDDUM  = QINSUR - RUNSRF                       ! m/s
      END IF
    END IF

! determine iteration times and finer time step

    NITER = 1

    IF(OPT_INF == 1) THEN    !OPT_INF =2 may cause water imbalance
      NITER = 3
      IF (PDDUM*DT>DZSNSO(1)*SMCMAX ) THEN
         NITER = NITER*2
      END IF
    END IF

    DTFINE  = DT / NITER

! solve soil moisture

    DO ITER = 1, NITER
```

```
       CALL SRT   (NSOIL  ,ZSOIL  ,DTFINE ,PDDUM  ,ETRANI ,  & !in
                   QSEVA  ,SH2O   ,SMC    ,ZWT    ,FCR    ,  & !in
                   SICEMAX,FCRMAX ,ipoint ,                  & !in
                   RHSTT  ,AI     ,BI     ,CI     ,QDRAIN ,  & !out
                   WCND   )                                    !out

       CALL SSTEP (NSOIL  ,NSNOW  ,DTFINE ,ZSOIL  ,DZSNSO ,  & !in
                   SICE   ,ipoint ,                          & !in
                   SH2O   ,SMC    ,AI     ,BI     ,CI     ,  & !inout
                   RHSTT  ,                                  & !inout
                   WPLUS)                                      !out
       RSAT =  RSAT + WPLUS
    END DO

    RUNSRF = RUNSRF * 1000. + RSAT * 1000./DT  ! m/s -> mm/s
    QDRAIN = QDRAIN * 1000.

! removal of soil water due to groundwater flow (option 2)

    IF(OPT_RUN == 2) THEN
         WTSUB = 0.
         DO K = 1, NSOIL
           WTSUB = WTSUB + WCND(K)*DZSNSO(K)
         END DO

         DO K = 1, NSOIL
           MH2O    = RUNSUB*DT*(WCND(K)*DZSNSO(K))/WTSUB       ! mm
           SH2O(K) = SH2O(K) - MH2O/(DZSNSO(K)*1000.)
         END DO
    END IF

! Limit MLIQ to be greater than or equal to watmin.
! Get water needed to bring MLIQ equal WATMIN from lower layer.

   IF(OPT_RUN /= 1) THEN
      DO IZ = 1, NSOIL
         MLIQ(IZ) = SH2O(IZ)*DZSNSO(IZ)*1000.
      END DO

      WATMIN = 0.01           ! mm
      DO IZ = 1, NSOIL-1
          IF (MLIQ(IZ) .LT. 0.) THEN
             XS = WATMIN-MLIQ(IZ)
          ELSE
             XS = 0.
          END IF
          MLIQ(IZ  ) = MLIQ(IZ  ) + XS
          MLIQ(IZ+1) = MLIQ(IZ+1) - XS
      END DO

        IZ = NSOIL
        IF (MLIQ(IZ) .LT. WATMIN) THEN
           XS = WATMIN-MLIQ(IZ)
        ELSE
           XS = 0.
        END IF
        MLIQ(IZ) = MLIQ(IZ) + XS
        RUNSUB   = RUNSUB - XS/DT

      DO IZ = 1, NSOIL
        SH2O(IZ)     = MLIQ(IZ) / (DZSNSO(IZ)*1000.)
      END DO
   END IF

  END SUBROUTINE SOILWATER
! ==================================================================================================
  SUBROUTINE ZWTEQ (NSOIL  ,NSNOW  ,ZSOIL  ,DZSNSO ,SH2O   ,ZWT)
! ----------------------------------------------------------------------
! calculate equilibrium water table depth (Niu et al., 2005)
```

```
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! input

  INTEGER,                         INTENT(IN) :: NSOIL  !no. of soil layers
  INTEGER,                         INTENT(IN) :: NSNOW  !maximum no. of snow layers
  REAL, DIMENSION(1:NSOIL),        INTENT(IN) :: ZSOIL  !depth of soil layer-bottom [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !snow/soil layer depth [m]
  REAL, DIMENSION(1:NSOIL),        INTENT(IN) :: SH2O   !soil liquid water content [m3/m3]

! output

  REAL,                           INTENT(OUT) :: ZWT    !water table depth [m]

! locals

  INTEGER :: K                      !do-loop index
  INTEGER, PARAMETER :: NFINE = 100 !no. of fine soil layers of 6m soil
  REAL    :: WD1                    !water deficit from coarse (4-L) soil moisture profile
  REAL    :: WD2                    !water deficit from fine (100-L) soil moisture profile
  REAL    :: DZFINE                 !layer thickness of the 100-L soil layers to 6.0 m
  REAL    :: TEMP                   !temporary variable
  REAL, DIMENSION(1:NFINE) :: ZFINE !layer-bottom depth of the 100-L soil layers to 6.0 m
! ----------------------------------------------------------------------

   WD1 = 0.
   DO K = 1,NSOIL
     WD1 = WD1 + (SMCMAX-SH2O(K)) * DZSNSO(K) ! [m]
   ENDDO

   DZFINE = 3.0 * (-ZSOIL(NSOIL)) / NFINE
   do K =1,NFINE
      ZFINE(K) = FLOAT(K) * DZFINE
   ENDDO

   ZWT = -3.*ZSOIL(NSOIL) - 0.001   ! initial value [m]

   WD2 = 0.
   DO K = 1,NFINE
     TEMP  = 1. + (ZWT-ZFINE(K))/PSISAT
     WD2   = WD2 + SMCMAX*(1.-TEMP**(-1./BEXP))*DZFINE
     IF(ABS(WD2-WD1).LE.0.01) THEN
        ZWT = ZFINE(K)
        EXIT
     ENDIF
   ENDDO

  END SUBROUTINE ZWTEQ
! ----------------------------------------------------------------------
! ==================================================================================================
  SUBROUTINE INFIL (NSOIL  ,DT     ,ZSOIL  ,SH2O   ,SICE   , & !in
                    SICEMAX,QINSUR ,                         & !in
                    PDDUM  ,RUNSRF )                           !out
! --------------------------------------------------------------------------------
! compute inflitration rate at soil surface and surface runoff
! --------------------------------------------------------------------------------
    IMPLICIT NONE
! --------------------------------------------------------------------------------
! inputs
  INTEGER,                  INTENT(IN) :: NSOIL  !no. of soil layers
  REAL,                     INTENT(IN) :: DT     !time step (sec)
  REAL, DIMENSION(1:NSOIL), INTENT(IN) :: ZSOIL  !depth of soil layer-bottom [m]
  REAL, DIMENSION(1:NSOIL), INTENT(IN) :: SH2O   !soil liquid water content [m3/m3]
  REAL, DIMENSION(1:NSOIL), INTENT(IN) :: SICE   !soil ice content [m3/m3]
  REAL,                     INTENT(IN) :: QINSUR !water input on soil surface [mm/s]
  REAL,                     INTENT(IN) :: SICEMAX!maximum soil ice content (m3/m3)

! outputs
```

```
  REAL,                      INTENT(OUT) :: RUNSRF !surface runoff [mm/s]
  REAL,                      INTENT(OUT) :: PDDUM  !infiltration rate at surface

! locals
  INTEGER :: IALP1, J, JJ,  K
  REAL                       :: VAL
  REAL                       :: DDT
  REAL                       :: PX
  REAL                       :: DT1, DD, DICE
  REAL                       :: FCR
  REAL                       :: SUM
  REAL                       :: ACRT
  REAL                       :: WDF
  REAL                       :: WCND
  REAL                       :: SMCAV
  REAL                       :: INFMAX
  REAL, DIMENSION(1:NSOIL)  :: DMAX
  INTEGER, PARAMETER         :: CVFRZ = 3
! ----------------------------------------------------------------------------------

    IF (QINSUR >  0.0) THEN
       DT1 = DT /86400.
       SMCAV = SMCMAX - SMCWLT

! maximum infiltration rate

       DMAX(1)= -ZSOIL(1) * SMCAV
       DICE   = -ZSOIL(1) * SICE(1)
       DMAX(1)= DMAX(1)* (1.0-(SH2O(1) + SICE(1) - SMCWLT)/SMCAV)

       DD = DMAX(1)

       DO K = 2,NSOIL
          DICE    = DICE + (ZSOIL(K-1) - ZSOIL(K) ) * SICE(K)
          DMAX(K) = (ZSOIL(K-1) - ZSOIL(K)) * SMCAV
          DMAX(K) = DMAX(K) * (1.0-(SH2O(K) + SICE(K) - SMCWLT)/SMCAV)
          DD      = DD + DMAX(K)
       END DO

       VAL = (1. - EXP ( - KDT * DT1))
       DDT = DD * VAL
       PX  = MAX(0.,QINSUR * DT)
       INFMAX = (PX * (DDT / (PX + DDT)))/ DT

! impermeable fraction due to frozen soil

       FCR = 1.
       IF (DICE >  1.E-2) THEN
          ACRT = CVFRZ * FRZX / DICE
          SUM = 1.
          IALP1 = CVFRZ - 1
          DO J = 1,IALP1
             K = 1
             DO JJ = J +1,IALP1
                K = K * JJ
             END DO
             SUM = SUM + (ACRT ** (CVFRZ - J)) / FLOAT(K)
          END DO
          FCR = 1. - EXP (-ACRT) * SUM
       END IF

! correction of infiltration limitation

       INFMAX = INFMAX * FCR

       CALL WDFCND2 (WDF,WCND,SH2O(1),SICEMAX)
       INFMAX = MAX (INFMAX,WCND)
       INFMAX = MIN (INFMAX,PX)
```

```
       RUNSRF= MAX(0., QINSUR - INFMAX)
       PDDUM = QINSUR - RUNSRF

    END IF

  END SUBROUTINE INFIL
! ==================================================================================================
  SUBROUTINE SRT (NSOIL  ,ZSOIL  ,DT     ,PDDUM  ,ETRANI , & !in
                  QSEVA  ,SH2O   ,SMC    ,ZWT    ,FCR    , & !in
                  SICEMAX,FCRMAX ,ipoint ,                 & !in
                  RHSTT  ,AI     ,BI     ,CI     ,QDRAIN , & !out
                  WCND   )                                   !out
! ----------------------------------------------------------------------
! calculate the right hand side of the time tendency term of the soil
! water diffusion equation.  also to compute ( prepare ) the matrix
! coefficients for the tri-diagonal matrix of the implicit time scheme.
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
!input

    INTEGER,                  INTENT(IN)  :: ipoint  !
    INTEGER,                  INTENT(IN)  :: NSOIL
    REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: ZSOIL
    REAL,                     INTENT(IN)  :: DT
    REAL,                     INTENT(IN)  :: PDDUM
    REAL,                     INTENT(IN)  :: QSEVA
    REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: ETRANI
    REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: SH2O
    REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: SMC
    REAL,                     INTENT(IN)  :: ZWT    ! water table depth [m]
    REAL, DIMENSION(1:NSOIL), INTENT(IN)  :: FCR
    REAL, INTENT(IN)                      :: FCRMAX !maximum of FCR (-)
    REAL,                     INTENT(IN)  :: SICEMAX!maximum soil ice content (m3/m3)

! output

    REAL, DIMENSION(1:NSOIL), INTENT(OUT) :: RHSTT
    REAL, DIMENSION(1:NSOIL), INTENT(OUT) :: AI
    REAL, DIMENSION(1:NSOIL), INTENT(OUT) :: BI
    REAL, DIMENSION(1:NSOIL), INTENT(OUT) :: CI
    REAL, DIMENSION(1:NSOIL), INTENT(OUT) :: WCND    !hydraulic conductivity (m/s)
    REAL,                     INTENT(OUT) :: QDRAIN  !bottom drainage (m/s)

! local
    INTEGER                               :: K
    REAL, DIMENSION(1:NSOIL)              :: DDZ
    REAL, DIMENSION(1:NSOIL)              :: DENOM
    REAL, DIMENSION(1:NSOIL)              :: DSMDZ
    REAL, DIMENSION(1:NSOIL)              :: WFLUX
    REAL, DIMENSION(1:NSOIL)              :: WDF
    REAL, DIMENSION(1:NSOIL)              :: SMX
    REAL                                  :: TEMP1

! Niu and Yang (2006), J. of Hydrometeorology
! ----------------------------------------------------------------------

    IF(OPT_INF == 1) THEN
      DO K = 1, NSOIL
        CALL WDFCND1 (WDF(K),WCND(K),SMC(K),FCR(K))
        SMX(K) = SMC(K)
      END DO
    END IF

    IF(OPT_INF == 2) THEN
      DO K = 1, NSOIL
        CALL WDFCND2 (WDF(K),WCND(K),SH2O(K),SICEMAX)
        SMX(K) = SH2O(K)
      END DO
```

```
    END IF

    DO K = 1, NSOIL
       IF(K == 1) THEN
          DENOM(K) = - ZSOIL (K)
          TEMP1    = - ZSOIL (K+1)
          DDZ(K)   = 2.0 / TEMP1
          DSMDZ(K) = 2.0 * (SMX(K) - SMX(K+1)) / TEMP1
          WFLUX(K) = WDF(K) * DSMDZ(K) + WCND(K) - PDDUM + ETRANI(K) + QSEVA
       ELSE IF (K < NSOIL) THEN
          DENOM(k) = (ZSOIL(K-1) - ZSOIL(K))
          TEMP1    = (ZSOIL(K-1) - ZSOIL(K+1))
          DDZ(K)   = 2.0 / TEMP1
          DSMDZ(K) = 2.0 * (SMX(K) - SMX(K+1)) / TEMP1
          WFLUX(K) = WDF(K  ) * DSMDZ(K  ) + WCND(K  )          &
                   - WDF(K-1) * DSMDZ(K-1) - WCND(K-1) + ETRANI(K)
       ELSE
          DENOM(K) = (ZSOIL(K-1) - ZSOIL(K))
          IF(OPT_RUN == 1 .or. OPT_RUN == 2) THEN
             QDRAIN   = 0.
          END IF
          IF(OPT_RUN == 3) THEN
             QDRAIN   = SLOPE*WCND(K)
          END IF
          IF(OPT_RUN == 4) THEN
             QDRAIN   = (1.0-FCRMAX)*WCND(K)
          END IF
          WFLUX(K) = -(WDF(K-1)*DSMDZ(K-1))-WCND(K-1)+ETRANI(K) + QDRAIN
       END IF
    END DO

    DO K = 1, NSOIL
       IF(K == 1) THEN
          AI(K)    =   0.0
          BI(K)    =   WDF(K  ) * DDZ(K  ) / DENOM(K)
          CI(K)    = - BI (K)
       ELSE IF (K < NSOIL) THEN
          AI(K)    = - WDF(K-1) * DDZ(K-1) / DENOM(K)
          CI(K)    = - WDF(K  ) * DDZ(K  ) / DENOM(K)
          BI(K)    = - ( AI (K) + CI (K) )
       ELSE
          AI(K)    = - WDF(K-1) * DDZ(K-1) / DENOM(K)
          CI(K)    = 0.0
          BI(K)    = - ( AI (K) + CI (K) )
       END IF
          RHSTT(K) = WFLUX(K) / (-DENOM(K))
    END DO

! ----------------------------------------------------------------------
  END SUBROUTINE SRT
! ----------------------------------------------------------------------
! ==================================================================================================
  SUBROUTINE SSTEP (NSOIL  ,NSNOW  ,DT     ,ZSOIL  ,DZSNSO , & !in
                    SICE   ,ipoint ,                         & !in
                    SH2O   ,SMC    ,AI     ,BI     ,CI     , & !inout
                    RHSTT  ,                                 & !inout
                    WPLUS  )                                   !out

! ----------------------------------------------------------------------
! calculate/update soil moisture content values
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
!input

    INTEGER,                         INTENT(IN) :: ipoint  !
    INTEGER,                         INTENT(IN) :: NSOIL  !
    INTEGER,                         INTENT(IN) :: NSNOW  !
    REAL, INTENT(IN)                            :: DT
```

```
    REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: ZSOIL
    REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: SICE
    REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO ! snow/soil layer thickness [m]

!input and output
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: SH2O
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: SMC
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: AI
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: BI
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: CI
    REAL, DIMENSION(1:NSOIL), INTENT(INOUT) :: RHSTT

!output
    REAL, INTENT(OUT)                       :: WPLUS     !saturation excess water (m)

!local
    INTEGER                                 :: K
    REAL, DIMENSION(1:NSOIL)                :: RHSTTIN
    REAL, DIMENSION(1:NSOIL)                :: CIIN
    REAL                                    :: STOT
    REAL                                    :: EPORE
! ----------------------------------------------------------------------
    WPLUS = 0.0

    DO K = 1,NSOIL
       RHSTT (K) =   RHSTT(K) * DT
       AI (K)    =      AI(K) * DT
       BI (K)    = 1. + BI(K) * DT
       CI (K)    =      CI(K) * DT
    END DO

! copy values for input variables before calling rosr12

    DO K = 1,NSOIL
       RHSTTIN(k) = RHSTT(K)
       CIIN(k)    = CI(K)
    END DO

! call ROSR12 to solve the tri-diagonal matrix

    CALL ROSR12 (CI,AI,BI,CIIN,RHSTTIN,RHSTT,1,NSOIL,0)

    DO K = 1,NSOIL
        SH2O(K) = SH2O(K) + CI(K)
    ENDDO

!  excessive water above saturation in a layer is moved to
!  its unsaturated layer like in a bucket

    DO K = NSOIL,2,-1
      EPORE         = SMCMAX - SICE(K)
      WPLUS         = MAX((SH2O(K)-EPORE), 0.0) * DZSNSO(K)
      SH2O(K)       = MIN(EPORE,SH2O(K))
      SH2O(K-1)     = SH2O(K-1) + WPLUS/DZSNSO(K-1)
    END DO

    EPORE         = SMCMAX - SICE(1)
    WPLUS         = MAX((SH2O(1)-EPORE), 0.0) * DZSNSO(1)
    SH2O(1)       = MIN(EPORE,SH2O(1))

  END SUBROUTINE SSTEP
! ==================================================================================================
  SUBROUTINE WDFCND1 (WDF,WCND,SMC,FCR)
! ----------------------------------------------------------------------
! calculate soil water diffusivity and soil hydraulic conductivity.
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input
```

```
    REAL,INTENT(IN)   :: SMC
    REAL,INTENT(IN)   :: FCR

! output
    REAL,INTENT(OUT)  :: WCND
    REAL,INTENT(OUT)  :: WDF

! local
    REAL :: EXPON
    REAL :: FACTR
    REAL :: VKWGT
! ----------------------------------------------------------------------

! soil water diffusivity

    FACTR = MAX(0.01, SMC/SMCMAX)
    EXPON = BEXP + 2.0
    WDF   = DWSAT * FACTR ** EXPON
    WDF   = WDF * (1.0 - FCR)

! hydraulic conductivity

    EXPON = 2.0*BEXP + 3.0
    WCND  = DKSAT * FACTR ** EXPON
    WCND  = WCND * (1.0 - FCR)

  END SUBROUTINE WDFCND1
! ==================================================================================
  SUBROUTINE WDFCND2 (WDF,WCND,SMC,SICE)
! ----------------------------------------------------------------------
! calculate soil water diffusivity and soil hydraulic conductivity.
! ----------------------------------------------------------------------
    IMPLICIT NONE
! ----------------------------------------------------------------------
! input
    REAL,INTENT(IN)   :: SMC
    REAL,INTENT(IN)   :: SICE

! output
    REAL,INTENT(OUT)  :: WCND
    REAL,INTENT(OUT)  :: WDF

! local
    REAL :: EXPON
    REAL :: FACTR
    REAL :: VKWGT
! ----------------------------------------------------------------------

! soil water diffusivity

    FACTR = MAX(0.01, SMC/SMCMAX)
    EXPON = BEXP + 2.0
    WDF   = DWSAT * FACTR ** EXPON

    IF (SICE > 0.0) THEN
    VKWGT = 1./ (1. + (500.* SICE)**3.)
    WDF   = VKWGT * WDF + (1.-VKWGT)*DWSAT*(0.2/SMCMAX)**EXPON
    END IF

! hydraulic conductivity

    EXPON = 2.0*BEXP + 3.0
    WCND  = DKSAT * FACTR ** EXPON

  END SUBROUTINE WDFCND2
! ==================================================================================
! ----------------------------------------------------------------------
  SUBROUTINE GROUNDWATER(NSNOW   ,NSOIL   ,DT      ,SICE    ,ZSOIL   , & !in
                         STC     ,WCND    ,FCRMAX  ,ipoint ,           & !in
```

```
                                 SH2O    ,ZWT     ,WA      ,WT      ,           & !inout
                                 QIN     ,QDIS    )                               !out
! ----------------------------------------------------------------------
  IMPLICIT NONE
! ----------------------------------------------------------------------
! input
  INTEGER,                         INTENT(IN) :: ipoint!
  INTEGER,                         INTENT(IN) :: NSNOW !maximum no. of snow layers
  INTEGER,                         INTENT(IN) :: NSOIL !no. of soil layers
  REAL,                            INTENT(IN) :: DT    !timestep [sec]
  REAL,                            INTENT(IN) :: FCRMAX!maximum FCR (-)
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: SICE  !soil ice content [m3/m3]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: ZSOIL !depth of soil layer-bottom [m]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: WCND  !hydraulic conductivity (m/s)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: STC   !snow/soil temperature (k)

! input and output
  REAL, DIMENSION(    1:NSOIL), INTENT(INOUT) :: SH2O  !liquid soil water [m3/m3]
  REAL,                         INTENT(INOUT) :: ZWT   !the depth to water table [m]
  REAL,                         INTENT(INOUT) :: WA    !water storage in aquifer [mm]
  REAL,                         INTENT(INOUT) :: WT    !water storage in aquifer
                                                           !+ saturated soil [mm]
! output
  REAL,                           INTENT(OUT) :: QIN   !groundwater recharge [mm/s]
  REAL,                           INTENT(OUT) :: QDIS  !groundwater discharge [mm/s]

! local
  INTEGER                                     :: IZ    !do-loop index
  INTEGER                                     :: IWT   !layer index above water table layer
  REAL,  DIMENSION(    1:NSOIL)               :: DZMM  !layer thickness [mm]
  REAL,  DIMENSION(    1:NSOIL)               :: ZNODE !node depth [m]
  REAL,  DIMENSION(    1:NSOIL)               :: MLIQ  !liquid water mass [kg/m2 or mm]
  REAL,  DIMENSION(    1:NSOIL)               :: EPORE !effective porosity [-]
  REAL,  DIMENSION(    1:NSOIL)               :: HK    !hydraulic conductivity [mm/s]
  REAL,  DIMENSION(    1:NSOIL)               :: SMC   !total soil water  content [m3/m3]
  REAL*8                                      :: S_NODE!degree of saturation of IWT layer
  REAL                                        :: DZSUM !cumulative depth above water table [m]
  REAL                                        :: SMPFZ !matric potential (frozen effects) [mm]
  REAL                                        :: KA    !aquifer hydraulic conductivity [mm/s]
  REAL                                        :: WH_ZWT!water head at water table [mm]
  REAL                                        :: WH    !water head at layer above ZWT [mm]
  REAL                                        :: WS    !water used to fill air pore [mm]
  REAL                                        :: WTSUB !sum of HK*DZMM
  REAL                                        :: WATMIN!minimum soil vol soil moisture [m3/m3]
  REAL                                        :: XS    !excessive water above saturation [mm]
  REAL, PARAMETER                             :: ROUS = 0.2    !specific yield [-]
  REAL, PARAMETER                             :: CMIC = 0.20   !microprore content (0.0-1.0)
                                                               !0.0-close to free drainage
! -------------------------------------------------------------
      QDIS      = 0.0
      QIN       = 0.0

! Derive layer-bottom depth in [mm]

      DZMM(1) = -ZSOIL(1)*1.E3
      DO IZ = 2, NSOIL
         DZMM(IZ)  = 1.E3 * (ZSOIL(IZ - 1) - ZSOIL(IZ))
      ENDDO

! Derive node (middle) depth in [m]

      ZNODE(1) = -ZSOIL(1) / 2.
      DO IZ = 2, NSOIL
         ZNODE(IZ)  = -ZSOIL(IZ-1) + 0.5 * (ZSOIL(IZ-1) - ZSOIL(IZ))
      ENDDO

! Convert volumetric soil moisture "sh2o" to mass

      DO IZ = 1, NSOIL
```

```
         SMC(IZ)      = SH2O(IZ) + SICE(IZ)
         MLIQ(IZ)     = SH2O(IZ) * DZMM(IZ)
         EPORE(IZ)    = MAX(0.01,SMCMAX - SICE(IZ))
         HK(IZ)       = 1.E3*WCND(IZ)
      ENDDO

! The layer index of the first unsaturated layer,
! i.e., the layer right above the water table

      IWT = NSOIL
      DO IZ = 2,NSOIL
         IF(ZWT   .LE.  -ZSOIL(IZ) ) THEN
            IWT = IZ-1
            GOTO 888
         END IF
      ENDDO
 888  CONTINUE

! Groundwater discharge [mm/s]

      FFF   = 6.0
      RSBMX = 5.0

      QDIS = (1.0-FCRMAX)*RSBMX*EXP(-TIMEAN)*EXP(-FFF*(ZWT-2.0))

! Matric potential at the layer above the water table

      S_NODE = MIN(1.0,SMC(IWT)/SMCMAX )
      S_NODE = MAX(S_NODE,0.01)
      SMPFZ  = -PSISAT*1000.*S_NODE**(-BEXP)   ! m --> mm
      SMPFZ  = MAX(-120000.0,CMIC*SMPFZ)

! Recharge rate qin to groundwater

      KA  = HK(IWT)

      WH_ZWT  = - ZWT * 1.E3                          !(mm)
      WH      = SMPFZ  - ZNODE(IWT)*1.E3              !(mm)
      QIN     = - KA * (WH_ZWT-WH)  /((ZWT-ZNODE(IWT))*1.E3)
      QIN     = MAX(-10.0/DT,MIN(10./DT,QIN))

! Water storage in the aquifer + saturated soil

      WT  = WT + (QIN - QDIS) * DT     !(mm)

      IF(IWT.EQ.NSOIL) THEN
         WA          = WA + (QIN - QDIS) * DT     !(mm)
         WT          = WA
         ZWT         = (-ZSOIL(NSOIL) + 25.) - WA/1000./ROUS      !(m)
         MLIQ(NSOIL) = MLIQ(NSOIL) - QIN * DT        ! [mm]

         MLIQ(NSOIL) = MLIQ(NSOIL) + MAX(0.,(WA - 5000.))
         WA          = MIN(WA, 5000.)
      ELSE

         IF (IWT.EQ.NSOIL-1) THEN
            ZWT = -ZSOIL(NSOIL)                   &
                 - (WT-ROUS*1000*25.) / (EPORE(NSOIL))/1000.
         ELSE
            WS = 0.   ! water used to fill soil air pores
            DO IZ = IWT+2,NSOIL
               WS = WS + EPORE(IZ) * DZMM(IZ)
            ENDDO
            ZWT = -ZSOIL(IWT+1)                  &
                  - (WT-ROUS*1000.*25.-WS) /(EPORE(IWT+1))/1000.
         ENDIF

         WTSUB = 0.
         DO IZ = 1, NSOIL
```

```
           WTSUB = WTSUB + HK(IZ)*DZMM(IZ)
         END DO

         DO IZ = 1, NSOIL           ! Removing subsurface runoff
         MLIQ(IZ) = MLIQ(IZ) - QDIS*DT*HK(IZ)*DZMM(IZ)/WTSUB
         END DO
      END IF

      ZWT = MAX(1.5,ZWT)

!
! Limit MLIQ to be greater than or equal to watmin.
! Get water needed to bring MLIQ equal WATMIN from lower layer.
!
      WATMIN = 0.01
      DO IZ = 1, NSOIL-1
          IF (MLIQ(IZ) .LT. 0.) THEN
             XS = WATMIN-MLIQ(IZ)
          ELSE
             XS = 0.
          END IF
          MLIQ(IZ  ) = MLIQ(IZ  ) + XS
          MLIQ(IZ+1) = MLIQ(IZ+1) - XS
      END DO

        IZ = NSOIL
        IF (MLIQ(IZ) .LT. WATMIN) THEN
           XS = WATMIN-MLIQ(IZ)
        ELSE
           XS = 0.
        END IF
        MLIQ(IZ) = MLIQ(IZ) + XS
        WA       = WA - XS
        WT       = WT - XS

      DO IZ = 1, NSOIL
        SH2O(IZ)      = MLIQ(IZ) / DZMM(IZ)
      END DO

  END SUBROUTINE GROUNDWATER
! ====================================================================================================
! ******************** end of water subroutines ******************************************
! ====================================================================================================
  SUBROUTINE CARBON (NSNOW  ,NSOIL  ,VEGTYP ,NROOT  ,DT     ,ZSOIL  , & !in
                     DZSNSO ,STC    ,SMC    ,TV     ,TG     ,PSN    , & !in
                     FOLN   ,SMCMAX ,BTRAN  ,APAR   ,FVEG   ,IGS    , & !in
                     TROOT  ,IST    ,IMONTH ,LAT    ,ipoint ,           & !in
                     LFMASS ,RTMASS ,STMASS ,WOOD   ,STBLCP ,FASTCP , & !inout
                     GPP    ,NPP    ,NEE    ,AUTORS ,HETERS ,TOTSC  , & !out
                     TOTLB  ,XLAI   ,XSAI   )                   !out
! --------------------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! --------------------------------------------------------------------------------------------------
      IMPLICIT NONE
! --------------------------------------------------------------------------------------------------
! inputs (carbon)

  INTEGER                        , INTENT(IN) :: ipoint !grid index
  INTEGER                        , INTENT(IN) :: VEGTYP !vegetation type
  INTEGER                        , INTENT(IN) :: IMONTH !month index
  INTEGER                        , INTENT(IN) :: NSNOW  !number of snow layers
  INTEGER                        , INTENT(IN) :: NSOIL  !number of soil layers
  INTEGER                        , INTENT(IN) :: NROOT  !no. of root layers
  REAL                           , INTENT(IN) :: LAT    !latitude (radians)
  REAL                           , INTENT(IN) :: DT     !time step (s)
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: ZSOIL  !depth of layer-bottom from soil surface
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !snow/soil layer thickness [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: STC    !snow/soil temperature [k]
  REAL, DIMENSION(       1:NSOIL), INTENT(IN) :: SMC    !soil moisture (ice + liq.) [m3/m3]
```

```
  REAL                              , INTENT(IN) :: TV     !vegetation temperature (k)
  REAL                              , INTENT(IN) :: TG     !ground temperature (k)
  REAL                              , INTENT(IN) :: FOLN   !foliage nitrogen (%)
  REAL                              , INTENT(IN) :: SMCMAX !soil porosity (m3/m3)
  REAL                              , INTENT(IN) :: BTRAN  !soil water transpiration factor (0 to 1)
  REAL                              , INTENT(IN) :: PSN    !total leaf photosyn (umolco2/m2/s) [+]
  REAL                              , INTENT(IN) :: APAR   !PAR by canopy (w/m2)
  REAL                              , INTENT(IN) :: IGS    !growing season index (0=off, 1=on)
  REAL                              , INTENT(IN) :: FVEG   !vegetation greenness fraction
  REAL                              , INTENT(IN) :: TROOT  !root-zone averaged temperature (k)
  INTEGER                           , INTENT(IN) :: IST    !surface type 1->soil; 2->lake

! input & output (carbon)

  REAL                           , INTENT(INOUT) :: LFMASS !leaf mass [g/m2]
  REAL                           , INTENT(INOUT) :: RTMASS !mass of fine roots [g/m2]
  REAL                           , INTENT(INOUT) :: STMASS !stem mass [g/m2]
  REAL                           , INTENT(INOUT) :: WOOD   !mass of wood (incl. woody roots) [g/m2]
  REAL                           , INTENT(INOUT) :: STBLCP !stable carbon in deep soil [g/m2]
  REAL                           , INTENT(INOUT) :: FASTCP !short-lived carbon in shallow soil [g/m2]

! outputs: (carbon)

  REAL                             , INTENT(OUT) :: GPP    !net instantaneous assimilation [g/m2/s C]
  REAL                             , INTENT(OUT) :: NPP    !net primary productivity [g/m2/s C]
  REAL                             , INTENT(OUT) :: NEE    !net ecosystem exchange [g/m2/s CO2]
  REAL                             , INTENT(OUT) :: AUTORS !net ecosystem respiration [g/m2/s C]
  REAL                             , INTENT(OUT) :: HETERS !organic respiration [g/m2/s C]
  REAL                             , INTENT(OUT) :: TOTSC  !total soil carbon [g/m2 C]
  REAL                             , INTENT(OUT) :: TOTLB  !total living carbon ([g/m2 C]
  REAL                             , INTENT(OUT) :: XLAI   !leaf area index [-]
  REAL                             , INTENT(OUT) :: XSAI   !stem area index [-]
!   REAL                            , INTENT(OUT) :: VOCFLX(5) ! voc fluxes [ug C m-2 h-1]

! local variables

  INTEGER :: J         !do-loop index
  REAL    :: WROOT     !root zone soil water [-]
  REAL    :: WSTRES    !water stress coeficient [-]  (1. for wilting )
  REAL    :: LAPM      !leaf area per unit mass [m2/g]
! -------------------------------------------------------------------------------------------

   IF(VEGTYP == 16 .OR. VEGTYP == 19 .OR. VEGTYP == 24) THEN
      XLAI   = 0.
      XSAI   = 0.
      GPP    = 0.
      NPP    = 0.
      NEE    = 0.
      AUTORS = 0.
      HETERS = 0.
      TOTSC  = 0.
      TOTLB  = 0.
      LFMASS = 0.
      RTMASS = 0.
      STMASS = 0.
      WOOD   = 0.
      STBLCP = 0.
      FASTCP = 0.

      RETURN
   END IF

      LAPM       = SLA(VEGTYP) / 1000.   ! m2/kg -> m2/g

! water stress

      WSTRES  = 1.- BTRAN

      WROOT  = 0.
```

```
      DO J=1,NROOT
        WROOT = WROOT + SMC(J)/SMCMAX *  DZSNSO(J) / (-ZSOIL(NROOT))
      ENDDO

  CALL CO2FLUX (NSNOW  ,NSOIL  ,VEGTYP ,IGS    ,DT     , & !in
                DZSNSO ,STC    ,PSN    ,TROOT  ,TV     , & !in
                WROOT  ,WSTRES ,FOLN   ,LAPM   ,IMONTH , & !in
                LAT    ,IPOINT ,FVEG   ,                 & !in
                XLAI   ,XSAI   ,LFMASS ,RTMASS ,STMASS , & !inout
                FASTCP ,STBLCP ,WOOD   ,                 & !inout
                GPP    ,NPP    ,NEE    ,AUTORS ,HETERS , & !out
                TOTSC  ,TOTLB  )                           !out

!   CALL BVOC (VOCFLX,  VEGTYP,  VEGFAC,   APAR,   TV)
!   CALL CH4

  END SUBROUTINE CARBON
! ==================================================================================================
  SUBROUTINE CO2FLUX (NSNOW  ,NSOIL  ,VEGTYP ,IGS    ,DT     , & !in
                      DZSNSO ,STC    ,PSN    ,TROOT  ,TV     , & !in
                      WROOT  ,WSTRES ,FOLN   ,LAPM   ,IMONTH , & !in
                      LAT    ,IPOINT ,FVEG   ,                 & !in
                      XLAI   ,XSAI   ,LFMASS ,RTMASS ,STMASS , & !inout
                      FASTCP ,STBLCP ,WOOD   ,                 & !inout
                      GPP    ,NPP    ,NEE    ,AUTORS ,HETERS , & !out
                      TOTSC  ,TOTLB  )                           !out
! -----------------------------------------------------------------------------------------
! The original code is from RE Dickinson et al.(1998), modifed by Guo-Yue Niu, 2004
! -----------------------------------------------------------------------------------------
  USE VEG_PARAMETERS
! -----------------------------------------------------------------------------------------
  IMPLICIT NONE
! -----------------------------------------------------------------------------------------

! input

  INTEGER                          , INTENT(IN) :: IPOINT !grid index
  INTEGER                          , INTENT(IN) :: IMONTH !month index
  INTEGER                          , INTENT(IN) :: VEGTYP !vegetation physiology type
  INTEGER                          , INTENT(IN) :: NSNOW  !number of snow layers
  INTEGER                          , INTENT(IN) :: NSOIL  !number of soil layers
  REAL                             , INTENT(IN) :: DT     !time step (s)
  REAL                             , INTENT(IN) :: LAT    !latitude (radians)
  REAL                             , INTENT(IN) :: IGS    !growing season index (0=off, 1=on)
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: DZSNSO !snow/soil layer thickness [m]
  REAL, DIMENSION(-NSNOW+1:NSOIL), INTENT(IN) :: STC    !snow/soil temperature [k]
  REAL                             , INTENT(IN) :: PSN    !total leaf photosynthesis (umolco2/m2/s)
  REAL                             , INTENT(IN) :: TROOT  !root-zone averaged temperature (k)
  REAL                             , INTENT(IN) :: TV     !leaf temperature (k)
  REAL                             , INTENT(IN) :: WROOT  !root zone soil water
  REAL                             , INTENT(IN) :: WSTRES !soil water stress
  REAL                             , INTENT(IN) :: FOLN   !foliage nitrogen (%)
  REAL                             , INTENT(IN) :: LAPM   !leaf area per unit mass [m2/g]
  REAL                             , INTENT(IN) :: FVEG   !vegetation greenness fraction

! input and output

  REAL                          , INTENT(INOUT) :: XLAI   !leaf  area index from leaf carbon [-]
  REAL                          , INTENT(INOUT) :: XSAI   !stem area index from leaf carbon [-]
  REAL                          , INTENT(INOUT) :: LFMASS !leaf mass [g/m2]
  REAL                          , INTENT(INOUT) :: RTMASS !mass of fine roots [g/m2]
  REAL                          , INTENT(INOUT) :: STMASS !stem mass [g/m2]
  REAL                          , INTENT(INOUT) :: FASTCP !short lived carbon [g/m2]
  REAL                          , INTENT(INOUT) :: STBLCP !stable carbon pool [g/m2]
  REAL                          , INTENT(INOUT) :: WOOD   !mass of wood (incl. woody roots) [g/m2]

! output

  REAL                            , INTENT(OUT) :: GPP    !net instantaneous assimilation [g/m2/s]
```

```
  REAL                          , INTENT(OUT) :: NPP    !net primary productivity [g/m2]
  REAL                          , INTENT(OUT) :: NEE    !net ecosystem exchange (autors+heters-gpp)
  REAL                          , INTENT(OUT) :: AUTORS !net ecosystem resp. (maintance and growth)
  REAL                          , INTENT(OUT) :: HETERS !organic respiration
  REAL                          , INTENT(OUT) :: TOTSC  !total soil carbon (g/m2)
  REAL                          , INTENT(OUT) :: TOTLB  !total living carbon (g/m2)

! local

  REAL                        :: CFLUX    !carbon flux to atmosphere [g/m2/s]
  REAL                        :: LFMSMN   !minimum leaf mass [g/m2]
  REAL                        :: RSWOOD   !wood respiration [g/m2]
  REAL                        :: RSLEAF   !leaf maintenance respiration per timestep [g/m2]
  REAL                        :: RSROOT   !fine root respiration per time step [g/m2]
  REAL                        :: NPPL     !leaf net primary productivity [g/m2/s]
  REAL                        :: NPPR     !root net primary productivity [g/m2/s]
  REAL                        :: NPPW     !wood net primary productivity [g/m2/s]
  REAL                        :: NPPS     !wood net primary productivity [g/m2/s]
  REAL                        :: DIELF    !death of leaf mass per time step [g/m2]

  REAL                        :: ADDNPPLF !leaf assimil after resp. losses removed [g/m2]
  REAL                        :: ADDNPPST !stem assimil after resp. losses removed [g/m2]
  REAL                        :: CARBFX   !carbon assimilated per model step [g/m2]
  REAL                        :: GRLEAF   !growth respiration rate for leaf [g/m2/s]
  REAL                        :: GRROOT   !growth respiration rate for root [g/m2/s]
  REAL                        :: GRWOOD   !growth respiration rate for wood [g/m2/s]
  REAL                        :: GRSTEM   !growth respiration rate for stem [g/m2/s]
  REAL                        :: LEAFPT   !fraction of carbon allocated to leaves [-]
  REAL                        :: LFDEL    !maximum  leaf mass  available to change [g/m2/s]
  REAL                        :: LFTOVR   !stem turnover per time step [g/m2]
  REAL                        :: STTOVR   !stem turnover per time step [g/m2]
  REAL                        :: WDTOVR   !wood turnover per time step [g/m2]
  REAL                        :: RSSOIL   !soil respiration per time step [g/m2]
  REAL                        :: RTTOVR   !root carbon loss per time step by turnover [g/m2]
  REAL                        :: STABLC   !decay rate of fast carbon to slow carbon [g/m2/s]
  REAL                        :: WOODF    !calculated wood to root ratio [-]
  REAL                        :: NONLEF   !fraction of carbon to root and wood [-]
  REAL                        :: ROOTPT   !fraction of carbon flux to roots [-]
  REAL                        :: WOODPT   !fraction of carbon flux to wood [-]
  REAL                        :: STEMPT   !fraction of carbon flux to stem [-]
  REAL                        :: RESP     !leaf respiration [umol/m2/s]
  REAL                        :: RSSTEM   !stem respiration [g/m2/s]

  REAL                        :: FSW      !soil water factor for microbial respiration
  REAL                        :: FST      !soil temperature factor for microbial respiration
  REAL                        :: FNF      !foliage nitrogen adjustemt to respiration (<= 1)
  REAL                        :: TF       !temperature factor
  REAL                        :: RF       !respiration reduction factor (<= 1)
  REAL                        :: STDEL
  REAL                        :: STMSMN
  REAL                        :: SAPM     !stem area per unit mass (m2/g)
  REAL                        :: DIEST
! -------------------------- constants -------------------------------
  REAL                        :: BF       !parameter for present wood allocation [-]
  REAL                        :: RSWOODC  !wood respiration coeficient [1/s]
  REAL                        :: STOVRC   !stem turnover coefficient [1/s]
  REAL                        :: RSDRYC   !degree of drying that reduces soil respiration [-]
  REAL                        :: RTOVRC   !root turnover coefficient [1/s]
  REAL                        :: WSTRC    !water stress coeficient [-]
  REAL                        :: LAIMIN   !minimum leaf area index [m2/m2]
  REAL                        :: XSAMIN   !minimum leaf area index [m2/m2]
  REAL                        :: SC
  REAL                        :: SD
  REAL                        :: VEGFRAC

! Respiration as a function of temperature

  real :: r,x
          r(x) = exp(0.08*(x-298.16))
```

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

! constants
    RTOVRC  = 2.0E-8        !original was 2.0e-8
    RSDRYC  = 40.0          !original was 40.0
    RSWOODC = 3.0E-10       !
    BF      = 0.90          !original was 0.90   ! carbon to roots
    WSTRC   = 100.0
    LAIMIN  = 0.05
    XSAMIN  = 0.01

    SAPM    = 3.*0.001      ! m2/kg -->m2/g
    LFMSMN  = laimin/lapm
    STMSMN  = xsamin/sapm
! ----------------------------------------------------------------------------------

! respiration

     IF(IGS .EQ. 0.) THEN
       RF = 0.5
     ELSE
       RF = 1.0
     ENDIF

     FNF     = MIN( FOLN/MAX(1.E-06,FOLNMX(VEGTYP)), 1.0 )
     TF      = ARM(VEGTYP)**( (TV-298.16)/10. )
     RESP    = RMF25(VEGTYP) * TF * FNF * XLAI * RF * (1.-WSTRES) ! umol/m2/s
     RSLEAF  = MIN(LFMASS/DT,RESP*12.e-6)                         ! g/m2/s

     RSROOT  = RMR25(VEGTYP)*(RTMASS*1E-3)*TF *RF* 12.e-6         ! g/m2/s
     RSSTEM  = RMS25(VEGTYP)*(STMASS*1E-3)*TF *RF* 12.e-6         ! g/m2/s
     RSWOOD  = RSWOODC * R(TV) * WOOD*WDPOOL(VEGTYP)

! carbon assimilation
! 1 mole -> 12 g carbon or 44 g CO2; 1 umol -> 12.e-6 g carbon;

     CARBFX  = PSN * 12.e-6              ! umol co2 /m2/ s -> g/m2/s carbon

! fraction of carbon into leaf versus nonleaf

     LEAFPT = EXP(0.01*(1.-EXP(0.75*XLAI))*XLAI)
     IF(VEGTYP ==13) LEAFPT = EXP(0.01*(1.-EXP(0.50*XLAI))*XLAI)

     NONLEF = 1.0 - LEAFPT
     STEMPT = XLAI/10.0
     LEAFPT = LEAFPT - STEMPT

!  fraction of carbon into wood versus root

     IF(WOOD.GT.0) THEN
        WOODF = (1.-EXP(-BF*(WRRAT(VEGTYP)*RTMASS/WOOD))/BF)*WDPOOL(VEGTYP)
     ELSE
        WOODF = 0.
     ENDIF

     ROOTPT = NONLEF*(1.-WOODF)
     WOODPT = NONLEF*WOODF

! leaf and root turnover per time step

     LFTOVR = LTOVRC(VEGTYP)*1.E-6*LFMASS
     STTOVR = LTOVRC(VEGTYP)*1.E-6*STMASS
     RTTOVR = RTOVRC*RTMASS
     WDTOVR = 9.5E-10*WOOD

! seasonal leaf die rate dependent on temp and water stress
! water stress is set to 1 at permanent wilting point

     SC  = EXP(-0.3*MAX(0.,TV-TDLEF(VEGTYP))) * (LFMASS/120.)
```

```
      SD   = EXP((WSTRES-1.)*WSTRC)
      DIELF = LFMASS*1.E-6*(DILEFW(VEGTYP) * SD + DILEFC(VEGTYP)*SC)
      DIEST = STMASS*1.E-6*(DILEFW(VEGTYP) * SD + DILEFC(VEGTYP)*SC)

! calculate growth respiration for leaf, rtmass and wood

      GRLEAF = MAX(0.0,FRAGR(VEGTYP)*(LEAFPT*CARBFX - RSLEAF))
      GRSTEM = MAX(0.0,FRAGR(VEGTYP)*(STEMPT*CARBFX - RSSTEM))
      GRROOT = MAX(0.0,FRAGR(VEGTYP)*(ROOTPT*CARBFX - RSROOT))
      GRWOOD = MAX(0.0,FRAGR(VEGTYP)*(WOODPT*CARBFX - RSWOOD))

! Impose lower T limit for photosynthesis

      ADDNPPLF = MAX(0.,LEAFPT*CARBFX - GRLEAF-RSLEAF)
      ADDNPPST = MAX(0.,STEMPT*CARBFX - GRSTEM-RSSTEM)
      IF(TV.LT.TMIN(VEGTYP)) ADDNPPLF =0.
      IF(TV.LT.TMIN(VEGTYP)) ADDNPPST =0.

! update leaf, root, and wood carbon
! avoid reducing leaf mass below its minimum value but conserve mass

      LFDEL = (LFMASS - LFMSMN)/DT
      STDEL = (STMASS - STMSMN)/DT
      DIELF = MIN(DIELF,LFDEL+ADDNPPLF-LFTOVR)
      DIEST = MIN(DIEST,STDEL+ADDNPPST-STTOVR)

! net primary productivities

      NPPL   = MAX(ADDNPPLF,-LFDEL)
      NPPS   = MAX(ADDNPPST,-STDEL)
      NPPR   = ROOTPT*CARBFX - RSROOT - GRROOT
      NPPW   = WOODPT*CARBFX - RSWOOD - GRWOOD

! masses of plant components

      LFMASS = LFMASS + (NPPL-LFTOVR-DIELF)*DT
      STMASS = STMASS + (NPPS-STTOVR-DIEST)*DT   ! g/m2
      RTMASS = RTMASS + (NPPR-RTTOVR)      *DT

      IF(RTMASS.LT.0.0) THEN
            RTTOVR = NPPR
            RTMASS = 0.0
      ENDIF
      WOOD = (WOOD+(NPPW-WDTOVR)*DT)*WDPOOL(VEGTYP)

! soil carbon budgets

      FASTCP = FASTCP + (RTTOVR+LFTOVR+STTOVR+WDTOVR+DIELF)*DT

      FST = 2.0**( (STC(1)-283.16)/10. )
      FSW = WROOT / (0.20+WROOT) * 0.23 / (0.23+WROOT)
      RSSOIL = FSW * FST * MRP(VEGTYP)* MAX(0.,FASTCP*1.E-3)*12.E-6

      STABLC = 0.1*RSSOIL
      FASTCP = FASTCP - (RSSOIL + STABLC)*DT
      STBLCP = STBLCP + STABLC*DT

!  total carbon flux

      CFLUX  = - CARBFX + RSLEAF + RSROOT + RSWOOD + RSSTEM &
               + RSSOIL + GRLEAF + GRROOT + GRWOOD                ! g/m2/s

! for outputs

      GPP    = CARBFX                                            !g/m2/s C
      NPP    = NPPL + NPPW + NPPR                                !g/m2/s C
      AUTORS = RSROOT + RSWOOD  + RSLEAF +  &                    !g/m2/s C
               GRLEAF + GRROOT + GRWOOD                          !g/m2/s C
      HETERS = RSSOIL                                            !g/m2/s C
```

```
     NEE    = (AUTORS + HETERS - GPP)*44./12.                    !g/m2/s CO2
     TOTSC  = FASTCP + STBLCP                                    !g/m2   C
     TOTLB  = LFMASS + RTMASS + WOOD                             !g/m2   C

! leaf area index and stem area index

     XLAI     = MAX(LFMASS*LAPM,LAIMIN)
     XSAI     = MAX(STMASS*SAPM,XSAMIN)

  END SUBROUTINE CO2FLUX
! ==================================================================================================
! --------------------------------------------------------------------------------------------------
  SUBROUTINE BVOCFLUX(VOCFLX,   VEGTYP,   VEGFRAC,   APAR,    TV )
! --------------------------------------------------------------------------------------------------

! --------------------------------------------------------------------------------------------------
      implicit none
! --------------------------------------------------------------------------------------------------

! ------------------------ code history ---------------------------
! source file:       BVOC
! purpose:           BVOC emissions
! DESCRIPTION:
! Volatile organic compound emission
! This code simulates volatile organic compound emissions
! following the algorithm presented in Guenther, A., 1999: Modeling
! Biogenic Volatile Organic Compound Emissions to the Atmosphere. In
! Reactive Hydrocarbons in the Atmosphere, Ch. 3
! This model relies on the assumption that 90% of isoprene and monoterpene
! emissions originate from canopy foliage:
!    E = epsilon * gamma * density * delta
! The factor delta (longterm activity factor) applies to isoprene emission
! from deciduous plants only. We neglect this factor at the present time.
! This factor is discussed in Guenther (1997).
! Subroutine written to operate at the patch level.
! IN FINAL IMPLEMENTATION, REMEMBER:
! 1. may wish to call this routine only as freq. as rad. calculations
! 2. may wish to place epsilon values directly in pft-physiology file
! ------------------------ input/output variables -----------------
! input
  integer                     ,INTENT(IN) :: vegtyp  !vegetation type
  real                        ,INTENT(IN) :: vegfrac !green vegetation fraction [0.0-1.0]
  real                        ,INTENT(IN) :: apar    !photosynthesis active energy by canopy (w/m2)
  real                        ,INTENT(IN) :: tv      !vegetation canopy temperature (k)

! output
  real                        ,INTENT(OUT) :: vocflx(5) ! voc fluxes [ug C m-2 h-1]

! Local Variables

  real, parameter :: R      = 8.314    ! univ. gas constant [J K-1 mol-1]
  real, parameter :: alpha  = 0.0027   ! empirical coefficient
  real, parameter :: cl1    = 1.066    ! empirical coefficient
  real, parameter :: ct1    = 95000.0  ! empirical coefficient [J mol-1]
  real, parameter :: ct2    = 230000.0 ! empirical coefficient [J mol-1]
  real, parameter :: ct3    = 0.961    ! empirical coefficient
  real, parameter :: tm     = 314.0    ! empirical coefficient [K]
  real, parameter :: tstd   = 303.0    ! std temperature [K]
  real, parameter :: bet    = 0.09     ! beta empirical coefficient [K-1]

  integer ivoc        ! do-loop index
  integer ityp        ! do-loop index
  real epsilon(5)
  real gamma(5)
  real density
  real elai
  real slarea(27)
  real eps(27,5)
```

```
    real par,cl,reciprod,ct

    data (slarea(ityp),ityp=1,27) &
                /0.0228,0.0200,0.0200,0.0295,0.0223,0.0277,0.0060,0.0227,0.0188, &
                 0.0236,0.0258,0.0200,0.0200,0.0090,0.0223,0.0422,0.0390,10*0.02/ !??? Lindsey
    data (eps(ityp,1),ityp=1,27) &
                /41.87, 0.00, 0.00, 2.52, 0.04,17.11, 0.02,21.62, 0.11, &
                 22.80,46.86, 0.00, 0.00, 0.46,30.98, 2.31, 1.63, 10*0.0/  ! isoprene  24.0
    data (eps(ityp,2),ityp=1,27) &
                / 0.98, 0.00, 0.00, 0.16, 0.09, 0.28, 0.05, 0.92, 0.22, &
                  0.59, 0.38, 0.00, 0.00, 3.34, 0.96, 1.47, 1.07, 10*0.0/  ! monoterpenes 0.8
    data (eps(ityp,3),ityp=1,27) &
                / 1.82, 0.00, 0.00, 0.23, 0.05, 0.81, 0.03, 1.73, 1.26, &
                  1.37, 1.84, 0.00, 0.00, 1.85, 1.84, 1.70, 1.21, 10*0.0/  ! OVOC 1.0
    data (eps(ityp,4),ityp=1,27) /27*0.0/  ! ORVOC 1.0
    data (eps(ityp,5),ityp=1,27) /27*0.0/  ! CO    0.3

! epsilon :

    do ivoc = 1, 5
    epsilon(ivoc) = eps(VEGTYP,ivoc)
    end do

! gamma : Activity factor. Units [dimensionless]

      reciprod = 1. / (R * tv * tstd)
      ct = exp(ct1 * (tv - tstd) * reciprod) / &
           (ct3 + exp(ct2 * (tv - tm) * reciprod))

      par = apar * 4.6 ! (multiply w/m2 by 4.6 to get umol/m2/s)
      cl  = alpha * cl1 * par * (1. + alpha * alpha * par * par)**(-0.5)

   gamma(1) = cl * ct ! for isoprenes

   do ivoc = 2, 5
   gamma(ivoc) = exp(bet * (tv - tstd))
   end do

! Foliage density

! transform vegfrac to lai

   elai    = max(0.0,-6.5/2.5*alog((1.-vegfrac)))
   density = elai / (slarea(VEGTYP) * 0.5)

! calculate the voc flux

   do ivoc = 1, 5
   vocflx(ivoc) = epsilon(ivoc) * gamma(ivoc) * density
   end do

   end subroutine bvocflux
! ==================================================================================================
! ********************************* end of carbon subroutines *****************************
! ==================================================================================================
  SUBROUTINE REDPRM (VEGTYP,SOILTYP,SLOPETYP,SLDPTH,ZSOIL,NSOIL)
    use module_sf_noahlsm_param_init

    IMPLICIT NONE
! ----------------------------------------------------------------------
! Internally set (default valuess)
! all soil and vegetation parameters required for the execusion oF
! the Noah lsm are defined in VEGPARM.TBL, SOILPARM.TB, and GENPARM.TBL.
! ----------------------------------------------------------------------
!     Vegetation parameters:
!             CMXTBL: MAX CNPY Capacity
!              NROOT: Rooting depth
!
! ----------------------------------------------------------------------
```

```
!      Soil parameters:
!       SSATPSI: SAT (saturation) soil potential
!        SSATDW: SAT soil diffusivity
!           F1: Soil thermal diffusivity/conductivity coef.
!        QUARTZ: Soil quartz content
!  Modified by F. Chen (12/22/97)  to use the STATSGO soil map
!  Modified By F. Chen (01/22/00)  to include PLaya, Lava, and White San
!  Modified By F. Chen (08/05/02)  to include additional parameters for the Noah
! NOTE: SATDW = BB*SATDK*(SATPSI/MAXSMC)
!         F11 = ALOG10(SATPSI) + BB*ALOG10(MAXSMC) + 2.0
!       REFSMC1=MAXSMC*(5.79E-9/SATDK)**(1/(2*BB+3)) 5.79E-9 m/s= 0.5 mm
!       REFSMC=REFSMC1+1./3.(MAXSMC-REFSMC1)
!       WLTSMC1=MAXSMC*(200./SATPSI)**(-1./BB)    (Wetzel and Chang, 198
!       WLTSMC=WLTSMC1-0.5*WLTSMC1
! Note: the values for playa is set for it to have a thermal conductivit
! as sand and to have a hydrulic conductivity as clay
!
! ----------------------------------------------------------------------
! BLANK        OCEAN/SEA
!       CSOIL_DATA: soil heat capacity [J M-3 K-1]
!       ZBOT_DATA: depth[M] of lower boundary soil temperature
!       CZIL_DATA: calculate roughness length of heat
!       SMLOW_DATA and MHIGH_DATA: two soil moisture wilt, soil moisture referen
!                 parameters
! Set maximum number of soil- and veg- in data statement.
! ----------------------------------------------------------------------
    INTEGER, PARAMETER     :: MAX_SOILTYP=30,MAX_VEGTYP=30

! Veg parameters
    INTEGER, INTENT(IN)    :: VEGTYP
! Soil parameters
    INTEGER, INTENT(IN)    :: SOILTYP
! General parameters
    INTEGER, INTENT(IN)    :: SLOPETYP
! General parameters
    INTEGER, INTENT(IN)    :: NSOIL
! Layer parameters
    REAL,DIMENSION(NSOIL),INTENT(IN) :: SLDPTH
    REAL,DIMENSION(NSOIL),INTENT(IN) :: ZSOIL

    REAL                   :: FRZFACT
    INTEGER                :: I
! ----------------------------------------------------------------------
!
    IF (SOILTYP .gt. SLCATS) THEN
       WRITE (*,*) 'Warning: too many input soil types'
       print*, 'SOILTYP must be less than SLCATS:'
       print*, "    SOILTYP = ", SOILTYP
       print*, "    SLCATS  = ", SLCATS
       STOP 333
    END IF
    IF (VEGTYP .gt. LUCATS) THEN
       WRITE (*,*) 'Warning: too many input landuse types'
       STOP 333
    END IF

! ----------------------------------------------------------------------
!  SET-UP SOIL PARAMETERS
! ----------------------------------------------------------------------
    CSOIL  = CSOIL_DATA
    BEXP   = BB (SOILTYP)
    DKSAT  = SATDK (SOILTYP)
    DWSAT  = SATDW (SOILTYP)
    F1     = F11 (SOILTYP)
    PSISAT = SATPSI (SOILTYP)
    QUARTZ = QTZ (SOILTYP)
    SMCDRY = DRYSMC (SOILTYP)
    SMCMAX = MAXSMC (SOILTYP)
    SMCREF = REFSMC (SOILTYP)
```

```
    SMCWLT = WLTSMC (SOILTYP)

! ----------------------------------------------------------------------
! Set-up universal parameters (not dependent on SOILTYP, VEGTYP)
! ----------------------------------------------------------------------
    ZBOT   = ZBOT_DATA
    CZIL   = CZIL_DATA

    FRZK   = FRZK_DATA
    REFDK  = REFDK_DATA
    REFKDT = REFKDT_DATA
    KDT    = REFKDT * DKSAT / REFDK
    SLOPE  = SLOPE_DATA (SLOPETYP)

! adjust FRZK parameter to actual soil type: FRZK * FRZFACT

    if(SOILTYP /= 14) then
      FRZFACT = (SMCMAX / SMCREF) * (0.412 / 0.468)
      FRZX = FRZK * FRZFACT
    end if

!    write(*,*) FRZK, FRZX, KDT, SLOPE, SLOPETYP
! ----------------------------------------------------------------------
! SET-UP VEGETATION PARAMETERS
! ----------------------------------------------------------------------
    ! Six redprm_canres variables:
    TOPT = TOPT_DATA
    RGL = RGLTBL (VEGTYP)
    RSMAX = RSMAX_DATA
    RSMIN = RSTBL (VEGTYP)
    HS = HSTBL (VEGTYP)
    NROOT = NROTBL (VEGTYP)

!    SHDFAC = SHDTBL(VEGTYP)
!    IF (VEGTYP .eq. BARE) SHDFAC = 0.0

    IF (NROOT .gt. NSOIL) THEN
       WRITE (*,*) 'Warning: too many root layers'
       write (*,*) 'NROOT = ', nroot
       write (*,*) 'NSOIL = ', nsoil
       STOP 333
    END IF

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

  END  SUBROUTINE REDPRM
! ==================================================================================================
  SUBROUTINE LSMZEN (CALDAY, LOND, LATD, COSZ)

! cosine solar zenith angle from:
!    o day (1.x to 365.x), where x=0 (e.g. 213.0) denotes 00:00 at greenwich
!    o latitude,  where SH = - and NH = +
!    o longitude, where WH = - and EH = +
! the solar declination must match that used in the atmospheric model.
! -----------------------------------------------------------------
! input
     REAL, INTENT(IN) :: CALDAY  !calendar day + fraction (1.xx -> 365.xx)
     REAL, INTENT(IN) :: LATD    !latitude  (degree): + = NH ( -90 ->  90)
     REAL, INTENT(IN) :: LOND    !longitude (degree): + = EH (-180 -> 180)

! output
     REAL,INTENT(OUT) :: COSZ    !cosine zenith angle

! local
     REAL :: DAYSPY              !days per year
     REAL :: PI                  !pi
     REAL :: THETA               !earth orbit seasonal angle in radians
     REAL :: DELTA               !solar declination angle  in radians
     REAL :: SIND                !sine   of declination
```

```
      REAL :: COSD                 !cosine of declination
      REAL :: PHI                  !greenwich calendar day + longitude offset
      REAL :: LOCTIM               !local time (hour)
      REAL :: HRANG                !solar hour angle, 24 hour periodicity (radians)
      REAL :: LAT                  !latitude  (radians): + = NH
      REAL :: LON                  !longitude (radians): + = EH

      INTEGER MCSEC             !current seconds in day (0, ..., 86400)
! -----------------------------------------------------------------

      DAYSPY = 365.
      PI = 4.*ATAN(1.)

      LAT = LATD * PI/180.
      LON = LOND * PI/180

! solar declination:

      THETA = (2.*PI*CALDAY)/DAYSPY
      DELTA = .006918 - .399912*COS(   THETA) + .070257*SIN(   THETA) &
                      - .006758*COS(2.*THETA) + .000907*SIN(2.*THETA) &
                      - .002697*COS(3.*THETA) + .001480*SIN(3.*THETA)
      SIND = SIN(DELTA)
      COSD = COS(DELTA)

! local time

         MCSEC = (CALDAY - INT(CALDAY)) * 86400.
         PHI = CALDAY + LON/(2.*PI)
         LOCTIM = (MCSEC + (PHI-CALDAY)*86400.) / 3600.

! hour angle

         HRANG = 360./24. * (LOCTIM-12.) * PI/180.

! cosine solar zenith angle. reset points with sun slightly below horizon
! to slightly above horizon, as discussed in notes.

         COSZ = SIN(LAT)*SIND + COS(LAT)*COSD*COS(HRANG)
         IF (COSZ .GE. -0.001 .AND. COSZ.LE. 0.) COSZ = 0.001

  END SUBROUTINE LSMZEN
! ==================================================================================================
  SUBROUTINE CALENDR(ISTEP   ,DTIME   ,IMONTH, IDAY, ITIME, CALDAY)

! ------------------------------------------------------------------------
! input

  INTEGER, INTENT(IN) :: ISTEP
  INTEGER, INTENT(IN) :: IMONTH
  INTEGER, INTENT(IN) :: IDAY
  INTEGER, INTENT(IN) :: ITIME
  REAL,    INTENT(IN) :: DTIME

! output

  REAL,   INTENT(OUT) :: CALDAY

! local
  INTEGER :: NDAY(12)     ! number of days per month
  INTEGER :: JDAY(12)     ! convert month index to julian day

   DATA NDAY/31,28,31,30,31,30,31,31,30,31,30,31/
   DATA JDAY/0,31,59,90,120,151,181,212,243,273,304,334/
! ------------------------------------------------------------------------

    CALDAY = JDAY(IMONTH) + IDAY + ITIME/24.
! ------------------------------------------------------------------------
  END SUBROUTINE CALENDR
```

```
! ==================================================================================================
END MODULE NOAHLSM_ROUTINES
! ==================================================================================================

MODULE MODULE_SF_NOAHLSM

  USE NOAHLSM_ROUTINES
  USE NOAHLSM_GLOBALS

END MODULE MODULE_SF_NOAHLSM
```

```
MODULE module_sf_Noahlsm_gridded_input

  contains

!-------------------------------------------------------------
        SUBROUTINE READLAND(DIR,IX,JX,VEGTYP,SOLTYP,SLOPE,LAT,LON,LANDSEA,tbot,ISC)

    IMPLICIT NONE

        CHARACTER(len=256) :: DIR
        INTEGER, INTENT(IN) :: IX,JX
        INTEGER :: I,J

        INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: VEGTYP
        INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: SOLTYP
        INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: SLOPE
        INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: LANDSEA
        INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: ISC      !soil color index
        REAL,    INTENT(OUT), DIMENSION(IX,JX) :: LAT
        REAL,    INTENT(OUT), DIMENSION(IX,JX) :: LON
        REAL,    INTENT(OUT), DIMENSION(IX,JX) :: TBOT

        INCLUDE 'netcdf.inc'

        INTEGER STATUS
        INTEGER NCID
        INTEGER START(2), COUNT(2)
        DATA START /1,  1/
        INTEGER lonID,latID,landID,plotID,vegeID,soilID,tbotID,iscID

        COUNT(1) = IX
        COUNT(2) = JX

        STATUS=NF_OPEN(TRIM(DIR)//'/static/lon_lat.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'lat2d', latID)
        STATUS=NF_INQ_VARID (NCID, 'lon2d', lonID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_real(NCID,latID,START,COUNT,LAT)
        STATUS=NF_GET_VARA_real(NCID,lonID,START,COUNT,LON)
        STATUS=NF_CLOSE(NCID)

        STATUS=NF_OPEN(TRIM(DIR)//'/static/landmask.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'landmask', landID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_INT(NCID,landID,START,COUNT,LANDSEA)
        STATUS=NF_CLOSE(NCID)

        STATUS=NF_OPEN(TRIM(DIR)//'/static/soilcolor.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'SC', iscID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_INT(NCID,iscID,START,COUNT,isc)
        STATUS=NF_CLOSE(NCID)

        STATUS=NF_OPEN(TRIM(DIR)//'/static/veg_soil.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'VEG2D', vegeID)
!       STATUS=NF_INQ_VARID (NCID, 'TOPSOIL2D', soilID)
        STATUS=NF_INQ_VARID (NCID, 'BOTSOIL2D', soilID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_INT(NCID,vegeID,START,COUNT,VEGTYP)
        STATUS=NF_GET_VARA_INT(NCID,soilID,START,COUNT,SOLTYP)
        STATUS=NF_CLOSE(NCID)

        STATUS=NF_OPEN(TRIM(DIR)//'/static/tbot.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'TBOT', tbotID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_real(NCID,tbotID,START,COUNT,tbot)
        STATUS=NF_CLOSE(NCID)

        OPEN(10,FILE='surface_datalog.dat')
```

```
        write(10,*)'      longitue    latitude    landmask   vegetype   soiltype     isc    tbot'
        DO J=1,JX
        DO I=1,IX
          if(soltyp(i,j) == 14) vegtyp(i,j) = 16
          SLOPE(I,J)    = 3      !only used in runoff option 3
          write(10,10)LON(I,J),LAT(I,J),LANDSEA(I,J),VEGTYP(I,J),SOLTYP(I,J),ISC(I,J),TBOT(I,J)
        ENDDO
        ENDDO
 10     FORMAT(2X,2F10.2,4I10,F10.2)
        write(6,*) '------------- successful reading surface data ------------------'
! -----------------------------------------------------------------------
  END SUBROUTINE READLAND
! -----------------------------------------------------------------------
  SUBROUTINE READINIT(dir      ,nx       ,ny       ,nsoil    ,nsnow    ,fini      , &
                    soiltypxy,nsltype ,maxsmc   ,zsoil    ,alboldxy,             &
                    fwetxy   ,sneqvoxy,qsnowxy  ,wslakexy,eahxy    ,tahxy     , &
                    smcxy    ,stcxy    ,sh2oxy   ,tsnoxy   ,snicexy ,snliqxy  , &
                    zsnsoxy  ,isnowxy  ,snowhxy  ,sneqvxy  ,canliqxy,canicexy, &
                    tgxy     ,tvxy     ,waxy     ,wtxy     ,zwtxy    ,lfmassxy, &
                    rtmassxy ,stmassxy,woodxy   ,stblcpxy,fastcpxy,xlaixy   , &
                    xsaixy    )

    implicit none

        character(len=256)  :: dir
        character(len=256)  :: fini
        integer, intent(in) :: nx,ny
        integer, intent(in) :: nsoil
        integer, intent(in) :: nsnow
        integer, intent(in) :: nsltype
        real,    intent(in) :: maxsmc(nsltype)
        real,    intent(in) :: zsoil(nsoil)
        integer, intent(in), dimension(nx,ny)      :: soiltypxy

        integer :: i,ix,iy,iz

        real, intent(out), dimension(nx,ny)                :: fwetxy
        real, intent(out), dimension(nx,ny)                :: sneqvoxy
        real, intent(out), dimension(nx,ny)                :: alboldxy
        real, intent(out), dimension(nx,ny)                :: qsnowxy
        real, intent(out), dimension(nx,ny)                :: wslakexy
        real, intent(out), dimension(nx,ny)                :: eahxy
        real, intent(out), dimension(nx,ny)                :: tahxy
        real, intent(out), dimension(nx,ny,        1:nsoil) :: smcxy    ! 1
        real, intent(out), dimension(nx,ny,        1:nsoil) :: stcxy    ! 2
        real, intent(out), dimension(nx,ny,        1:nsoil) :: sh2oxy   ! 3
        real, intent(out), dimension(nx,ny,-nsnow+1:     0) :: tsnoxY   ! 4
        real, intent(out), dimension(nx,ny,-nsnow+1:     0) :: snicexy  ! 5
        real, intent(out), dimension(nx,ny,-nsnow+1:     0) :: snliqxy  ! 6
        real, intent(out), dimension(nx,ny,-nsnow+1:nsoil) :: zsnsoxy  ! 7
        real, intent(out), dimension(nx,ny)                :: tvxy     ! 8
        real, intent(out), dimension(nx,ny)                :: tgxy     ! 9
        real, intent(out), dimension(nx,ny)                :: canliqxy !10
        real, intent(out), dimension(nx,ny)                :: canicexy !11
        real, intent(out), dimension(nx,ny)                :: snowhxy  !12
        real, intent(out), dimension(nx,ny)                :: sneqvxy  !13
        real, intent(out), dimension(nx,ny)                :: waxy     !14
        real, intent(out), dimension(nx,ny)                :: wtxy     !15
        real, intent(out), dimension(nx,ny)                :: zwtxy    !16
        integer, intent(out), dimension(nx,ny)             :: isnowxy  !17
        real, intent(out), dimension(nx,ny)                :: lfmassxy !18
        real, intent(out), dimension(nx,ny)                :: rtmassxy !19
        real, intent(out), dimension(nx,ny)                :: stmassxy !20
        real, intent(out), dimension(nx,ny)                :: woodxy   !21
        real, intent(out), dimension(nx,ny)                :: stblcpxy !22
        real, intent(out), dimension(nx,ny)                :: fastcpxy !23
        real, intent(out), dimension(nx,ny)                :: xlaixy !24
        real, intent(out), dimension(nx,ny)                :: xsaixy !25
```

```
        logical readini                   !true if read in initial data set

        write(*,*) NSLTYPE
        write(*,*)  (MAXSMC(i),i=1,nsltype)
        write(*,*) 'fini=', TRIM(fini)

        if (fini == 'arbitrary initialization') then
           readini  = .false.
        else
           readini  = .true.
        end if

!arbitary values

        fwetxy(:,:)           = 0.0     ! wetted fraction of canopy
        sneqvoxy(:,:)         = 0.0     ! snow water equivalent at last time step
        alboldxy(:,:)         = 0.65    ! snow surface albedo at last time step (CLASS type)
        qsnowxy(:,:)          = 0.0     ! snowfall on the ground through the canopy
        wslakexy(:,:)         = 0.0     ! lake water storage
        xsaixy(:,:)           = 0.1     ! stem area index
        xlaixy(:,:)           = 0.1     ! leaf area index
        eahxy(:,:)            = 2000.   ! water vapor pressure within the canopy air
        tahxy(:,:)            = 287.    ! temperature within the canopy air

!read from a file

        if (readini) then
            open(100,file =fini, status = 'old')
            do ix = 1,nx
            do iy = 1,ny
               read(100,'(1x,3i5)') i, i, isnowxy(ix,iy)
               read(100,*)(smcxy(ix,iy,iz),iz=1,nsoil),&
                             (stcxy(ix,iy,iz),iz=1,nsoil),&
                             (sh2oxy(ix,iy,iz),iz=1,nsoil)
               if(isnowxy(ix,iy) .lt. 0) then
               read(100,*)(tsnoxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0),&
                             (snicexy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0),&
                             (snliqxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0)
               end if
               read(100,*)(zsnsoxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,nsoil)
               read(100,*)tvxy(ix,iy),tgxy(ix,iy),canicexy(ix,iy),canliqxy(ix,iy),&
                              snowhxy(ix,iy),sneqvxy(ix,iy),waxy(ix,iy),wtxy(ix,iy),&
                              zwtxy(ix,iy)
               read(100,*) lfmassxy(ix,iy), rtmassxy(ix,iy),stmassxy(ix,iy), &
                            woodxy(ix,iy)  , stblcpxy(ix,iy),fastcpxy(ix,iy)

            end do
            end do

            close(100)
        else

!arbitary values

            do ix=1,nx
            do iy=1,ny
                do iz = 1,nsoil
                  smcxy (ix,iy,iz)=min(0.4,0.6*maxsmc(soiltypxy(ix,iy)) )
                  sh2oxy(ix,iy,iz)=min(0.4,0.6*maxsmc(soiltypxy(ix,iy)) )
                end do

                stcxy(ix,iy,1)      = 274.0
                stcxy(ix,iy,2)      = 278.0
                stcxy(ix,iy,3)      = 280.0
                stcxy(ix,iy,4)      = 284.0

                canliqxy(ix,iy)     = 0.0
                canicexy(ix,iy)     = 0.0
```

```
                tvxy(ix,iy)         = 287.0
                tgxy(ix,iy)         = 287.0
                snowhxy(ix,iy)      = 0.0
                sneqvxy(ix,iy)      = 0.0

                waxy(ix,iy)         = 4900.
                wtxy(ix,iy)         = 4900.
                zwtxy(ix,iy)        = (25.  + 2.0) - waxy(ix,iy)/1000/0.2

                lfmassxy(ix,iy)     = 50.0      !
                stmassxy(ix,iy)     = 50.0
                fastcpxy(ix,iy)     = 500.0     !g/m2
                rtmassxy(ix,iy)     = 500.0     !g/m2
                woodxy(ix,iy)       = 1000.00
                stblcpxy(ix,iy)     = 1000.00   !g/m2

            enddo
            enddo

            ! 5 outputs
            call snow_init (nx     ,ny     ,nsnow  ,nsoil   ,zsoil  , &
                            snowhxy,zsnsoxy,tsnoxy ,snicexy,snliqxy , &
                            isnowxy)
        end if

        write(6,*) '------------- successful initialization ------------------'

! --------------------------------------------------------
  END SUBROUTINE READINIT
! ------------------------------------------------------------------------------------------
  SUBROUTINE SNOW_INIT (IX     ,JX     ,NSNOW  ,NSOIL   ,ZSOIL  ,SNODEP  ,   &
                        ZSNSOXY,TSNOXY ,SNICEXY,SNLIQXY ,ISNOWXY)

! ------------------------------------------------------------------------------------------
   IMPLICIT NONE
! ------------------------------------------------------------------------------------------
   INTEGER, INTENT(IN) :: IX,JX,NSNOW,NSOIL
   REAL,    INTENT(IN), DIMENSION(IX,JX) :: SNODEp
   REAL, DIMENSION(1:NSOIL) :: ZSOIL
   INTEGER :: I,J,IZ

   INTEGER, INTENT(OUT), DIMENSION(IX,JX) :: ISNOWXY
   REAL,    INTENT(OUT), DIMENSION(IX,JX,-NSNOW+1:NSOIL) :: ZSNSOXY
   REAL,    INTENT(OUT), DIMENSION(IX,JX,-NSNOW+1:0) :: TSNOXY
   REAL,    INTENT(OUT), DIMENSION(IX,JX,-NSNOW+1:0) :: SNICEXY
   REAL,    INTENT(OUT), DIMENSION(IX,JX,-NSNOW+1:0) :: SNLIQXY

!local
   REAL,    DIMENSION(IX,JX,-NSNOW+1:    0) :: DZSNOXY
   REAL,    DIMENSION(IX,JX,-NSNOW+1:NSOIL) :: DZSNSOXY
! ------------------------------------------------------------------------------------------


   DO I = 1, IX
   DO J = 1, JX
    IF (SNODEP(I,J) < 0.025) THEN
          ISNOWXY(I,J) = 0
          DZSNOXY(I,J,-NSNOW+1:0) = 0.
       ELSE
          IF ((SNODEP(I,J) >= 0.025) .AND. (SNODEP(I,J) <= 0.05)) THEN
             ISNOWXY(I,J)     = -1
             DZSNOXY(I,J,0)   = SNODEP(I,J)
          ELSE IF ((SNODEP(I,J) > 0.05) .AND. (SNODEP(I,J) <= 0.10)) THEN
             ISNOWXY(I,J)     = -2
             DZSNOXY(I,J,-1)  = SNODEP(I,J)/2.
             DZSNOXY(I,J, 0)  = SNODEP(I,J)/2.
          ELSE IF ((SNODEP(I,J) > 0.10) .AND. (SNODEP(I,J) <= 0.25)) THEN
             ISNOWXY(I,J)     = -2
             DZSNOXY(I,J,-1)  = 0.05
```

```
              DZSNOXY(I,J, 0) = SNODEP(I,J) - DZSNOXY(I,J,-1)
           ELSE IF ((SNODEP(I,J) > 0.25) .AND. (SNODEP(I,J) <= 0.35)) THEN
              ISNOWXY(I,J)     = -3
              DZSNOXY(I,J,-2) = 0.05
              DZSNOXY(I,J,-1) = 0.5*(SNODEP(I,J)-DZSNOXY(I,J,-2))
              DZSNOXY(I,J, 0) = 0.5*(SNODEP(I,J)-DZSNOXY(I,J,-2))
           ELSE IF (SNODEP(I,J) > 0.35) THEN
              ISNOWXY(I,J)     = -3
              DZSNOXY(I,J,-2) = 0.05
              DZSNOXY(I,J,-1) = 0.10
              DZSNOXY(I,J, 0) = SNODEP(I,J) - DZSNOXY(I,J,-1) - DZSNOXY(I,J,-2)
           END IF
    END IF
  ENDDO
  ENDDO

  DO I = 1, IX
  DO J = 1, JX
       TSNOXY( I,J,-NSNOW+1:0) = 0.
       SNICEXY(I,J,-NSNOW+1:0) = 0.
       SNLIQXY(I,J,-NSNOW+1:0) = 0.
      DO IZ = ISNOWXY(I,j)+1, 0
         TSNOXY(I,J,IZ)  = 270.   ! [k]
         SNLIQXY(I,J,IZ) = 0.00 * DZSNOXY(I,J,IZ) * 200.
         SNICEXY(I,J,IZ) = 1.00 * DZSNOXY(I,J,IZ) * 200.   ! [mm or kg/m2]
      END DO

       DO IZ = ISNOWXY(I,J)+1, 0
            DZSNSOXY(I,J,IZ) = -DZSNOXY(I,J,IZ)
       END DO

       DZSNSOXY(I,J,1) = ZSOIL(1)
       DO IZ = 2,NSOIL
            DZSNSOXY(I,J,IZ) = (ZSOIL(IZ) - ZSOIL(IZ-1))
       END DO

       ZSNSOXY(I,J,ISNOWXY(I,J)+1) = DZSNSOXY(I,J,ISNOWXY(I,J)+1)
       DO IZ = ISNOWXY(I,J)+2 ,NSOIL
           ZSNSOXY(I,J,IZ) = ZSNSOXY(I,J,IZ-1) + DZSNSOXY(I,J,IZ)
       ENDDO

  END DO
  END DO

!  DO IZ = -NSNOW+1, NSOIL
!  WRITE(*,'(I10,4F10.3)')  IZ,ZSNSOXY(1,1,IZ),TSNOXY(1,1,IZ),SNICEXY(1,1,IZ),SNLIQXY(1,1,IZ)
!  END DO

  END SUBROUTINE SNOW_INIT
! ==================================================================================================
  SUBROUTINE READFORC(IX,JX,DT,IYEAR,IMON,IDAY,IHOUR,DIR,TMP2M,QAIR,DLWRF,U,V,    &
          PRES,DSWRF,PRCP)

        IMPLICIT NONE

        CHARACTER(len=256),INTENT(IN) :: DIR

        INTEGER, INTENT(IN) :: IX,JX
        INTEGER, INTENT(IN) :: iyear,imon,iday,ihour
        REAL, INTENT(IN) :: DT
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: DLWRF
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: DSWRF
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: PRCP
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: TMP2M
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: U
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: V
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: QAIR
        REAL, INTENT(OUT), DIMENSION(IX,JX) :: PRES
```

```
        INTEGER :: I,J,ivar
        INTEGER, PARAMETER :: nvar = 9
        REAL, allocatable, dimension(:,:) :: var
        REAL, DIMENSION(IX,JX,1:nvar) :: varin
        real :: ea, eair

        INCLUDE 'netcdf.inc'

        CHARACTER(len=256) :: ncfile
!        CHARACTER(len=4) ::filename(nvar)
        CHARACTER(len=12) ::varname(nvar)
        INTEGER STATUS
        INTEGER NCIDin
        INTEGER varID
        INTEGER START(2), COUNT(2)
        DATA START /1, 1/

        data (varname(ivar), ivar=1,nvar) &
                      /'T2D','Q2D','U2D','V2D','PSFC',&
                       'RAIN_NLDAS','RAIN_NEXRAD','SWDOWN','LWDOWN'  /

        allocate( var (ix,jx) )

        COUNT(1) = IX
        COUNT(2) = JX

        do I=1,IX
        do J=1,JX
             var(i,j)      = -999.
             DSWRF(I,J)    = -999.
             DLWRF(I,J)    = -999.
             PRCP(I,J)     = -999.
             U(I,J)        = -999.
             V(I,J)        = -999.
             TMP2M(I,J)    = -999.
             QAIR(I,J)     = -999.
             PRES(I,J)     = -999.
        end do
        end do


        write(*,*) '-------------------------------------------'
        write(ncfile,100) iyear,imon,iday,ihour
         if(imon < 10) then
           if(iday < 10) then
              if(ihour < 10) then
                  write(ncfile,100) iyear,imon,iday,ihour
 100              format('/forcings/',i4,'0',i1,'0',i1,'0',i1,'.NLDAS.nc')
              else
                  write(ncfile,200) iyear,imon,iday,ihour
 200              format('/forcings/',i4,'0',i1,'0',i1,i2,'.NLDAS.nc')
              end if
           else
              if(ihour < 10) then
                  write(ncfile,300) iyear,imon,iday,ihour
 300              format('/forcings/',i4,'0',i1,i2,'0',i1,'.NLDAS.nc')
              else
                  write(ncfile,400) iyear,imon,iday,ihour
 400              format('/forcings/',i4,'0',i1,i2,i2,'.NLDAS.nc')
              end if
           end if
         else
           if(iday < 10) then
              if(ihour < 10) then
                  write(ncfile,500) iyear,imon,iday,ihour
 500              format('/forcings/',i4,i2,'0',i1,'0',i1,'.NLDAS.nc')
              else
                  write(ncfile,600) iyear,imon,iday,ihour
 600              format('/forcings/',i4,i2,'0',i1,i2,'.NLDAS.nc')
```

```fortran
            end if
         else
            if(ihour < 10) then
                write(ncfile,700) iyear,imon,iday,ihour
  700           format('/forcings/',i4,i2,i2,'0',i1,'.NLDAS.nc')
            else
                write(ncfile,800) iyear,imon,iday,ihour
  800           format('/forcings/',i4,i2,i2,i2,'.NLDAS.nc')
            end if
         end if
      end if

      write(*,*) 'READFORC: opening ',trim(ncfile)
      ncfile = TRIM(DIR)//ncfile
      STATUS=NF_OPEN(ncfile,NF_NOWRITE,NCIDin)
      write(*,*) '-------------------------------------------'

      do ivar = 1,  nvar
!      write(*,*) varname(ivar)
         STATUS=NF_INQ_VARID (NCIDin, varname(ivar), varID)
         STATUS=NF_GET_VARA_real(NCIDin,varID,START,COUNT,var)
!         write(*,10) ((var(i,j),i=IX/2,IX/2+10),j=JX/2,JX/2+10)
!         write(*,10) ((var(i,j),i=317,321),j=4,4)
         do I=1,IX
         do J=1,JX
         varin(i,j,ivar) = var(i,j)
         end do
         end do
      end do
      STATUS=NF_CLOSE(NCIDin)

      do I=1,IX
      do J=1,JX
            TMP2M(I,J)     = varin(i,j,1)
            QAIR(I,J)      = varin(i,j,2)
            U(I,J)         = sqrt(varin(i,j,3)**2 + varin(i,j,4)**2)
            V(I,J)         = 0.
            PRES(I,J)      = varin(i,j,5)
            PRCP(I,J)      = varin(i,j,6) / DT          ! NLDAS precipitation

            if(varin(i,j,7).gt.0. .and. varin(i,j,7).ne.5.0 .and. &
            varin(i,j,7).lt.1000.) then
                 PRCP(I,J)      = varin(i,j,7) / DT     ! NEXRAD precipitation
            end if

            DSWRF(I,J)     = varin(i,j,8)
            DLWRF(I,J)     = varin(i,j,9)

            IF(DLWRF(I,J) >= 500. ) THEN
               EAIR    = QAIR(I,J)*PRES(I,J) / (0.622+0.378*QAIR(I,J))
               ea      = 0.70 + 5.95e-05 * 0.01*EAIR*exp(1500.0/TMP2M(I,J))
               DLWRF(I,J) = ea * 5.67e-08 *TMP2M(I,J)**4
            end if
      end do
      end do

  10 format(1x,5f15.5)

     return
! -----------------------------------------------------
  END SUBROUTINE READFORC
! -----------------------------------------------------
! ===================================================================================================
  SUBROUTINE READVEG(DIR,IX,JX,LANDSEA,SHDFAC)

      CHARACTER(len=256)  :: DIR
      INTEGER, INTENT(IN) :: IX,JX
      INTEGER, INTENT(IN), DIMENSION(IX,JX) :: LANDSEA
      REAL, INTENT(OUT), DIMENSION(IX,JX,12) :: SHDFAC
```

```
        INTEGER :: I,J,IM

        INCLUDE 'netcdf.inc'

        INTEGER STATUS,NCID
        INTEGER START(3), COUNT(3)
        DATA START /1,  1,  1/
        INTEGER gvfID

        COUNT(1) = IX
        COUNT(2) = JX
        COUNT(3) = 12

        STATUS=NF_OPEN(TRIM(DIR)//'/static/gvf.nc',NF_NOWRITE,NCID)
        STATUS=NF_INQ_VARID (NCID, 'GVF', gvfID)
        status=NF_ENDDEF(ncid)
        STATUS=NF_GET_VARA_real(NCID,gvfID,START,COUNT,SHDFAC)
        STATUS=NF_CLOSE(NCID)

        open(30,file='shdfac_chk.dat',status='unknown')

        do i = 1,ix
        do j = 1,jx
          write(30,'(3I5,12F8.2)')i,j,landsea(i,j),(shdfac(i,j,im),im=1,12)
        end do
        end do
! -------------------------------------------------------
  END SUBROUTINE READVEG
! -------------------------------------------------------

END MODULE module_sf_Noahlsm_gridded_input
```

```
MODULE module_sf_Noah_NC_output

  contains

!----------------------------------------------------------------------
    SUBROUTINE NC_OUT(nsoil    ,nx      ,ny      ,it      ,dt      ,   &
                      iyear    ,imonth  ,iday    ,DIR     ,EXP, lonxy    ,  &
                      latxy    ,vegtypxy,imstep  ,nday    ,ND      ,  &
                      snowhxy  ,sneqvxy ,tgxy    ,stcxy   ,smcxy   ,   &
                      sh2oxy   ,prcpxy  ,runsfxy ,runsbxy ,ecanxy  ,   &
                      edirxy   ,etranxy ,zwtxy   ,fsaxy   ,firaxy  ,   &
                      fshxy    ,flhxy   ,fghxy   ,aparxy  ,psnxy   ,   &
                      savxy    ,sagxy   ,fsnoxy  ,xlaixy  ,xsaixy  ,   &
                      tradxy   ,neexy   ,gppxy   ,nppxy   ,tsxy    ,   &
                      fvegxy   ,cmxy    ,chxy    ,                     &
                      snowhm   ,sneqvm  ,tgm     ,stcm    ,smcm    ,   &
                      sh2om    ,prcpm   ,runsfm  ,runsbm  ,ecanm   ,   &
                      edirm    ,etranm  ,zwtm    ,fsam    ,firam   ,   &
                      fshm     ,flhm    ,fghm    ,aparm   ,psnm    ,   &
                      savm     ,sagm    ,fsnom   ,xlaim   ,xsaim   ,   &
                      tradm    ,neem    ,gppm    ,nppm    ,tsm     ,   &
                      fvegm    ,cmm     ,chm     )
    implicit none

! inputs
    integer, intent(in) :: iyear,imonth,iday
    CHARACTER(len=256),INTENT(IN)  :: DIR
    CHARACTER(len=8),INTENT(IN)  :: EXP
    integer, intent(in)          :: it
    integer, intent(in)          :: imstep
    integer, intent(in)          :: nx
    integer, intent(in)          :: ny
    integer, intent(in)          :: nsoil
    integer, intent(in)          :: nday(12)
    real, intent(in)             :: dt
    integer, dimension(1:nx,1:ny), intent(in)      :: vegtypxy
    real, dimension(1:nx,1:ny), intent(in)         :: lonxy
    real, dimension(1:nx,1:ny), intent(in)         :: latxy
    real, dimension(1:nx,1:ny), intent(in)         :: snowhxy
    real, dimension(1:nx,1:ny), intent(in)         :: sneqvxy
    real, dimension(1:nx,1:ny), intent(in)         :: tgxy
    real, dimension(1:nx,1:ny), intent(in)         :: prcpxy
    real, dimension(1:nx,1:ny), intent(in)         :: runsfxy
    real, dimension(1:nx,1:ny), intent(in)         :: runsbxy
    real, dimension(1:nx,1:ny), intent(in)         :: ecanxy
    real, dimension(1:nx,1:ny), intent(in)         :: edirxy
    real, dimension(1:nx,1:ny), intent(in)         :: etranxy
    real, dimension(1:nx,1:ny), intent(in)         :: zwtxy
    real, dimension(1:nx,1:ny), intent(in)         :: fsaxy
    real, dimension(1:nx,1:ny), intent(in)         :: firaxy
    real, dimension(1:nx,1:ny), intent(in)         :: fshxy
    real, dimension(1:nx,1:ny), intent(in)         :: flhxy
    real, dimension(1:nx,1:ny), intent(in)         :: fghxy
    real, dimension(1:nx,1:ny), intent(in)         :: aparxy
    real, dimension(1:nx,1:ny), intent(in)         :: psnxy
    real, dimension(1:nx,1:ny), intent(in)         :: savxy
    real, dimension(1:nx,1:ny), intent(in)         :: sagxy
    real, dimension(1:nx,1:ny), intent(in)         :: fsnoxy
    real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: stcxy
    real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: smcxy
    real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: sh2oxy
    real, dimension(1:nx,1:ny), intent(in)         :: xlaixy
    real, dimension(1:nx,1:ny), intent(in)         :: xsaixy
    real, dimension(1:nx,1:ny), intent(in)         :: tradxy
    real, dimension(1:nx,1:ny), intent(in)         :: neexy
    real, dimension(1:nx,1:ny), intent(in)         :: gppxy
    real, dimension(1:nx,1:ny), intent(in)         :: nppxy
    real, dimension(1:nx,1:ny), intent(in)         :: tsxy
    real, dimension(1:nx,1:ny), intent(in)         :: fvegxy
```

```
    real, dimension(1:nx,1:ny), intent(in)         :: cmxy
    real, dimension(1:nx,1:ny), intent(in)         :: chxy

    integer :: ND
    INTEGER, PARAMETER :: nvar = 60
    integer :: ix,iy,iz
    integer, dimension(1:nsoil)  :: ilev
    real, dimension(1:nx)        :: lon
    real, dimension(1:ny)        :: lat
    real, dimension(nx,ny)       :: snowhm
    real, dimension(nx,ny)       :: sneqvm
    real, dimension(nx,ny)       :: tgm
    real, dimension(nx,ny)       :: prcpm
    real, dimension(nx,ny)       :: runsfm
    real, dimension(nx,ny)       :: runsbm
    real, dimension(nx,ny)       :: ecanm
    real, dimension(nx,ny)       :: edirm
    real, dimension(nx,ny)       :: etranm
    real, dimension(nx,ny)       :: zwtm
    real, dimension(nx,ny)       :: fsam
    real, dimension(nx,ny)       :: firam
    real, dimension(nx,ny)       :: fshm
    real, dimension(nx,ny)       :: flhm
    real, dimension(nx,ny)       :: fghm
    real, dimension(nx,ny)       :: aparm
    real, dimension(nx,ny)       :: psnm
    real, dimension(nx,ny)       :: savm
    real, dimension(nx,ny)       :: sagm
    real, dimension(nx,ny)       :: fsnom
    real, dimension(nx,ny)       :: xlaim
    real, dimension(nx,ny)       :: xsaim
    real, dimension(nx,ny)       :: tradm
    real, dimension(nx,ny)       :: neem
    real, dimension(nx,ny)       :: gppm
    real, dimension(nx,ny)       :: nppm
    real, dimension(nx,ny)       :: tsm
    real, dimension(nx,ny)       :: fvegm
    real, dimension(nx,ny)       :: cmm
    real, dimension(nx,ny)       :: chm
    real, dimension(nx,ny,nsoil) :: stcm
    real, dimension(nx,ny,nsoil) :: smcm
    real, dimension(nx,ny,nsoil) :: sh2om

    real ::  miss
    character(len=256) :: ncfile
    character(len=120) :: vname
    character(len=120) :: vunit

    INCLUDE 'netcdf.inc'

    INTEGER STATUS
    INTEGER NCID
    INTEGER varID(nvar),lonID,latID,levID
    INTEGER VARDIMS(2),VARDIMS3d(3),XDIM,YDIM,ZDIM
    INTEGER START3d(3), COUNT3d(3)
    INTEGER START(2), COUNT(2)
    DATA START   /  1,    1/
    DATA START3d /  1,    1,    1/
    DATA miss/-999.9/

     count(1)    = nx
     count(2)    = ny
     count3d(1)  = nx
     count3d(2)  = ny
     count3d(3)  = nsoil

! compute daily mean

     if(imstep == 1) then
```

```
      ND = 0
      do ix = 1,nx
      do iy = 1,ny
      if(vegtypxy(ix,iy) > 0) then
         snowhm(ix,iy) = 0.
         sneqvm(ix,iy) = 0.
         tgm   (ix,iy) = 0.
         prcpm (ix,iy) = 0.
         runsfm(ix,iy) = 0.
         runsbm(ix,iy) = 0.
         ecanm (ix,iy) = 0.
         edirm (ix,iy) = 0.
         etranm(ix,iy) = 0.
         zwtm  (ix,iy) = 0.
         fsam  (ix,iy) = 0.
         firam (ix,iy) = 0.
         fshm  (ix,iy) = 0.
         flhm  (ix,iy) = 0.
         fghm  (ix,iy) = 0.
         aparm (ix,iy) = 0.
         psnm  (ix,iy) = 0.
         savm  (ix,iy) = 0.
         sagm  (ix,iy) = 0.
         fsnom (ix,iy) = 0.
         xlaim (ix,iy) = 0.
         xsaim (ix,iy) = 0.
         tradm (ix,iy) = 0.
         tsm   (ix,iy) = 0.
         neem  (ix,iy) = 0.
         gppm  (ix,iy) = 0.
         nppm  (ix,iy) = 0.
         fvegm (ix,iy) = 0.

         do iz = 1, nsoil
         stcm  (ix,iy,iz) = 0.
         smcm  (ix,iy,iz) = 0.
         sh2om (ix,iy,iz) = 0.
         end do
      end if
      end do
      end do
   end if

   ND = ND + 1
   do ix = 1,nx
   do iy = 1,ny
   if(vegtypxy(ix,iy) > 0) then
      snowhm(ix,iy) = snowhm(ix,iy) + snowhxy(ix,iy)
      sneqvm(ix,iy) = sneqvm(ix,iy) + sneqvxy(ix,iy)
      tgm   (ix,iy) = tgm   (ix,iy) + tgxy   (ix,iy)
      prcpm (ix,iy) = prcpm (ix,iy) + prcpxy (ix,iy)
      runsfm(ix,iy) = runsfm(ix,iy) + runsfxy(ix,iy)
      runsbm(ix,iy) = runsbm(ix,iy) + runsbxy(ix,iy)
      ecanm (ix,iy) = ecanm (ix,iy) + ecanxy (ix,iy)
      edirm (ix,iy) = edirm (ix,iy) + edirxy (ix,iy)
      etranm(ix,iy) = etranm(ix,iy) + etranxy(ix,iy)
      zwtm  (ix,iy) = zwtm  (ix,iy) + zwtxy  (ix,iy)
      fsam  (ix,iy) = fsam  (ix,iy) + fsaxy  (ix,iy)
      firam (ix,iy) = firam (ix,iy) + firaxy (ix,iy)
      fshm  (ix,iy) = fshm  (ix,iy) + fshxy  (ix,iy)
      flhm  (ix,iy) = flhm  (ix,iy) + flhxy  (ix,iy)
      fghm  (ix,iy) = fghm  (ix,iy) + fghxy  (ix,iy)
      aparm (ix,iy) = aparm (ix,iy) + aparxy (ix,iy)
      psnm  (ix,iy) = psnm  (ix,iy) + psnxy  (ix,iy)
      savm  (ix,iy) = savm  (ix,iy) + savxy  (ix,iy)
      sagm  (ix,iy) = sagm  (ix,iy) + sagxy  (ix,iy)
      fsnom (ix,iy) = fsnom (ix,iy) + fsnoxy (ix,iy)
      xlaim (ix,iy) = xlaim (ix,iy) + xlaixy (ix,iy)
      xsaim (ix,iy) = xsaim (ix,iy) + xsaixy (ix,iy)
```

```
        tradm (ix,iy) = tradm (ix,iy) + tradxy (ix,iy)
        tsm   (ix,iy) = tsm   (ix,iy) + tsxy   (ix,iy)
        neem  (ix,iy) = neem  (ix,iy) + neexy  (ix,iy)
        gppm  (ix,iy) = gppm  (ix,iy) + gppxy  (ix,iy)
        nppm  (ix,iy) = nppm  (ix,iy) + nppxy  (ix,iy)
        fvegm (ix,iy) = fvegm (ix,iy) + fvegxy (ix,iy)
        cmm   (ix,iy) = cmm   (ix,iy) + cmxy   (ix,iy)
        chm   (ix,iy) = chm   (ix,iy) + chxy   (ix,iy)

        do iz = 1, nsoil
        stcm  (ix,iy,iz) = stcm  (ix,iy,iz) + stcxy  (ix,iy,iz)
        smcm  (ix,iy,iz) = smcm  (ix,iy,iz) + smcxy  (ix,iy,iz)
        sh2om (ix,iy,iz) = sh2om (ix,iy,iz) + sh2oxy (ix,iy,iz)
        end do

     end if
     end do
     end do

     if(ND == nday(imonth)*(86400./dt)) then
        do ix = 1,nx
          lon(ix) = lonxy(ix,1)
        end do

        do iy = 1, ny
          lat(iy) = latxy(1,iy)
        end do

        do iz = 1, nsoil
          ilev(iz) = iz
        end do

        do ix = 1,nx
        do iy = 1,ny
        if(vegtypxy(ix,iy) > 0) then
          snowhm(ix,iy) = snowhm(ix,iy) / ND
          sneqvm(ix,iy) = sneqvm(ix,iy) / ND
          tgm   (ix,iy) = tgm   (ix,iy) / ND
          prcpm (ix,iy) = prcpm (ix,iy) / ND
          runsfm(ix,iy) = runsfm(ix,iy) / ND
          runsbm(ix,iy) = runsbm(ix,iy) / ND
          ecanm (ix,iy) = ecanm (ix,iy) / ND
          edirm (ix,iy) = edirm (ix,iy) / ND
          etranm(ix,iy) = etranm(ix,iy) / ND
          zwtm  (ix,iy) = zwtm  (ix,iy) / ND
          fsam  (ix,iy) = fsam  (ix,iy) / ND
          firam (ix,iy) = firam (ix,iy) / ND
          fshm  (ix,iy) = fshm  (ix,iy) / ND
          flhm  (ix,iy) = flhm  (ix,iy) / ND
          fghm  (ix,iy) = fghm  (ix,iy) / ND
          aparm (ix,iy) = aparm (ix,iy) / ND
          psnm  (ix,iy) = psnm  (ix,iy) / ND
          savm  (ix,iy) = savm  (ix,iy) / ND
          sagm  (ix,iy) = sagm  (ix,iy) / ND
          fsnom (ix,iy) = fsnom (ix,iy) / ND
          xlaim (ix,iy) = xlaim (ix,iy) / ND
          xsaim (ix,iy) = xsaim (ix,iy) / ND
          tradm (ix,iy) = tradm (ix,iy) / ND
          tsm   (ix,iy) = tsm   (ix,iy) / ND
          neem  (ix,iy) = neem  (ix,iy) / ND
          gppm  (ix,iy) = gppm  (ix,iy) / ND
          nppm  (ix,iy) = nppm  (ix,iy) / ND
          fvegm (ix,iy) = fvegm (ix,iy) / ND
          cmm   (ix,iy) = cmm   (ix,iy) / ND
          chm   (ix,iy) = chm   (ix,iy) / ND

          do iz = 1, nsoil
          stcm  (ix,iy,iz) = stcm  (ix,iy,iz) / ND
          smcm  (ix,iy,iz) = smcm  (ix,iy,iz) / ND
```

```
          sh2om (ix,iy,iz) = sh2om (ix,iy,iz) / ND
          end do
        end if
        end do
        end do

        ! creat nc output files and initialize output file ID

        if(imonth < 10) then
              write(ncfile,100) TRIM(EXP),iyear,imonth
        else
              write(ncfile,200) TRIM(EXP),iyear,imonth
        end if

100     format('/results/',a,'/hist/Noah.monthlymean.',i4,'0',i1,'.nc')
200     format('/results/',a,'/hist/Noah.monthlymean.',i4,i2,'.nc')

        write(*,*) '---------------------------------------------------------------------------------'
        write(*,*) 'opening ncfile: ',TRIM(DIR)//TRIM(ncfile)
        write(*,*) '---------------------------------------------------------------------------------'

        ncfile = TRIM(DIR)//ncfile

        STATUS=NF_CREATE(ncfile,NF_CLOBBER, NCID)
        STATUS=NF_DEF_DIM(NCID, 'nx'   ,nx      ,  XDIM)
        STATUS=NF_DEF_DIM(NCID, 'ny'   ,ny      ,  YDIM)
        STATUS=NF_DEF_DIM(NCID, 'nz'   ,nsoil   ,  ZDIM)
        VARDIMS(1) = XDIM
        VARDIMS(2) = YDIM
        VARDIMS3d(1) = XDIM
        VARDIMS3d(2) = YDIM
        VARDIMS3d(3) = ZDIM
        STATUS = nf_def_var(NCID,'lon',nf_float,1,XDIM,lonID)
        STATUS = nf_def_var(NCID,'lat',nf_float,1,YDIM,latID)
        STATUS = nf_def_var(NCID,'lev',nf_float,1,ZDIM,levID)

     ! initilize output variables ID

        STATUS = nf_def_var(NCID,'SNOWD',NF_FLOAT,2,VARDIMS,varID(1))
        vname = 'snow depth'
        vunit = 'm'
        STATUS = nf_put_att_text(NCID,varID(1),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(1),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(1),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'SWE',NF_FLOAT,2,VARDIMS,varID(2))
        vname = 'snow water equivalent'
        vunit = 'kg/m2'
        STATUS = nf_put_att_text(NCID,varID(2),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(2),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(2),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'TG',NF_FLOAT,2,VARDIMS,varID(3))
        vname = 'ground surface temperature'
        vunit = 'K'
        STATUS = nf_put_att_text(NCID,varID(3),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(3),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(3),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'PRCP',NF_FLOAT,2,VARDIMS,varID(4))
        vname = 'precipitation'
        vunit = 'mm/s'
        STATUS = nf_put_att_text(NCID,varID(4),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(4),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(4),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'RUNSF',NF_FLOAT,2,VARDIMS,varID(5))
        vname = 'surface runoff'
        vunit = 'mm/s'
```

```
STATUS = nf_put_att_text(NCID,varID(5),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(5),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(5),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'RUNSB',NF_FLOAT,2,VARDIMS,varID(6))
vname = 'subsurface runoff'
vunit = 'mm/s'
STATUS = nf_put_att_text(NCID,varID(6),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(6),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(6),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'ECAN',NF_FLOAT,2,VARDIMS,varID(7))
vname = 'canopy interception loss'
vunit = 'mm/s'
STATUS = nf_put_att_text(NCID,varID(7),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(7),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(7),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'ESOIL',NF_FLOAT,2,VARDIMS,varID(8))
vname = 'soil surface evaporation'
vunit = 'mm/s'
STATUS = nf_put_att_text(NCID,varID(8),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(8),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(8),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'ETRAN',NF_FLOAT,2,VARDIMS,varID(9))
vname = 'transpiration'
vunit = 'mm/s'
STATUS = nf_put_att_text(NCID,varID(9),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(9),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(9),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'ZWT',NF_FLOAT,2,VARDIMS,varID(10))
vname = 'depth to water table'
vunit = 'm'
STATUS = nf_put_att_text(NCID,varID(10),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(10),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(10),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'STC',NF_FLOAT,3,VARDIMS3d,varID(11))
vname = 'soil temperature (4-L)'
vunit = 'K'
STATUS = nf_put_att_text(NCID,varID(11),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(11),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(11),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'SMC',NF_FLOAT,3,VARDIMS3d,varID(12))
vname = 'soil moisture content (4-L)'
vunit = 'm3/m3'
STATUS = nf_put_att_text(NCID,varID(12),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(12),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(12),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'SH2O',NF_FLOAT,3,VARDIMS3d,varID(13))
vname = 'soil liquid water content (4-L)'
vunit = 'm3/m3'
STATUS = nf_put_att_text(NCID,varID(13),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(13),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(13),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'FSA',NF_FLOAT,2,VARDIMS,varID(14))
vname = 'net solar radiation'
vunit = 'W/m2'
STATUS = nf_put_att_text(NCID,varID(14),'long_name',lencs(vname),vname)
STATUS = nf_put_att_text(NCID,varID(14),'units'    ,lencs(vunit),vunit)
status = nf_put_att_REAL(NCID,varID(14),'missing_value',NF_FLOAT,1,miss)

STATUS = nf_def_var(NCID,'FIRA',NF_FLOAT,2,VARDIMS,varID(15))
vname = 'net longwave radiation'
```

```
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(15),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(15),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(15),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FSH',NF_FLOAT,2,VARDIMS,varID(16))
        vname = 'sensible heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(16),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(16),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(16),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FLH',NF_FLOAT,2,VARDIMS,varID(17))
        vname = 'latent heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(17),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(17),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(17),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FGH',NF_FLOAT,2,VARDIMS,varID(18))
        vname = 'ground heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(18),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(18),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(18),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'APAR',NF_FLOAT,2,VARDIMS,varID(19))
        vname = 'photosyn active radiation'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(19),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(19),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(19),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'PSN',NF_FLOAT,2,VARDIMS,varID(20))
        vname = 'total photosynthesis'
        vunit = 'umol co2/m2/s'
        STATUS = nf_put_att_text(NCID,varID(20),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(20),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(20),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'SAV',NF_FLOAT,2,VARDIMS,varID(21))
        vname = 'solar rad absorbed by veg.`'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(21),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(21),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(21),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'SAG',NF_FLOAT,2,VARDIMS,varID(22))
        vname = 'solar rad absorbed by ground'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(22),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(22),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(22),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FSNO',NF_FLOAT,2,VARDIMS,varID(23))
        vname = 'snow cover fraction on the ground'
        vunit = 'fraction'
        STATUS = nf_put_att_text(NCID,varID(23),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(23),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(23),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'LAI',NF_FLOAT,2,VARDIMS,varID(24))
        vname = 'leaf area index'
        vunit = 'm2/m2'
        STATUS = nf_put_att_text(NCID,varID(24),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(24),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(24),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'SAI',NF_FLOAT,2,VARDIMS,varID(25))
```

```
        vname = 'stem area index'
        vunit = 'm2/m2'
        STATUS = nf_put_att_text(NCID,varID(25),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(25),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(25),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'TRAD',NF_FLOAT,2,VARDIMS,varID(26))
        vname = 'surface radiative temperature'
        vunit = 'K'
        STATUS = nf_put_att_text(NCID,varID(26),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(26),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(26),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'NEE',NF_FLOAT,2,VARDIMS,varID(27))
        vname = 'net CO2 flux'
        vunit = 'g/m2/s CO2'
        STATUS = nf_put_att_text(NCID,varID(27),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(27),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(27),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'GPP',NF_FLOAT,2,VARDIMS,varID(28))
        vname = 'GPP'
        vunit = 'g/m2/s carbon'
        STATUS = nf_put_att_text(NCID,varID(28),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(28),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(28),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'NPP',NF_FLOAT,2,VARDIMS,varID(29))
        vname = 'NPP'
        vunit = 'g/m2/s carbon'
        STATUS = nf_put_att_text(NCID,varID(29),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(29),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(29),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'TS',NF_FLOAT,2,VARDIMS,varID(30))
        vname = 'surface temperature'
        vunit = 'K'
        STATUS = nf_put_att_text(NCID,varID(30),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(30),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(30),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FVEG',NF_FLOAT,2,VARDIMS,varID(31))
        vname = 'greenness vegetation fraction'
        vunit = 'fraction'
        STATUS = nf_put_att_text(NCID,varID(31),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(31),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(31),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'CM',NF_FLOAT,2,VARDIMS,varID(32))
        vname = 'Surface exchange coefficient for momentum'
        vunit = '-'
        STATUS = nf_put_att_text(NCID,varID(32),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(32),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(32),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'CH',NF_FLOAT,2,VARDIMS,varID(33))
        vname = 'Surface exchange coefficient for heat'
        vunit = '-'
        STATUS = nf_put_att_text(NCID,varID(33),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(33),'units'    ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(33),'missing_value',NF_FLOAT,1,miss)

        STATUS = NF_ENDDEF(NCID)
        status= NF_CLOSE(NCID)

        ! NC format output

        STATUS = NF_OPEN (ncfile, NF_WRITE, NCID)
        status = nf_put_vara_real(NCID,lonID,1,nx,lon)
```

```
        status = nf_put_vara_real(NCID,latID,1,ny,lat)
        status = nf_put_vara_int (NCID,levID,1,nsoil,ilev)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 1),START,COUNT,snowhm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 2),START,COUNT,sneqvm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 3),START,COUNT,tgm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 4),START,COUNT,prcpm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 5),START,COUNT,runsfm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 6),START,COUNT,runsbm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 7),START,COUNT,ecanm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 8),START,COUNT,edirm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 9),START,COUNT,etranm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(10),START,COUNT,zwtm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(11),START3d,COUNT3d,stcm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(12),START3d,COUNT3d,smcm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(13),START3d,COUNT3d,sh2om)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(14),START,COUNT,fsam)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(15),START,COUNT,firam)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(16),START,COUNT,fshm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(17),START,COUNT,flhm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(18),START,COUNT,fghm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(19),START,COUNT,aparm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(20),START,COUNT,psnm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(21),START,COUNT,savm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(22),START,COUNT,sagm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(23),START,COUNT,fsnom)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(24),START,COUNT,xlaim)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(25),START,COUNT,xsaim)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(26),START,COUNT,tradm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(27),START,COUNT,neem)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(28),START,COUNT,gppm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(29),START,COUNT,nppm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(30),START,COUNT,tsm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(31),START,COUNT,fvegm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(32),START,COUNT,cmm)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(33),START,COUNT,chm)

        status= NF_CLOSE(NCID)

        ND = 0
        do ix = 1,nx
        do iy = 1,ny
        if(vegtypxy(ix,iy) > 0) then
           snowhm(ix,iy) = 0.
           sneqvm(ix,iy) = 0.
           tgm   (ix,iy) = 0.
           prcpm (ix,iy) = 0.
           runsfm(ix,iy) = 0.
           runsbm(ix,iy) = 0.
           ecanm (ix,iy) = 0.
           edirm (ix,iy) = 0.
           etranm(ix,iy) = 0.
           zwtm  (ix,iy) = 0.
           fsam  (ix,iy) = 0.
           firam (ix,iy) = 0.
           fshm  (ix,iy) = 0.
           flhm  (ix,iy) = 0.
           fghm  (ix,iy) = 0.
           aparm (ix,iy) = 0.
           psnm  (ix,iy) = 0.
           savm  (ix,iy) = 0.
           sagm  (ix,iy) = 0.
           fsnom (ix,iy) = 0.
           xlaim (ix,iy) = 0.
           xsaim (ix,iy) = 0.
           tradm (ix,iy) = 0.
           tsm   (ix,iy) = 0.
           neem  (ix,iy) = 0.
           gppm  (ix,iy) = 0.
           nppm  (ix,iy) = 0.
```

```
          fvegm (ix,iy) = 0.
          cmm   (ix,iy) = 0.
          chm   (ix,iy) = 0.

          do iz = 1, nsoil
          stcm  (ix,iy,iz) = 0.
          smcm  (ix,iy,iz) = 0.
          sh2om (ix,iy,iz) = 0.
          end do
       end if
       end do
       end do
    end if

! ------------------------------------------------------------------
   END SUBROUTINE NC_OUT
! ------------------------------------------------------------------
!-------------------------------------------------------------------
   SUBROUTINE NC_OUT_3hr(nsoil   ,nx      ,ny      ,it      ,dt  , &
                  iyear   ,imonth  ,iday    ,DIR     ,EXP     ,lonxy      , &
                  latxy   ,vegtypxy,idstep  ,nday    ,&
                  snowhxy ,sneqvxy ,tgxy    ,stcxy   ,smcxy   ,  &
                  sh2oxy  ,prcpxy  ,runsfxy ,runsbxy ,ecanxy  ,  &
                  edirxy  ,etranxy ,zwtxy   ,fsaxy   ,firaxy  ,  &
                  fshxy   ,flhxy   ,fghxy   ,aparxy  ,psnxy   ,  &
                  savxy   ,sagxy   ,fsnoxy  ,xlaixy  ,xsaixy  ,  &
                  tradxy  ,neexy   ,gppxy   ,nppxy   ,tsxy    ,  &
                  fvegxy  )

   implicit none

! inputs
   integer, intent(in) :: iyear,imonth
   CHARACTER(len=256),INTENT(IN) :: DIR
   CHARACTER(len=8),INTENT(IN) :: EXP
   integer, intent(in)          :: it
   integer, intent(in)          :: idstep
   integer, intent(in)          :: nx
   integer, intent(in)          :: ny
   integer, intent(in)          :: nsoil
   integer, intent(in)          :: iday
   integer, intent(in)          :: nday(12)
   real, intent(in)             :: dt
   integer, dimension(1:nx,1:ny), intent(in)      :: vegtypxy
   real, dimension(1:nx,1:ny), intent(in)         :: lonxy
   real, dimension(1:nx,1:ny), intent(in)         :: latxy
   real, dimension(1:nx,1:ny), intent(in)         :: snowhxy
   real, dimension(1:nx,1:ny), intent(in)         :: sneqvxy
   real, dimension(1:nx,1:ny), intent(in)         :: tgxy
   real, dimension(1:nx,1:ny), intent(in)         :: prcpxy
   real, dimension(1:nx,1:ny), intent(in)         :: runsfxy
   real, dimension(1:nx,1:ny), intent(in)         :: runsbxy
   real, dimension(1:nx,1:ny), intent(in)         :: ecanxy
   real, dimension(1:nx,1:ny), intent(in)         :: edirxy
   real, dimension(1:nx,1:ny), intent(in)         :: etranxy
   real, dimension(1:nx,1:ny), intent(in)         :: zwtxy
   real, dimension(1:nx,1:ny), intent(in)         :: fsaxy
   real, dimension(1:nx,1:ny), intent(in)         :: firaxy
   real, dimension(1:nx,1:ny), intent(in)         :: fshxy
   real, dimension(1:nx,1:ny), intent(in)         :: flhxy
   real, dimension(1:nx,1:ny), intent(in)         :: fghxy
   real, dimension(1:nx,1:ny), intent(in)         :: aparxy
   real, dimension(1:nx,1:ny), intent(in)         :: psnxy
   real, dimension(1:nx,1:ny), intent(in)         :: savxy
   real, dimension(1:nx,1:ny), intent(in)         :: sagxy
   real, dimension(1:nx,1:ny), intent(in)         :: fsnoxy
   real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: stcxy
   real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: smcxy
   real, dimension(1:nx,1:ny,1:nsoil), intent(in) :: sh2oxy
```

```
    real, dimension(1:nx,1:ny), intent(in)          :: xlaixy
    real, dimension(1:nx,1:ny), intent(in)          :: xsaixy
    real, dimension(1:nx,1:ny), intent(in)          :: tradxy
    real, dimension(1:nx,1:ny), intent(in)          :: neexy
    real, dimension(1:nx,1:ny), intent(in)          :: gppxy
    real, dimension(1:nx,1:ny), intent(in)          :: nppxy
    real, dimension(1:nx,1:ny), intent(in)          :: tsxy
    real, dimension(1:nx,1:ny), intent(in)          :: fvegxy

    INTEGER, PARAMETER :: nvar = 60
    integer :: N,ND,ix,iy,iz
    integer :: nt
    integer, dimension(1:nsoil)  :: ilev
    real, dimension(1:nx)        :: lon
    real, dimension(1:ny)        :: lat

    real ::  miss
    character(len=120) :: ncfile
    character(len=120) :: vname
    character(len=120) :: vunit

    INCLUDE 'netcdf.inc'

    INTEGER STATUS
    INTEGER NCID
    INTEGER varID(nvar),lonID,latID,levID,TIMID
    INTEGER XDIM,YDIM,ZDIM,TDIM
    INTEGER VARDIMS(3)
    INTEGER START(3), COUNT(3)
    INTEGER VARDIMS4d(4)
    INTEGER START4d(4), COUNT4d(4)
    DATA START   /  1,    1,    1/
    DATA START4d /  1,    1,    1,    1/
    DATA miss/-999.9/

     count(1)    = nx
     count(2)    = ny
     count(3)    = 1

     count4d(1)  = nx
     count4d(2)  = ny
     count4d(3)  = nsoil
     count4d(4)  = 1

     start(3)    = idstep
     start4d(4)  = idstep

     IF(idstep == 1) THEN

        do ix = 1,nx
          lon(ix) = lonxy(ix,1)
        end do

        do iy = 1, ny
          lat(iy) = latxy(1,iy)
        end do

        do iz = 1, nsoil
          ilev(iz) = iz
        end do

        nt = 86400./dt

        ! creat nc output files and initialize output file ID

        if(imonth < 10) then
           if(iday < 10) then
              write(ncfile,100) TRIM(EXP),iyear,imonth,iday
           else
```

```
            write(ncfile,200) TRIM(EXP),iyear,imonth,iday
         end if
      else
         if(iday < 10) then
            write(ncfile,300) TRIM(EXP),iyear,imonth,iday
         else
            write(ncfile,400) TRIM(EXP),iyear,imonth,iday
         end if
      end if
100   format('/results/',a,'/3hrly/Noah.3hrly.',i4,'0',i1,'0',i1,'.nc')
200   format('/results/',a,'/3hrly/Noah.3hrly.',i4,'0',i1,i2,'.nc')
300   format('/results/',a,'/3hrly/Noah.3hrly.',i4,i2,'0',i1,'.nc')
400   format('/results/',a,'/3hrly/Noah.3hrly.',i4,i2,i2,'.nc')

      write(*,*) '---------------------------------------------------------------------------------'
      write(*,*) 'opening ncfile: ../Noah_data',trim(ncfile)
      write(*,*) '---------------------------------------------------------------------------------'

      ncfile = TRIM(DIR)//ncfile

      STATUS=NF_CREATE(ncfile,NF_CLOBBER, NCID)
      STATUS=NF_DEF_DIM(NCID, 'time' ,nt      , TDIM)
      STATUS=NF_DEF_DIM(NCID, 'nx'   ,nx      , XDIM)
      STATUS=NF_DEF_DIM(NCID, 'ny'   ,ny      , YDIM)
      STATUS=NF_DEF_DIM(NCID, 'nz'   ,nsoil   , ZDIM)
      VARDIMS(1) = XDIM
      VARDIMS(2) = YDIM
      VARDIMS(3) = TDIM

      VARDIMS4d(1) = XDIM
      VARDIMS4d(2) = YDIM
      VARDIMS4d(3) = ZDIM
      VARDIMS4d(4) = TDIM


     ! initilize output variables ID

      status = nf_def_var(NCID,'time',nf_float,1,TDIM,TIMID)
      vname = 'time step'
      vunit = '3hrly timestep'
      STATUS = nf_put_att_text(NCID,TIMID,'long_name',lencs(vname),vname)
      STATUS = nf_put_att_text(NCID,TIMID,'units'    ,lencs(vunit),vunit)

      STATUS = nf_def_var(NCID,'lon',nf_float,1,XDIM,lonID)
      vname = 'longitude coordinate'
      vunit = 'degrees_east'
      STATUS = nf_put_att_text(NCID,lonID,'long_name',lencs(vname),vname)
      STATUS = nf_put_att_text(NCID,lonID,'units'    ,lencs(vunit),vunit)

      STATUS = nf_def_var(NCID,'lat',nf_float,1,YDIM,latID)
      vname = 'latitude coordinate'
      vunit = 'degrees_north'
      STATUS = nf_put_att_text(NCID,latID,'long_name',lencs(vname),vname)
      STATUS = nf_put_att_text(NCID,latID,'units'    ,lencs(vunit),vunit)

      STATUS = nf_def_var(NCID,'lev',nf_int,1,ZDIM,levID)
      vname = 'soil layers'
      vunit = 'from surface to bottom'
      STATUS = nf_put_att_text(NCID,levID,'long_name',lencs(vname),vname)
      STATUS = nf_put_att_text(NCID,levID,'units'    ,lencs(vunit),vunit)

      STATUS = nf_def_var(NCID,'SNOWD',NF_FLOAT,3,VARDIMS,varID(1))
      vname = 'snow depth'
      vunit = 'm'
      STATUS = nf_put_att_text(NCID,varID(1),'long_name',lencs(vname),vname)
      STATUS = nf_put_att_text(NCID,varID(1),'units'    ,lencs(vunit),vunit)
      status = nf_put_att_REAL(NCID,varID(1),'missing_value',NF_FLOAT,1,miss)

      STATUS = nf_def_var(NCID,'SWE',NF_FLOAT,3,VARDIMS,varID(2))
```

```
         vname = 'snow water equivalent'
         vunit = 'kg/m2'
         STATUS = nf_put_att_text(NCID,varID(2),'long_name',lencs(vname),vname)
         STATUS = nf_put_att_text(NCID,varID(2),'units'    ,lencs(vunit),vunit)
         status = nf_put_att_REAL(NCID,varID(2),'missing_value',NF_FLOAT,1,miss)

         STATUS = nf_def_var(NCID,'TG',NF_FLOAT,3,VARDIMS,varID(3))
         vname = 'ground surface temperature'
         vunit = 'K'
         STATUS = nf_put_att_text(NCID,varID(3),'long_name',lencs(vname),vname)
         STATUS = nf_put_att_text(NCID,varID(3),'units'    ,lencs(vunit),vunit)
         status = nf_put_att_REAL(NCID,varID(3),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'PRCP',NF_FLOAT,3,VARDIMS,varID(4))
!         vname = 'precipitation'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(4),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(4),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(4),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'RUNSF',NF_FLOAT,3,VARDIMS,varID(5))
!         vname = 'surface runoff'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(5),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(5),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(5),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'RUNSB',NF_FLOAT,3,VARDIMS,varID(6))
!         vname = 'subsurface runoff'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(6),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(6),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(6),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'ECAN',NF_FLOAT,3,VARDIMS,varID(7))
!         vname = 'canopy interception loss'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(7),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(7),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(7),'missing_value',NF_FLOAT,1,miss)
!
!         STATUS = nf_def_var(NCID,'ESOIL',NF_FLOAT,3,VARDIMS,varID(8))
!         vname = 'soil surface evaporation'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(8),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(8),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(8),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'ETRAN',NF_FLOAT,3,VARDIMS,varID(9))
!         vname = 'transpiration'
!         vunit = 'mm/s'
!         STATUS = nf_put_att_text(NCID,varID(9),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(9),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(9),'missing_value',NF_FLOAT,1,miss)
!
!         STATUS = nf_def_var(NCID,'ZWT',NF_FLOAT,3,VARDIMS,varID(10))
!         vname = 'depth to water table'
!         vunit = 'm'
!         STATUS = nf_put_att_text(NCID,varID(10),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(10),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(10),'missing_value',NF_FLOAT,1,miss)

         STATUS = nf_def_var(NCID,'STC',NF_FLOAT,4,VARDIMS4d,varID(11))
         vname = 'soil temperature (4-L)'
         vunit = 'K'
         STATUS = nf_put_att_text(NCID,varID(11),'long_name',lencs(vname),vname)
         STATUS = nf_put_att_text(NCID,varID(11),'units'    ,lencs(vunit),vunit)
         status = nf_put_att_REAL(NCID,varID(11),'missing_value',NF_FLOAT,1,miss)
```

```
!        STATUS = nf_def_var(NCID,'SMC',NF_FLOAT,4,VARDIMS4d,varID(12))
!        vname = 'soil moisture content (4-L)'
!        vunit = 'm3/m3'
!        STATUS = nf_put_att_text(NCID,varID(12),'long_name',lencs(vname),vname)
!        STATUS = nf_put_att_text(NCID,varID(12),'units'     ,lencs(vunit),vunit)
!        status = nf_put_att_REAL(NCID,varID(12),'missing_value',NF_FLOAT,1,miss)

!        STATUS = nf_def_var(NCID,'SH2O',NF_FLOAT,4,VARDIMS4d,varID(13))
!        vname = 'soil liquid water content (4-L)'
!        vunit = 'm3/m3'
!        STATUS = nf_put_att_text(NCID,varID(13),'long_name',lencs(vname),vname)
!        STATUS = nf_put_att_text(NCID,varID(13),'units'     ,lencs(vunit),vunit)
!        status = nf_put_att_REAL(NCID,varID(13),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FSA',NF_FLOAT,3,VARDIMS,varID(14))
        vname = 'net solar radiation'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(14),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(14),'units'     ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(14),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FIRA',NF_FLOAT,3,VARDIMS,varID(15))
        vname = 'net longwave radiation'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(15),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(15),'units'     ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(15),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FSH',NF_FLOAT,3,VARDIMS,varID(16))
        vname = 'sensible heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(16),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(16),'units'     ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(16),'missing_value',NF_FLOAT,1,miss)

        STATUS = nf_def_var(NCID,'FLH',NF_FLOAT,3,VARDIMS,varID(17))
        vname = 'latent heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(17),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(17),'units'     ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(17),'missing_value',NF_FLOAT,1,miss)
        STATUS = nf_def_var(NCID,'FGH',NF_FLOAT,3,VARDIMS,varID(18))
        vname = 'ground heat flux'
        vunit = 'W/m2'
        STATUS = nf_put_att_text(NCID,varID(18),'long_name',lencs(vname),vname)
        STATUS = nf_put_att_text(NCID,varID(18),'units'     ,lencs(vunit),vunit)
        status = nf_put_att_REAL(NCID,varID(18),'missing_value',NF_FLOAT,1,miss)

!        STATUS = nf_def_var(NCID,'APAR',NF_FLOAT,3,VARDIMS,varID(19))
!        vname = 'photosyn active radiation'
!        vunit = 'W/m2'
!        STATUS = nf_put_att_text(NCID,varID(19),'long_name',lencs(vname),vname)
!        STATUS = nf_put_att_text(NCID,varID(19),'units'     ,lencs(vunit),vunit)
!        status = nf_put_att_REAL(NCID,varID(19),'missing_value',NF_FLOAT,1,miss)

!        STATUS = nf_def_var(NCID,'PSN',NF_FLOAT,3,VARDIMS,varID(20))
!        vname = 'total photosynthesis'
!        vunit = 'umol co2/m2/s'
!        STATUS = nf_put_att_text(NCID,varID(20),'long_name',lencs(vname),vname)
!        STATUS = nf_put_att_text(NCID,varID(20),'units'     ,lencs(vunit),vunit)
!        status = nf_put_att_REAL(NCID,varID(20),'missing_value',NF_FLOAT,1,miss)

!        STATUS = nf_def_var(NCID,'SAV',NF_FLOAT,3,VARDIMS,varID(21))
!        vname = 'solar rad absorbed by veg.`'
!        vunit = 'W/m2'
!        STATUS = nf_put_att_text(NCID,varID(21),'long_name',lencs(vname),vname)
!        STATUS = nf_put_att_text(NCID,varID(21),'units'     ,lencs(vunit),vunit)
!        status = nf_put_att_REAL(NCID,varID(21),'missing_value',NF_FLOAT,1,miss)
```

```
!         STATUS = nf_def_var(NCID,'SAG',NF_FLOAT,3,VARDIMS,varID(22))
!         vname = 'solar rad absorbed by ground'
!         vunit = 'W/m2'
!         STATUS = nf_put_att_text(NCID,varID(22),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(22),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(22),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'FSNO',NF_FLOAT,3,VARDIMS,varID(23))
!         vname = 'snow cover fraction on the ground'
!         vunit = 'fraction'
!         STATUS = nf_put_att_text(NCID,varID(23),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(23),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(23),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'LAI',NF_FLOAT,3,VARDIMS,varID(24))
!         vname = 'leaf area index'
!         vunit = 'm2/m2'
!         STATUS = nf_put_att_text(NCID,varID(24),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(24),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(24),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'SAI',NF_FLOAT,3,VARDIMS,varID(25))
!         vname = 'stem area index'
!         vunit = 'm2/m2'
!         STATUS = nf_put_att_text(NCID,varID(25),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(25),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(25),'missing_value',NF_FLOAT,1,miss)

          STATUS = nf_def_var(NCID,'TRAD',NF_FLOAT,3,VARDIMS,varID(26))
          vname = 'surface radiative temperature'
          vunit = 'K'
          STATUS = nf_put_att_text(NCID,varID(26),'long_name',lencs(vname),vname)
          STATUS = nf_put_att_text(NCID,varID(26),'units'    ,lencs(vunit),vunit)
          status = nf_put_att_REAL(NCID,varID(26),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'NEE',NF_FLOAT,3,VARDIMS,varID(27))
!         vname = 'net CO2 flux'
!         vunit = 'g/m2/s CO2'
!         STATUS = nf_put_att_text(NCID,varID(27),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(27),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(27),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'GPP',NF_FLOAT,3,VARDIMS,varID(28))
!         vname = 'GPP'
!         vunit = 'g/m2/s carbon'
!         STATUS = nf_put_att_text(NCID,varID(28),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(28),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(28),'missing_value',NF_FLOAT,1,miss)

!         STATUS = nf_def_var(NCID,'NPP',NF_FLOAT,3,VARDIMS,varID(29))
!         vname = 'NPP'
!         vunit = 'g/m2/s carbon'
!         STATUS = nf_put_att_text(NCID,varID(29),'long_name',lencs(vname),vname)
!         STATUS = nf_put_att_text(NCID,varID(29),'units'    ,lencs(vunit),vunit)
!         status = nf_put_att_REAL(NCID,varID(29),'missing_value',NF_FLOAT,1,miss)

          STATUS = nf_def_var(NCID,'TS',NF_FLOAT,3,VARDIMS,varID(30))
          vname = 'surface temperature'
          vunit = 'K'
          STATUS = nf_put_att_text(NCID,varID(30),'long_name',lencs(vname),vname)
          STATUS = nf_put_att_text(NCID,varID(30),'units'    ,lencs(vunit),vunit)
          status = nf_put_att_REAL(NCID,varID(30),'missing_value',NF_FLOAT,1,miss)

          STATUS = nf_def_var(NCID,'FVEG',NF_FLOAT,3,VARDIMS,varID(31))
          vname = 'greenness vegetation fraction'
          vunit = 'fraction'
          STATUS = nf_put_att_text(NCID,varID(31),'long_name',lencs(vname),vname)
          STATUS = nf_put_att_text(NCID,varID(31),'units'    ,lencs(vunit),vunit)
          status = nf_put_att_REAL(NCID,varID(31),'missing_value',NF_FLOAT,1,miss)
```

```
        STATUS = NF_ENDDEF(NCID)
        status= NF_CLOSE(NCID)

     END IF

     ! NC format output

        STATUS = NF_OPEN (ncfile, NF_WRITE, NCID)

        STATUS = NF_PUT_VARA_REAL(NCID,TIMID,START(3),COUNT(3),idstep*1.)
        if(idstep == 1)then
        status = nf_put_vara_real(NCID, lonID,START(1),COUNT(1),  lon)
        status = nf_put_vara_real(NCID, latID,START(2),COUNT(2),  lat)
        status = nf_put_vara_int (NCID, levID,       1,   nsoil, ilev)
        end if
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 1),START,COUNT,snowhxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 2),START,COUNT,sneqvxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID( 3),START,COUNT,tgxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 4),START,COUNT,prcpxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 5),START,COUNT,runsfxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 6),START,COUNT,runsbxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 7),START,COUNT,ecanxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 8),START,COUNT,edirxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID( 9),START,COUNT,etranxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(10),START,COUNT,zwtxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(11),START4d,COUNT4d,stcxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(12),START4d,COUNT4d,smcxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(13),START4d,COUNT4d,sh2oxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(14),START,COUNT,fsaxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(15),START,COUNT,firaxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(16),START,COUNT,fshxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(17),START,COUNT,flhxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(18),START,COUNT,fghxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(19),START,COUNT,aparxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(20),START,COUNT,psnxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(21),START,COUNT,savxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(22),START,COUNT,sagxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(23),START,COUNT,fsnoxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(24),START,COUNT,xlaixy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(25),START,COUNT,xsaixy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(26),START,COUNT,tradxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(27),START,COUNT,neexy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(28),START,COUNT,gppxy)
!        STATUS = NF_PUT_VARA_REAL(NCID,varID(29),START,COUNT,nppxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(30),START,COUNT,tsxy)
        STATUS = NF_PUT_VARA_REAL(NCID,varID(31),START,COUNT,fvegxy)


        status= NF_CLOSE(NCID)

! ------------------------------------------------------------------
    END SUBROUTINE NC_OUT_3hr
! ------------------------------------------------------------------
! ------------------------------------------------------------------
   SUBROUTINE write_ini(DIR,EXP   ,nx     ,ny     ,nsoil  ,nsnow  ,iyear   ,  &
                       imonth  ,iday   ,itime  ,latxy  ,lonxy  ,            &
                       smcxy   ,stcxy  ,sh2oxy ,tsnoxy ,snicexy ,snliqxy ,  &
                       zsnsoxy ,isnowxy ,snowhxy ,sneqvxy ,canliqxy,canicexy,  &
                       tgxy    ,tvxy   ,waxy   ,wtxy   ,zwtxy  ,lfmassxy,  &
                       rtmassxy ,stmassxy,woodxy  ,stblcpxy,fastcpxy)
! ------------------------------------------------------------------
   implicit none

     CHARACTER(len=256),INTENT(IN) :: DIR
     CHARACTER(len=8),INTENT(IN) :: EXP
     integer, intent(in) :: nx
     integer, intent(in) :: ny
     integer, intent(in) :: nsoil
```

```
integer, intent(in) :: nsnow
integer, intent(in) :: iyear
integer, intent(in) :: imonth
integer, intent(in) :: iday
integer, intent(in) :: itime

real, dimension(1:nx,1:ny), intent(in)          :: lonxy
real, dimension(1:nx,1:ny), intent(in)          :: latxy
real, intent(in), dimension(nx,ny,       1:nsoil) :: smcxy    ! 1
real, intent(in), dimension(nx,ny,       1:nsoil) :: stcxy    ! 2
real, intent(in), dimension(nx,ny,       1:nsoil) :: sh2oxy   ! 3
real, intent(in), dimension(nx,ny,-nsnow+1:     0) :: tsnoxY   ! 4
real, intent(in), dimension(nx,ny,-nsnow+1:     0) :: snicexy  ! 5
real, intent(in), dimension(nx,ny,-nsnow+1:     0) :: snliqxy  ! 6
real, intent(in), dimension(nx,ny,-nsnow+1:nsoil) :: zsnsoxy  ! 7
real, intent(in), dimension(nx,ny)                :: tvxy     ! 8
real, intent(in), dimension(nx,ny)                :: tgxy     ! 9
real, intent(in), dimension(nx,ny)                :: canliqxy !10
real, intent(in), dimension(nx,ny)                :: canicexy !11
real, intent(in), dimension(nx,ny)                :: snowhxy  !12
real, intent(in), dimension(nx,ny)                :: sneqvxy  !13
real, intent(in), dimension(nx,ny)                :: waxy     !14
real, intent(in), dimension(nx,ny)                :: wtxy     !15
real, intent(in), dimension(nx,ny)                :: zwtxy    !16
integer, intent(in), dimension(nx,ny)             :: isnowxy  !17
REAL, intent(in), dimension(nx,ny)    :: lfmassxy!leaf mass [g/m2]
REAL, intent(in), dimension(nx,ny)    :: rtmassxy!mass of fine roots [g/m2]
REAL, intent(in), dimension(nx,ny)    :: stmassxy!stem mass [g/m2]
REAL, intent(in), dimension(nx,ny)    :: woodxy  !mass of wood (incl. woody roots) [g/m2]
REAL, intent(in), dimension(nx,ny)    :: stblcpxy!stable carbon in deep soil [g/m2]
REAL, intent(in), dimension(nx,ny)    :: fastcpxy!short-lived carbon, shallow soil [g/m2]


integer :: ix,iy,iz
character(len=256) :: ncfile

if(imonth < 10) then
     write(ncfile,100) TRIM(EXP),iyear,imonth,iday
else
     write(ncfile,200) TRIM(EXP),iyear,imonth,iday
end if

write(*,*) '=================================================================================='
write(*,*) 'openning initial file: ../Noah_data',trim(ncfile)
write(*,*) '=================================================================================='
ncfile = TRIM(DIR)//ncfile

open(200, file = ncfile, status = 'unknown')

do ix = 1,nx
do iy = 1,ny
    write(200,300)ix,iy,isnowxy(ix,iy)
    write(200,*)(smcxy(ix,iy,iz),iz=1,nsoil),&
                (stcxy(ix,iy,iz),iz=1,nsoil),&
                (sh2oxy(ix,iy,iz),iz=1,nsoil)
    if(isnowxy(ix,iy) .lt. 0) then
    write(200,*)(tsnoxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0),&
                (snicexy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0),&
                (snliqxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,0)
    end if
    write(200,*)(zsnsoxy(ix,iy,iz),iz = isnowxy(ix,iy)+1,nsoil)
    write(200,*)tvxy(ix,iy),tgxy(ix,iy),canicexy(ix,iy),canliqxy(ix,iy),&
                snowhxy(ix,iy),sneqvxy(ix,iy),waxy(ix,iy),wtxy(ix,iy),&
                zwtxy(ix,iy)
    write(200,*) lfmassxy(ix,iy), rtmassxy(ix,iy),stmassxy(ix,iy), &
               woodxy(ix,iy)  , stblcpxy(ix,iy),fastcpxy(ix,iy)

end do
end do
```

```
    close(200)

 100   format('/results/',a,'/ini/Noah.ini.',i4,'0',i1,i2,'2400.dat')
 200   format('/results/',a,'/ini/Noah.ini.',i4,i2,i2,'2400.dat')
 300   format(1x,3i5)

   END SUBROUTINE write_ini
! -------------------------------------------------------------------
    integer function lencs (chrstr)

      implicit none
      character(len=*), intent(in) :: chrstr       !input character string
      integer l

      lencs = 0
      do l = len(chrstr),1,-1
         if (chrstr(l:l).ne.' ' .and. chrstr(l:l).ne.char(0)) then
            lencs = l
            goto 10
         end if
      end do
   10 return

    end function lencs

END MODULE module_sf_Noah_NC_output
```

```
MODULE module_sf_Noahlsm_param_init

! VEGETATION PARAMETERS
  INTEGER :: LUCATS , BARE
  integer, PARAMETER :: NLUS=50
  CHARACTER(LEN=4) LUTYPE
  INTEGER, DIMENSION(1:NLUS) :: NROTBL
  real, dimension(1:NLUS) ::  RSTBL, RGLTBL, HSTBL, SHDTBL
  REAL ::   TOPT_DATA,RSMAX_DATA

! SOIL PARAMETERS
  INTEGER :: SLCATS
  INTEGER, PARAMETER :: NSLTYPE=30
  CHARACTER(LEN=4) SLTYPE
  REAL, DIMENSION (1:NSLTYPE) :: BB,DRYSMC,F11,                           &
       MAXSMC, REFSMC,SATPSI,SATDK,SATDW, WLTSMC,QTZ

! LSM GENERAL PARAMETERS
  INTEGER :: SLPCATS
  INTEGER, PARAMETER :: NSLOPE=30
  REAL, DIMENSION (1:NSLOPE) :: SLOPE_DATA
  REAL ::  SBETA_DATA,FXEXP_DATA,CSOIL_DATA,SALP_DATA,REFDK_DATA,           &
       REFKDT_DATA,FRZK_DATA,ZBOT_DATA,CZIL_DATA

!
CONTAINS
!
!-----------------------------------------------------------------
  SUBROUTINE LSM_PARM_INIT
!-----------------------------------------------------------------


    character(LEN=4) :: MMINLU, MMINSL

    MMINLU='USGS'
    MMINSL='STAS'
    call SOIL_VEG_GEN_PARM( MMINLU, MMINSL)

!-----------------------------------------------------------------
  END SUBROUTINE LSM_PARM_INIT
!-----------------------------------------------------------------


  SUBROUTINE SOIL_VEG_GEN_PARM( MMINLU, MMINSL)
!-----------------------------------------------------------------

    IMPLICIT NONE

    integer :: LUMATCH, IINDEX, LC, NUM_SLOPE

    character(LEN=4) :: MMINLU, MMINSL

    character(len=4) :: lutype, sltype

!-----SPECIFY VEGETATION RELATED CHARACTERISTICS :
!             SHDFAC: Green vegetation fraction (in percentage)
!             NROTBL: Rooting depth (layer)
!              RSMIN: Mimimum stomatal resistance (s m-1)
!                RGL: Parameters used in radiation stress function
!                 HS: Parameter used in vapor pressure deficit functio

!               TOPT: Optimum transpiration air temperature. (K)
!              RSMAX: Max. stomatal resistance (s m-1)
!
!-----READ IN VEGETAION PROPERTIES FROM VEGPARM.TBL
!
    OPEN(19, FILE='VEGPARM.TBL',FORM='FORMATTED',STATUS='OLD')

    PRINT *, 'INPUT LANDUSE = ',MMINLU

    LUMATCH=0
```

```
    READ (19,*)
    READ (19,2000,END=2002)LUTYPE
    READ (19,*)LUCATS,IINDEX
2000 FORMAT (A4)

    IF(LUTYPE.EQ.MMINLU)THEN
       PRINT *, 'LANDUSE TYPE = ',LUTYPE,' FOUND',            &
            LUCATS,' CATEGORIES'
       LUMATCH=1
    ENDIF

    IF(LUTYPE.EQ.MMINLU)THEN
       DO LC=1,LUCATS
          READ (19,*)IINDEX,SHDTBL(LC),NROTBL(LC),RSTBL(LC),RGLTBL(LC),HSTBL(LC)
       ENDDO

       READ (19,*)
       READ (19,*)TOPT_DATA
       READ (19,*)
       READ (19,*)RSMAX_DATA
       READ (19,*)
       READ (19,*)BARE
    ENDIF
!m
2002 CONTINUE

    CLOSE (19)

!
!-----READ IN SOIL PROPERTIES FROM SOILPARM.TBL
!
    OPEN(19, FILE='SOILPARM.TBL',FORM='FORMATTED',STATUS='OLD')

    MMINSL='STAS'                         !oct2
    PRINT *, 'INPUT SOIL TEXTURE CLASSIFICAION = ',MMINSL

    LUMATCH=0

    READ (19,*)
    READ (19,2000,END=2003)SLTYPE
    READ (19,*)SLCATS,IINDEX
    IF(SLTYPE.EQ.MMINSL)THEN
       PRINT *, 'SOIL TEXTURE CLASSIFICATION = ',SLTYPE,' FOUND', &
            SLCATS,' CATEGORIES'
       LUMATCH=1
    ENDIF
    IF(SLTYPE.EQ.MMINSL)THEN
       DO LC=1,SLCATS
          READ (19,*) IINDEX,BB(LC),DRYSMC(LC),F11(LC),MAXSMC(LC),&
               REFSMC(LC),SATPSI(LC),SATDK(LC), SATDW(LC),    &
               WLTSMC(LC), QTZ(LC)
       ENDDO
    ENDIF

2003 CONTINUE

    CLOSE (19)

    IF(LUMATCH.EQ.0)THEN
       PRINT *,'SOIl TEXTURE IN INPUT FILE DOES NOT '
       PRINT *, 'MATCH SOILPARM TABLE'
       STOP 'INCONSISTENT OR MISSING SOILPARM FILE'
    ENDIF

!
!-----READ IN GENERAL PARAMETERS FROM GENPARM.TBL
!
    OPEN(19, FILE='GENPARM.TBL',FORM='FORMATTED',STATUS='OLD')
```

```
    READ (19,*)
    READ (19,*)
    READ (19,*) NUM_SLOPE

    SLPCATS=NUM_SLOPE

    DO LC=1,SLPCATS
       READ (19,*)SLOPE_DATA(LC)
    ENDDO

    READ (19,*)
    READ (19,*)SBETA_DATA
    READ (19,*)
    READ (19,*)FXEXP_DATA
    READ (19,*)
    READ (19,*)CSOIL_DATA
    READ (19,*)
    READ (19,*)SALP_DATA
    READ (19,*)
    READ (19,*)REFDK_DATA
    READ (19,*)
    READ (19,*)REFKDT_DATA
    READ (19,*)
    READ (19,*)FRZK_DATA
    READ (19,*)
    READ (19,*)ZBOT_DATA
    READ (19,*)
    READ (19,*)CZIL_DATA
    CLOSE (19)

    print*,"successful initialize general model parameters"
!-----------------------------------------------------------------
  END SUBROUTINE SOIL_VEG_GEN_PARM
!-----------------------------------------------------------------

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

END MODULE module_sf_Noahlsm_param_init
```

```
MODULE module_Noahlsm_utility

        REAL, PARAMETER      :: CP = 1004.5, RD = 287.04, SIGMA = 5.67E-8,     &
                                CPH2O = 4.218E+3,CPICE = 2.106E+3,            &
                                LSUBF = 3.335E+5

    CONTAINS

    SUBROUTINE CALTMP(T1,SFCTMP, SFCPRS, ZLVL, Q2,                            & !I
                          TH2, T1V, TH2V, RHO )

        IMPLICIT NONE

        REAL, INTENT(IN)       :: Q2, T1, SFCTMP, SFCPRS, ZLVL
        REAL, INTENT(OUT)      :: TH2, T1V, TH2V, RHO
        REAL                   :: T2V

        TH2 = SFCTMP + ( 0.0098 * ZLVL)
        T1V= T1 * (1.0+ 0.61 * Q2)
        TH2V = TH2 * (1.0+ 0.61 * Q2)
        T2V = SFCTMP * ( 1.0 + 0.61 * Q2 )
        RHO = SFCPRS/(RD * T2V)

        END SUBROUTINE CALTMP

        SUBROUTINE CALHUM(SFCTMP, SFCPRS, Q2SAT, DQSDT2)

        IMPLICIT NONE

        REAL, INTENT(IN)       :: SFCTMP, SFCPRS
        REAL, INTENT(OUT)      :: Q2SAT, DQSDT2
        REAL, PARAMETER        :: A2=17.67,A3=273.15,A4=29.65, ELWV=2.501E6,          &
                                  A23M4=A2*(A3-A4), E0=0.611, RV=461.0,             &
                                  EPSILON=0.622
        REAL                   :: ES, SFCPRSX

!  Q2SAT: saturated mixing ratio
        ES = E0 * EXP ( ELWV/RV*(1./A3 - 1./SFCTMP) )
! convert SFCPRS from Pa to KPa
        SFCPRSX = SFCPRS*1.E-3
        Q2SAT = EPSILON * ES / (SFCPRSX-ES)
! convert from  g/g to g/kg
        Q2SAT = Q2SAT * 1.E3
! Q2SAT is currently a 'mixing ratio'

! DQSDT2 is calculated assuming Q2SAT is a specific humidity
        DQSDT2=(Q2SAT/(1+Q2SAT))*A23M4/(SFCTMP-A4)**2

!DG Q2SAT needs to be in g/g when returned for SFLX
        Q2SAT = Q2SAT / 1.E3

        END SUBROUTINE CALHUM


END MODULE module_Noahlsm_utility
```

```
module Module_Date_utilities
contains
  subroutine geth_newdate (ndate, odate, idt)
    implicit none

    !  From old date ("YYYY-MM-DD HH:MM:SS.ffff" or "YYYYMMDDHHMMSSffff") and
    !  delta-time, compute the new date.

    !  on entry     -  odate  -  the old hdate.
    !                  idt    -  the change in time

    !  on exit      -  ndate  -  the new hdate.

    integer, intent(in)           :: idt
    character (len=*), intent(out) :: ndate
    character (len=*), intent(in)  :: odate

    !  Local Variables

    !  yrold    -  indicates the year associated with "odate"
    !  moold    -  indicates the month associated with "odate"
    !  dyold    -  indicates the day associated with "odate"
    !  hrold    -  indicates the hour associated with "odate"
    !  miold    -  indicates the minute associated with "odate"
    !  scold    -  indicates the second associated with "odate"

    !  yrnew    -  indicates the year associated with "ndate"
    !  monew    -  indicates the month associated with "ndate"
    !  dynew    -  indicates the day associated with "ndate"
    !  hrnew    -  indicates the hour associated with "ndate"
    !  minew    -  indicates the minute associated with "ndate"
    !  scnew    -  indicates the second associated with "ndate"

    !  mday     -  a list assigning the number of days in each month

    !  i        -  loop counter
    !  nday     -  the integer number of days represented by "idt"
    !  nhour    -  the integer number of hours in "idt" after taking out
    !              all the whole days
    !  nmin     -  the integer number of minutes in "idt" after taking out
    !              all the whole days and whole hours.
    !  nsec     -  the integer number of minutes in "idt" after taking out
    !              all the whole days, whole hours, and whole minutes.

    integer :: newlen, oldlen
    integer :: yrnew, monew, dynew, hrnew, minew, scnew, frnew
    integer :: yrold, moold, dyold, hrold, miold, scold, frold
    integer :: nday, nhour, nmin, nsec, nfrac, i, ifrc
    logical :: opass
    character (len=10) :: hfrc
    character (len=1) :: sp
    logical :: punct
    integer :: yrstart, yrend, mostart, moend, dystart, dyend
    integer :: hrstart, hrend, mistart, miend, scstart, scend, frstart
    integer :: units
    integer, dimension(12) :: mday = (/31,28,31,30,31,30,31,31,30,31,30,31/)

    ! Determine if odate is "YYYY-MM-DD_HH ... " or "YYYYMMDDHH...."
    if (odate(5:5) == "-") then
       punct = .TRUE.
    else
       punct = .FALSE.
    endif

    !  Break down old hdate into parts

    hrold = 0
    miold = 0
    scold = 0
```

```
    frold = 0
    oldlen = LEN(odate)
    if (punct) then
       yrstart = 1
       yrend = 4
       mostart = 6
       moend = 7
       dystart = 9
       dyend = 10
       hrstart = 12
       hrend = 13
       mistart = 15
       miend = 16
       scstart = 18
       scend = 19
       frstart = 21
       select case (oldlen)
       case (10)
          ! Days
          units = 1
       case (13)
          ! Hours
          units = 2
       case (16)
          ! Minutes
          units = 3
       case (19)
          ! Seconds
          units = 4
       case (21)
          ! Tenths
          units = 5
       case (22)
          ! Hundredths
          units = 6
       case (23)
          ! Thousandths
          units = 7
       case (24)
          ! Ten thousandths
          units = 8
       case default
          write(*,*) 'ERROR: geth_newdate:  odd length: #'//trim(odate)//'#'
          stop
       end select

       if (oldlen.ge.11) then
          sp = odate(11:11)
       else
          sp = ' '
       end if

    else

       yrstart = 1
       yrend = 4
       mostart = 5
       moend = 6
       dystart = 7
       dyend = 8
       hrstart = 9
       hrend = 10
       mistart = 11
       miend = 12
       scstart = 13
       scend = 14
       frstart = 15

       select case (oldlen)
```

```
         case (8)
            ! Days
            units = 1
         case (10)
            ! Hours
            units = 2
         case (12)
            ! Minutes
            units = 3
         case (14)
            ! Seconds
            units = 4
         case (15)
            ! Tenths
            units = 5
         case (16)
            ! Hundredths
            units = 6
         case (17)
            ! Thousandths
            units = 7
         case (18)
            ! Ten thousandths
            units = 8
         case default
            write(*,*) 'ERROR: geth_newdate:  odd length: #'//trim(odate)//'#'
            stop
         end select
      endif

!  Use internal READ statements to convert the CHARACTER string
!  date into INTEGER components.

      read(odate(yrstart:yrend),  '(i4)') yrold
      read(odate(mostart:moend),  '(i2)') moold
      read(odate(dystart:dyend), '(i2)') dyold
      if (units.ge.2) then
         read(odate(hrstart:hrend),'(i2)') hrold
         if (units.ge.3) then
            read(odate(mistart:miend),'(i2)') miold
            if (units.ge.4) then
               read(odate(scstart:scend),'(i2)') scold
               if (units.ge.5) then
                  read(odate(frstart:oldlen),*) frold
               end if
            end if
         end if
      end if

!  Set the number of days in February for that year.

      mday(2) = nfeb(yrold)

!  Check that ODATE makes sense.

      opass = .TRUE.

!  Check that the month of ODATE makes sense.

      if ((moold.gt.12).or.(moold.lt.1)) then
         write(*,*) 'GETH_NEWDATE:  Month of ODATE = ', moold
         opass = .FALSE.
      end if

!  Check that the day of ODATE makes sense.

      if ((dyold.gt.mday(moold)).or.(dyold.lt.1)) then
         write(*,*) 'GETH_NEWDATE:  Day of ODATE = ', dyold
         opass = .FALSE.
```

```
    end if

    !  Check that the hour of ODATE makes sense.

    if ((hrold.gt.23).or.(hrold.lt.0)) then
       write(*,*) 'GETH_NEWDATE:  Hour of ODATE = ', hrold
       opass = .FALSE.
    end if

    !  Check that the minute of ODATE makes sense.

    if ((miold.gt.59).or.(miold.lt.0)) then
       write(*,*) 'GETH_NEWDATE:  Minute of ODATE = ', miold
       opass = .FALSE.
    end if

    !  Check that the second of ODATE makes sense.

    if ((scold.gt.59).or.(scold.lt.0)) then
       write(*,*) 'GETH_NEWDATE:  Second of ODATE = ', scold
       opass = .FALSE.
    end if

    !  Check that the fractional part  of ODATE makes sense.

!KWM      IF ((scold.GT.59).or.(scold.LT.0)) THEN
!KWM         WRITE(*,*) 'GETH_NEWDATE:  Second of ODATE = ', scold
!KWM         opass = .FALSE.
!KWM      END IF

    if (.not.opass) then
       write(*,*) 'Crazy ODATE: ', odate(1:oldlen), oldlen
       stop
    end if

    !  Date Checks are completed.  Continue.


    !  Compute the number of days, hours, minutes, and seconds in idt

    if (units.ge.5) then !idt should be in fractions of seconds
       ifrc = oldlen-(frstart)+1
       ifrc = 10**ifrc
       nday   = abs(idt)/(86400*ifrc)
       nhour  = mod(abs(idt),86400*ifrc)/(3600*ifrc)
       nmin   = mod(abs(idt),3600*ifrc)/(60*ifrc)
       nsec   = mod(abs(idt),60*ifrc)/(ifrc)
       nfrac = mod(abs(idt), ifrc)
    else if (units.eq.4) then  !idt should be in seconds
       ifrc = 1
       nday   = abs(idt)/86400 ! integer number of days in delta-time
       nhour  = mod(abs(idt),86400)/3600
       nmin   = mod(abs(idt),3600)/60
       nsec   = mod(abs(idt),60)
       nfrac  = 0
    else if (units.eq.3) then !idt should be in minutes
       ifrc = 1
       nday   = abs(idt)/1440 ! integer number of days in delta-time
       nhour  = mod(abs(idt),1440)/60
       nmin   = mod(abs(idt),60)
       nsec   = 0
       nfrac  = 0
    else if (units.eq.2) then !idt should be in hours
       ifrc = 1
       nday   = abs(idt)/24 ! integer number of days in delta-time
       nhour  = mod(abs(idt),24)
       nmin   = 0
       nsec   = 0
       nfrac  = 0
```

```
    else if (units.eq.1) then !idt should be in days
       ifrc = 1
       nday   = abs(idt)     ! integer number of days in delta-time
       nhour  = 0
       nmin   = 0
       nsec   = 0
       nfrac  = 0
    else
       write(*,'(''GETH_NEWDATE: Strange length for ODATE: '', i3)') &
            oldlen
       write(*,*) '#'//odate(1:oldlen)//'#'
       stop
    end if

    if (idt.ge.0) then

       frnew = frold + nfrac
       if (frnew.ge.ifrc) then
          frnew = frnew - ifrc
          nsec = nsec + 1
       end if

       scnew = scold + nsec
       if (scnew .ge. 60) then
          scnew = scnew - 60
          nmin  = nmin + 1
       end if

       minew = miold + nmin
       if (minew .ge. 60) then
          minew = minew - 60
          nhour  = nhour + 1
       end if

       hrnew = hrold + nhour
       if (hrnew .ge. 24) then
          hrnew = hrnew - 24
          nday  = nday + 1
       end if

       dynew = dyold
       monew = moold
       yrnew = yrold
       do i = 1, nday
          dynew = dynew + 1
          if (dynew.gt.mday(monew)) then
             dynew = dynew - mday(monew)
             monew = monew + 1
             if (monew .gt. 12) then
                monew = 1
                yrnew = yrnew + 1
                ! If the year changes, recompute the number of days in February
                mday(2) = nfeb(yrnew)
             end if
          end if
       end do

    else if (idt.lt.0) then

       frnew = frold - nfrac
       if (frnew .lt. 0) then
          frnew = frnew + ifrc
          nsec = nsec + 1
       end if

       scnew = scold - nsec
       if (scnew .lt. 00) then
          scnew = scnew + 60
          nmin  = nmin + 1
```

```
       end if

       minew = miold - nmin
       if (minew .lt. 00) then
          minew = minew + 60
          nhour  = nhour + 1
       end if

       hrnew = hrold - nhour
       if (hrnew .lt. 00) then
          hrnew = hrnew + 24
          nday  = nday + 1
       end if

       dynew = dyold
       monew = moold
       yrnew = yrold
       do i = 1, nday
          dynew = dynew - 1
          if (dynew.eq.0) then
             monew = monew - 1
             if (monew.eq.0) then
                monew = 12
                yrnew = yrnew - 1
                ! If the year changes, recompute the number of days in February
                mday(2) = nfeb(yrnew)
             end if
             dynew = mday(monew)
          end if
       end do
    end if

    !  Now construct the new mdate

    newlen = LEN(ndate)

    if (punct) then

       if (newlen.gt.frstart) then
          write(ndate(1:scend),19) yrnew, monew, dynew, hrnew, minew, scnew
          write(hfrc,'(i10)') frnew+1000000000
          ndate = ndate(1:scend)//'.'//hfrc(31-newlen:10)

       else if (newlen.eq.scend) then
          write(ndate(1:scend),19) yrnew, monew, dynew, hrnew, minew, scnew
19        format(i4,'-',i2.2,'-',i2.2,'_',i2.2,':',i2.2,':',i2.2)

       else if (newlen.eq.miend) then
          write(ndate,16) yrnew, monew, dynew, hrnew, minew
16        format(i4,'-',i2.2,'-',i2.2,'_',i2.2,':',i2.2)

       else if (newlen.eq.hrend) then
          write(ndate,13) yrnew, monew, dynew, hrnew
13        format(i4,'-',i2.2,'-',i2.2,'_',i2.2)

       else if (newlen.eq.dyend) then
          write(ndate,10) yrnew, monew, dynew
10        format(i4,'-',i2.2,'-',i2.2)

       end if

    else

       if (newlen.gt.frstart) then
          write(ndate(1:scend),119) yrnew, monew, dynew, hrnew, minew, scnew
          write(hfrc,'(i10)') frnew+1000000000
          ndate = ndate(1:scend)//'.'//hfrc(31-newlen:10)

       else if (newlen.eq.scend) then
```

```fortran
         write(ndate(1:scend),119) yrnew, monew, dynew, hrnew, minew, scnew
119      format(i4,i2.2,i2.2,i2.2,i2.2,i2.2)

      else if (newlen.eq.miend) then
         write(ndate,116) yrnew, monew, dynew, hrnew, minew
116      format(i4,i2.2,i2.2,i2.2,i2.2)

      else if (newlen.eq.hrend) then
         write(ndate,113) yrnew, monew, dynew, hrnew
113      format(i4,i2.2,i2.2,i2.2)

      else if (newlen.eq.dyend) then
         write(ndate,110) yrnew, monew, dynew
110      format(i4,i2.2,i2.2)

      end if

    endif

    if (punct .and. (oldlen.ge.11) .and. (newlen.ge.11)) ndate(11:11) = sp

  end subroutine geth_newdate

  subroutine geth_idts (newdate, olddate, idt)

    implicit none

    !  From 2 input mdates ('YYYY-MM-DD HH:MM:SS.ffff'),
    !  compute the time difference.

    !  on entry     -  newdate  -  the new hdate.
    !                  olddate  -  the old hdate.

    !  on exit      -  idt    -  the change in time.
    !                            Units depend on length of date strings.

    character (len=*) , intent(in) :: newdate, olddate
    integer           , intent(out)   :: idt


    !  Local Variables

    !  yrnew    -  indicates the year associated with "ndate"
    !  yrold    -  indicates the year associated with "odate"
    !  monew    -  indicates the month associated with "ndate"
    !  moold    -  indicates the month associated with "odate"
    !  dynew    -  indicates the day associated with "ndate"
    !  dyold    -  indicates the day associated with "odate"
    !  hrnew    -  indicates the hour associated with "ndate"
    !  hrold    -  indicates the hour associated with "odate"
    !  minew    -  indicates the minute associated with "ndate"
    !  miold    -  indicates the minute associated with "odate"
    !  scnew    -  indicates the second associated with "ndate"
    !  scold    -  indicates the second associated with "odate"
    !  i        -  loop counter
    !  mday     -  a list assigning the number of days in each month

    ! ndate, odate: local values of newdate and olddate
    character(len=24) :: ndate, odate

    integer :: oldlen, newlen
    integer :: yrnew, monew, dynew, hrnew, minew, scnew, frnew
    integer :: yrold, moold, dyold, hrold, miold, scold, frold
    integer :: i, newdys, olddys
    logical :: npass, opass
    integer :: timesign
    integer :: ifrc
    integer, dimension(12) :: mday = (/31,28,31,30,31,30,31,31,30,31,30,31/)
    logical :: punct
```

```
    integer :: yrstart, yrend, mostart, moend, dystart, dyend
    integer :: hrstart, hrend, mistart, miend, scstart, scend, frstart
    integer :: units

    oldlen = len(olddate)
    newlen = len(newdate)
    if (newlen.ne.oldlen) then
       write(*,'("GETH_IDTS: NEWLEN /= OLDLEN: ", A, 3x, A)') newdate(1:newlen), olddate(1:oldlen)
       stop
    endif

    if (olddate.gt.newdate) then
       timesign = -1

       ifrc = oldlen
       oldlen = newlen
       newlen = ifrc

       ndate = olddate
       odate = newdate
    else
       timesign = 1
       ndate = newdate
       odate = olddate
    end if

    ! Break down old hdate into parts

    ! Determine if olddate is punctuated or not
    if (odate(5:5) == "-") then
       punct = .TRUE.
       if (ndate(5:5) /= "-") then
          write(*,'("GETH_IDTS: Dates appear to be different formats: ", A, 3x, A)') &
               ndate(1:newlen), odate(1:oldlen)
          stop
       endif
    else
       punct = .FALSE.
       if (ndate(5:5) == "-") then
          write(*,'("GETH_IDTS: Dates appear to be different formats: ", A, 3x, A)') &
               ndate(1:newlen), odate(1:oldlen)
          stop
       endif
    endif

    if (punct) then
       yrstart = 1
       yrend = 4
       mostart = 6
       moend = 7
       dystart = 9
       dyend = 10
       hrstart = 12
       hrend = 13
       mistart = 15
       miend = 16
       scstart = 18
       scend = 19
       frstart = 21
       select case (oldlen)
       case (10)
          ! Days
          units = 1
       case (13)
          ! Hours
          units = 2
       case (16)
          ! Minutes
          units = 3
```

```
         case (19)
            ! Seconds
            units = 4
         case (21)
            ! Tenths
            units = 5
         case (22)
            ! Hundredths
            units = 6
         case (23)
            ! Thousandths
            units = 7
         case (24)
            ! Ten thousandths
            units = 8
         case default
            write(*,*) 'ERROR: geth_idts:  odd length: #'//trim(odate)//'#'
            stop
         end select
      else

         yrstart = 1
         yrend = 4
         mostart = 5
         moend = 6
         dystart = 7
         dyend = 8
         hrstart = 9
         hrend = 10
         mistart = 11
         miend = 12
         scstart = 13
         scend = 14
         frstart = 15

         select case (oldlen)
         case (8)
            ! Days
            units = 1
         case (10)
            ! Hours
            units = 2
         case (12)
            ! Minutes
            units = 3
         case (14)
            ! Seconds
            units = 4
         case (15)
            ! Tenths
            units = 5
         case (16)
            ! Hundredths
            units = 6
         case (17)
            ! Thousandths
            units = 7
         case (18)
            ! Ten thousandths
            units = 8
         case default
            write(*,*) 'ERROR: geth_idts:  odd length: #'//trim(odate)//'#'
            stop
         end select
      endif


      hrold = 0
      miold = 0
```

```
   scold = 0
   frold = 0

   read(odate(yrstart:yrend), '(i4)') yrold
   read(odate(mostart:moend), '(i2)') moold
   read(odate(dystart:dyend), '(i2)') dyold
   if (units.ge.2) then
      read(odate(hrstart:hrend),'(i2)') hrold
      if (units.ge.3) then
         read(odate(mistart:miend),'(i2)') miold
         if (units.ge.4) then
            read(odate(scstart:scend),'(i2)') scold
            if (units.ge.5) then
               read(odate(frstart:oldlen),*) frold
            end if
         end if
      end if
   end if

!  Break down new hdate into parts

   hrnew = 0
   minew = 0
   scnew = 0
   frnew = 0

   read(ndate(yrstart:yrend), '(i4)') yrnew
   read(ndate(mostart:moend), '(i2)') monew
   read(ndate(dystart:dyend), '(i2)') dynew
   if (units.ge.2) then
      read(ndate(hrstart:hrend),'(i2)') hrnew
      if (units.ge.3) then
         read(ndate(mistart:miend),'(i2)') minew
         if (units.ge.4) then
            read(ndate(scstart:scend),'(i2)') scnew
            if (units.ge.5) then
               read(ndate(frstart:newlen),*) frnew
            end if
         end if
      end if
   end if

!  Check that the dates make sense.

   npass = .true.
   opass = .true.

!  Check that the month of NDATE makes sense.

   if ((monew.gt.12).or.(monew.lt.1)) then
      write(*,*) 'GETH_IDTS:  Month of NDATE = ', monew
      npass = .false.
   end if

!  Check that the month of ODATE makes sense.

   if ((moold.gt.12).or.(moold.lt.1)) then
      print*, 'GETH_IDTS:  Month of ODATE = ', moold
      opass = .false.
   end if

!  Check that the day of NDATE makes sense.

   if (monew.ne.2) then
      ! ...... For all months but February
      if ((dynew.gt.mday(monew)).or.(dynew.lt.1)) then
         print*, 'GETH_IDTS:  Day of NDATE = ', dynew
         npass = .false.
      end if
```

```
    else if (monew.eq.2) then
       ! ...... For February
       if ((dynew > nfeb(yrnew)).or.(dynew < 1)) then
          print*, 'GETH_IDTS:  Day of NDATE = ', dynew
          npass = .false.
       end if
    endif

    !  Check that the day of ODATE makes sense.

    if (moold.ne.2) then
       ! ...... For all months but February
       if ((dyold.gt.mday(moold)).or.(dyold.lt.1)) then
          print*, 'GETH_IDTS:  Day of ODATE = ', dyold
          opass = .false.
       end if
    else if (moold.eq.2) then
       ! ....... For February
       if ((dyold > nfeb(yrold)).or.(dyold < 1)) then
          print*, 'GETH_IDTS:  Day of ODATE = ', dyold
          opass = .false.
       end if
    end if

    !  Check that the hour of NDATE makes sense.

    if ((hrnew.gt.23).or.(hrnew.lt.0)) then
       print*, 'GETH_IDTS:  Hour of NDATE = ', hrnew
       npass = .false.
    end if

    !  Check that the hour of ODATE makes sense.

    if ((hrold.gt.23).or.(hrold.lt.0)) then
       print*, 'GETH_IDTS:  Hour of ODATE = ', hrold
       opass = .false.
    end if

    !  Check that the minute of NDATE makes sense.

    if ((minew.gt.59).or.(minew.lt.0)) then
       print*, 'GETH_IDTS:  Minute of NDATE = ', minew
       npass = .false.
    end if

    !  Check that the minute of ODATE makes sense.

    if ((miold.gt.59).or.(miold.lt.0)) then
       print*, 'GETH_IDTS:  Minute of ODATE = ', miold
       opass = .false.
    end if

    !  Check that the second of NDATE makes sense.

    if ((scnew.gt.59).or.(scnew.lt.0)) then
       print*, 'GETH_IDTS:  SECOND of NDATE = ', scnew
       npass = .false.
    end if

    !  Check that the second of ODATE makes sense.

    if ((scold.gt.59).or.(scold.lt.0)) then
       print*, 'GETH_IDTS:  Second of ODATE = ', scold
       opass = .false.
    end if

    if (.not. npass) then
       print*, 'Screwy NDATE: ', ndate(1:newlen)
       stop
```

```
   end if

   if (.not. opass) then
      print*, 'Screwy ODATE: ', odate(1:oldlen)
      stop
   end if

   !  Date Checks are completed.  Continue.

   !  Compute number of days from 1 January ODATE, 00:00:00 until ndate
   !  Compute number of hours from 1 January ODATE, 00:00:00 until ndate
   !  Compute number of minutes from 1 January ODATE, 00:00:00 until ndate

   newdys = 0
   do i = yrold, yrnew - 1
      newdys = newdys + 337 + nfeb(i)
   end do

   if (monew .gt. 1) then
      mday(2) = nfeb(yrnew)
      do i = 1, monew - 1
         newdys = newdys + mday(i)
      end do
      mday(2) = 28
   end if

   newdys = newdys + dynew - 1

   !  Compute number of hours from 1 January ODATE, 00:00:00 until odate
   !  Compute number of minutes from 1 January ODATE, 00:00:00 until odate

   olddys = 0

   if (moold .gt. 1) then
      mday(2) = nfeb(yrold)
      do i = 1, moold - 1
         olddys = olddys + mday(i)
      end do
      mday(2) = 28
   end if

   olddys = olddys + dyold -1

   !  Determine the time difference

   idt = (newdys - olddys)
   if (units.ge.2) then
      idt = idt*24 + (hrnew - hrold)
      if (units.ge.3) then
         idt = idt*60 + (minew - miold)
         if (units.ge.4) then
            idt = idt*60 + (scnew - scold)
            if (units.ge.5) then
               ifrc = oldlen-(frstart-1)
               ifrc = 10**ifrc
               idt = idt * ifrc + (frnew-frold)
            endif
         endif
      endif
   endif

   if (timesign .eq. -1) then
      idt = idt * timesign
   end if

 end subroutine geth_idts


 integer function nfeb(year)
```

```
    !
    ! Compute the number of days in February for the given year.
    !
    implicit none
    integer, intent(in) :: year ! Four-digit year

    nfeb = 28 ! By default, February has 28 days ...
    if (mod(year,4).eq.0) then
       nfeb = 29  ! But every four years, it has 29 days ...
       if (mod(year,100).eq.0) then
          nfeb = 28  ! Except every 100 years, when it has 28 days ...
          if (mod(year,400).eq.0) then
             nfeb = 29  ! Except every 400 years, when it has 29 days ...
             if (mod(year,3600).eq.0) then
                nfeb = 28  ! Except every 3600 years, when it has 28 days.
             endif
          endif
       endif
    endif
  end function nfeb

  integer function nmdays(hdate)
    !
    ! Compute the number of days in the month of given date hdate.
    !
    implicit none
    character(len=*), intent(in) :: hdate

    integer :: year, month
    integer, dimension(12), parameter :: ndays = (/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /)

    read(hdate(1:7), '(I4,1x,I2)') year, month

    if (month == 2) then
       nmdays = nfeb(year)
    else
       nmdays = ndays(month)
    endif
  end function nmdays

  function monthabbr_to_mm(mon) result(mm)
    implicit none

    character(len=3), intent(in) :: mon

    integer :: mm

    if (mon == "Jan") then
       mm = 1
    elseif (mon == "Feb") then
       mm = 2
    elseif (mon == "Mar") then
       mm = 3
    elseif (mon == "Apr") then
       mm = 4
    elseif (mon == "May") then
       mm = 5
    elseif (mon == "Jun") then
       mm = 6
    elseif (mon == "Jul") then
       mm = 7
    elseif (mon == "Aug") then
       mm = 8
    elseif (mon == "Sep") then
       mm = 9
    elseif (mon == "Oct") then
       mm = 10
    elseif (mon == "Nov") then
       mm = 11
```

```
    elseif (mon == "Dec") then
       mm = 12
    else
       write(*, '("Function monthabbr_to_mm:  mon = <",A,">")') mon
       stop  "Function monthabbr_to_mm:  Unrecognized mon"
    endif
  end function monthabbr_to_mm

  subroutine swap_date_format(indate, outdate)
    implicit none
    character(len=*), intent(in)  :: indate
    character(len=*), intent(out) :: outdate
    integer :: inlen

    inlen = len(indate)
    if (indate(5:5) == "-") then
       select case (inlen)
       case (10)
          ! YYYY-MM-DD
          outdate = indate(1:4)//indate(6:7)//indate(9:10)
       case (13)
          ! YYYY-MM-DD_HH
          outdate = indate(1:4)//indate(6:7)//indate(9:10)//indate(12:13)
       case (16)
          ! YYYY-MM-DD_HH:mm
          outdate = indate(1:4)//indate(6:7)//indate(9:10)//indate(12:13)//indate(15:16)
       case (19)
          ! YYYY-MM-DD_HH:mm:ss
          outdate = indate(1:4)//indate(6:7)//indate(9:10)//indate(12:13)//indate(15:16)//&
               indate(18:19)
       case (21,22,23,24)
          ! YYYY-MM-DD_HH:mm:ss.f[f[f[f]]]
          outdate = indate(1:4)//indate(6:7)//indate(9:10)//indate(12:13)//indate(15:16)//&
               indate(18:19)//indate(21:inlen)
       case default
          write(*,'("Unrecognized length: <", A,">")') indate
          stop
       end select
    else
       select case (inlen)
       case (8)
          ! YYYYMMDD
          outdate = indate(1:4)//"-"//indate(5:6)//"-"//indate(7:8)
       case (10)
          ! YYYYMMDDHH
          outdate = indate(1:4)//"-"//indate(5:6)//"-"//indate(7:8)//"_"//&
               indate(9:10)
       case (12)
          ! YYYYMMDDHHmm
          outdate = indate(1:4)//"-"//indate(5:6)//"-"//indate(7:8)//"_"//&
               indate(9:10)//":"//indate(11:12)
       case (14)
          ! YYYYMMDDHHmmss
          outdate = indate(1:4)//"-"//indate(5:6)//"-"//indate(7:8)//"_"//&
               indate(9:10)//":"//indate(11:12)//":"//indate(13:14)
       case (15,16,17,18)
          ! YYYYMMDDHHmmssf[f[f[f]]]
          outdate = indate(1:4)//"-"//indate(5:6)//"-"//indate(7:8)//"_"//&
               indate(9:10)//":"//indate(11:12)//":"//indate(13:14)//"."//indate(15:inlen)
       case default
          write(*,'("Unrecognized length: <", A,">")') indate
          stop
       end select
    endif

  end subroutine swap_date_format

  character(len=3) function mm_to_monthabbr(ii) result(mon)
    implicit none
```

```
    integer, intent(in) :: ii
    character(len=3), parameter, dimension(12) :: month = (/ &
         "Jan", "Feb", "Mar", "Apr", "May", "Jun", &
         "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" /)
    if (ii > 0 .and. ii < 13 ) then
       mon = month(ii)
    else
       stop "mm_to_monthabbr"
    endif
  end function mm_to_monthabbr

end module Module_Date_utilities
```