```
!*******************************************************************************
!   University of Torino model of land Processes Interaction with Atmosphere   *
!                                UTOPIA.f90                                     *
!                              version 201801                                   *
!*******************************************************************************
! Author: C. Cassardo (January 1990)                                           *
! Collaborators: J.J. Ji (89-92), A. Longhetto (89-2007), L. Filippi (89-90),  *
!   E. Carena (94), P.M. Ruti (1994-6), Y. Feng (1995), G.Y. Niu (1997),       *
!   M.W. Qian (97-98), P. Ramieri(97-98), F. Mutinelli (98), M.Romani (04),    *
!   G.P. Balsamo (98-99), M.Trevisan (04), N.Loglisci (02-06),                 *
!   M. Manfrin (05), S. Cavalletto (05-06), M. Galli (07-12),                  *
!   R. Bonanno (07-13), I. Cerenzia (12)                                       *
!*******************************************************************************
! Last revision : C. Cassardo (15-Oct-2015)                                    *
INCLUDE '../model/trig.f90'
INCLUDE '../model/ecoclimap_constants_piedmont.f90'
!*******************************************************************************
MODULE CONSTANTS
!*******************************************************************************
! Module containing all model constants and fixed parameters
! Author: M. Trevisan (25-Mar-2004)
! Last revision : C. Cassardo (15-May-2014)
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
IMPLICIT NONE
SAVE
! primary parameters
REAL,PARAMETER :: A_RESIST=3.    ! empirical parameter for routine RESIST (table 12 pag 62
Bonan)
REAL,PARAMETER :: A1=6.1078      ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: A2=17.269      ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: A3=273.15      ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: A4=35.86       ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: A5=0.622       ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: A6=0.378       ! parameter for functions SPECUM, DEWPOINT
REAL,PARAMETER :: AFW=0.14       ! water albedo
REAL,PARAMETER :: AGRMAX=100.        ! parameter for function HAZE
REAL,PARAMETER :: ALB_SNMIN=0.50        ! min snow albedo
REAL,PARAMETER :: ALB_SNMAXX=0.85    ! max snow albedo
REAL,PARAMETER :: BETA=1.035E-4      ! Coefficient for the computation of pressure
REAL,PARAMETER :: B0=6.109177956     ! for deriving vapor pressure respect to ice
REAL,PARAMETER :: B1=5.034698970E-01     ! same
REAL,PARAMETER :: B2=1.886013408E-02     ! same
REAL,PARAMETER :: B3=4.176223716E-04     ! same
REAL,PARAMETER :: B4=5.824720280E-06     ! same
REAL,PARAMETER :: B5=4.838803174E-08     ! same
REAL,PARAMETER :: B6=1.838826904E-10     ! same
REAL,PARAMETER :: BOZ=5.67E-8        ! Boltzmann constant [Wm-2K-4]
REAL,PARAMETER :: BTAU=0.50          ! fraction of net rad abs in sfc layer SOLARHS
REAL,PARAMETER :: CIICE=2050.        ! ice specific heat
REAL,PARAMETER :: CLF=334000.        ! ice fusion latent heat [J/kg]
REAL,PARAMETER :: CHARNOOK=0.016     ! Charnook constant (G94, table 4)
REAL,PARAMETER :: CPAIR=1003.0       ! gas spec heat at p const dry air [J/(kg*K)]
REAL,PARAMETER :: CSW=4186.          ! [J/(kg K)]  water specific heat
REAL,PARAMETER :: DENS_DRY_AIR=1.22 ! air density [kg/m3]
REAL,PARAMETER :: DENS_ICE=920.      ! ice density [kg/m3]
REAL,PARAMETER :: DENS_SN_MAX=450.   ! [kg/m3] maximum snow density
REAL,PARAMETER :: DENS_SN_MIN=80.    ! [kg/m3] minimum snow density
REAL,PARAMETER :: DENS_WAT=1000.0    ! water density [kgm-3]
REAL,PARAMETER :: DTK=273.15         ! zero Celsius degrees in Kelvin
REAL,PARAMETER :: EPSIL=1.E-7        ! computational zero
REAL,PARAMETER :: EPSSN=0.97         ! snow emissivity (Brutsaert,1984,p137)
REAL,PARAMETER :: ERRED = 287.       ! dry air constant [J/kg K]
REAL,PARAMETER :: ERR_VE=-9999.9     ! error parameter
REAL,PARAMETER :: ETAMIN=0.01        ! min volumetric soil water content [kg3/kg3]
REAL,PARAMETER :: EXT_COEF=0.05      ! light extinction coef for water (SOLARHS)
REAL,PARAMETER :: FF2MIN=0.15        ! min value of FF2 (RESIST)
REAL,PARAMETER :: FF3MIN=0.25        ! min value of FF3 (RESIST)
```

```
REAL,PARAMETER :: GRAVITY=9.81      ! gravity constant [m/s2]
REAL,PARAMETER :: LEA_WET_THRE=3.44E-6  ! threshold for considering a leaf as wet [mm]
INTEGER,PARAMETER :: NSOILMAX=11    ! number of soil levels
REAL,PARAMETER :: P0_REF = 1000.    ! reference ground pressure [hPa]
REAL,PARAMETER :: PI=3.141592654    ! Greek PI
REAL,PARAMETER :: PRECSNWHITE=0.0001    ! min snow to allow sfc albedo = max snow albedo
REAL,PARAMETER :: RGAS=8.314472     !Universal gas constant [J/K mol]
REAL,PARAMETER :: RHMAX=96.         ! parameter for function HAZE
REAL,PARAMETER :: RHMIN=84.         ! parameter for function HAZE
REAL,PARAMETER :: ROI=917.0         ! ice density (for soil freezing/melting)
REAL,PARAMETER :: ROOT_FRAC_COST=0.05 ! fraction of soil volume occupied by roots
REAL,PARAMETER :: ROW=999.8         ! water density (for soil freezing/melting)
REAL,PARAMETER :: RSMAX = 5000.     ! max value of RESIST_F (RESIST) [s/m]
REAL,PARAMETER :: WAT_VAP_CON=461.5 ! moist gas constant [Jkg-1K-1]
REAL,PARAMETER :: TAUA=0.008        ! albedo parameter
REAL,PARAMETER :: TAUB=0.071        ! albedo parameter
REAL,PARAMETER :: TAUC=0.006        ! albedo parameter
!REAL,PARAMETER :: TFRZ1=274.15     ! for soil freezing/melting
REAL,PARAMETER :: TFRZ1=271.15      ! for soil freezing/melting
REAL,PARAMETER :: TFRZ2=268.15      ! for soil freezing/melting
REAL,PARAMETER :: T_OPT=298.        ! vegetation optimum temperature [K]
REAL,PARAMETER :: VMAX=5.           ! parameter for function HAZE
REAL,PARAMETER :: VMIN=2.           ! parameter for function HAZE
REAL,PARAMETER :: VON_KARMAN=0.41   ! Von Karman constant
REAL,PARAMETER :: ROCI_I=2.094e6    ! heat capacity of ice (J M-3 K-1)
REAL,PARAMETER :: ROCI_W=4.188e6    ! heat capacity of water (J M-3 K-1)
REAL,PARAMETER :: TH_CO_I=2.2       ! thermal conductivity of ice (W M-1 K-1)
REAL,PARAMETER :: TH_CO_W=0.6       ! thermal conductivity of water (W M-1 K-1)
REAL,PARAMETER :: BAS_SFC_BS_LAY=0.5        ! base of the sfc abs layer [m] (SOLARHS)
! secondary parameters
REAL,PARAMETER :: CONV=2.*PI/360.   ! Radiant-Degrees conversion
!Double precision parameter
INTEGER,PARAMETER :: DBL_PREC=SELECTED_REAL_KIND(p=12)
INTEGER,PARAMETER :: NSU1=9, NSU2=27, NSU3=20, NSU4=9, NSU5=15
INTEGER :: NSU6 !=41
!max number of output variables = number of lines of file memoryout.config
INTEGER,PARAMETER :: NSUDYN=172
!max number of input data
INTEGER,PARAMETER :: MDAT_IN=15
END MODULE
!*****************************************************************************
MODULE VEGET_PARAM
!*****************************************************************************
! Module containing all vegetation parameters
! Originally this routine has been implemented by taking the datase of BATS in
! RAMS3b [Land Cover/Vegetation Type from Biosphere-Atmsophere Transfer Scheme
! (BATS) From NCAR/TN-275+ST, Dickinson et. al], then it has been updated.
!
! Vegetation list:
!   1 - Crop/mixed farming         2 - Short grass
!   3 - Evergreen needleleaf tree  4 - Deciduous needleleaf tree
!   5 - Deciduous broadleaf tree   6 - Evergreen broadleaf tree
!   7 - Tall grass                 8 - Desert
!   9 - Tundra                    10 - Irrigated crop
!  11 - Semi-desert               12 - Ice cap/glacier
!  13 - Bog or marsh              14 - Inland water
!  15 - Ocean                     16 - Evergreen shrub
!  17 - Deciduous shrub           18 - Mixed woodland
!  19 - Settlement                20 - Dense settlement
!  21 - Po Valley (SPC)           22 - Grugliasco
!  23 - Siberia                   24 - Hypothetical grass reference crop
!  25 - Hazel                     26 - Vineyard
!  27 - Asphalt                   28 - Pervious concrete
!  29 - Khabarovsk                30 - Kostroma
!  31 - Ogurtsovo                 32 - Tulun
!  33 - Uralsk                    34 - Yershow
!  35 - Crops for Korean site     36 - Mombercelli tartufaia
```

```
!
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
! Author: C. Cassardo (15-May-2014)
! Last revision : C. Cassardo (30-Dec-2021)
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
USE CONSTANTS
IMPLICIT NONE
SAVE
!--- general
INTEGER,PARAMETER :: NVEG=36
!
!--- fixed (do not change in the course of the year) parameters
!---1-------2-------3-------4-------5-------6-------7-------8-------9------10---
REAL,PARAMETER,DIMENSION(NVEG) :: ALB_VGL = &! albedo for > 0.7 um
 (/.05,    .04,    .03,    .05,    .05,    .05,    .04,    .14,    .05,    .15,&
   .04,    .18,    .02,    .01,    .01,    .03,    .03,    .04,    .03,    .02,&
   .04,    .04,    .04,    .04,    .05,    .04,    .05,    .06,    .10,    .10,&
   .10,    .10,    .10,    .10,    .05,    .05/)
REAL,PARAMETER,DIMENSION(NVEG) :: ALB_VGS = &! albedo for < 0.7 um
 (/.20,    .26,    .10,    .10,    .20,    .15,    .16,    .30,    .20,    .18,&
   .25,    .40,    .12,    .14,    .14,    .10,    .20,    .18,    .16,    .17,&
   .28,    .20,    .25,    .23,    .20,    .23,    .08,    .25,    .18,    .23,&
   .15,    .21,    .18,    .15,    .20,    .20/)
! Activ energy of Rubisco carboxylation [J/mol]
REAL,PARAMETER,DIMENSION(NVEG) :: HAV1 = &
 (/92940., 70000., 63553., 70000., 72105., 60790., 70000., 70000., 70000., 92940.,&
        ., ERR_VE, 70000., ERR_VE, ERR_VE, 70000., 70000., 72105., 70000., ERR_VE,&
        ., 70000., 70000., 70000., 72105., 72105., ERR_VE, ERR_VE, 70000., 70000.,&
        ., 70000., 70000., 70000., 92940., 72105./)
! Activ energy of electron transport reaction [J/mol]
REAL,PARAMETER,DIMENSION(NVEG) :: HAJ1=&
 (/82995., 50000., 51853., 50000., 54204., 43790., 50000., 50000., 50000., 82995.,&
        ., ERR_VE, 50000., ERR_VE, ERR_VE, 50000., 50000., 54204., 50000., ERR_VE,&
        ., 50000., 50000., 50000., 54204., 54204., ERR_VE, ERR_VE, 50000., 50000.,&
        ., 50000., 50000., 50000., 82995., 54204./)
! Deactivation energy of electron transport reaction [J/mol]
REAL,PARAMETER,DIMENSION(NVEG) :: HDJ1=&
 (/156885.,200000.,191320.,200000.,188788.,200000.,200000.,200000.,200000.,200000.,&
       0.,ERR_VE, 200000.,ERR_VE, ERR_VE, 200000.,200000.,200000.,200000.,ERR_VE,&
       0.,200000.,200000.,200000.,188788.,188788.,ERR_VE, ERR_VE, 50000., 50000.,&
        ., 50000., 50000., 50000., 200000.,188788./)
! derived from BATS z0 using inverse relation z0=0.13*hcanopy
REAL,PARAMETER,DIMENSION(NVEG) :: HCANOPY = &
 (/0.8,    0.5,    10.,    10.,    21.,    20.,    1.0,    0.5,    0.4,    0.6,&
   .0,     0.1,    0.3,    .004,   .004,   1.0,    1.0,    8.0,    5.0,    10.,&
   .05,    2.0,    0.085,  0.12,   5.0,    2.0,    0.004,  0.004,  1.5,    1.5,&
   .0,     1.5,    1.0,    1.0,    0.8,    10./)
! soil type (derived by BATS with original code 6 substituted with code 5)
INTEGER,PARAMETER,DIMENSION(NVEG) :: IEXSOL =&
 (/5,      5,      5,      5,      7,      8,      5,      3,      5,      5,&
   ,       12,     5,      5,      5,      5,      5,      5,      5,      5,&
   ,       5,      5,      3,      3,      7,      15,     16,     12,     12,&
    ,      3,      13,     5,      5,      7/)
! Rate of electron transport at optimum temperature [uml m-2 s-1]
REAL,DIMENSION(NVEG) :: JOPT1=&
 (/275.07, 200.,   153.14, 150.,   161.67, 175.13, 200.,   200.,   200.,   275.07,&
      .,   ERR_VE, 200.,   ERR_VE, ERR_VE, 200.,   200.,   161.67, 200.,   ERR_VE,&
      .,   200.,   200.,   200.,   161.67, 161.67, ERR_VE, ERR_VE, 200.,   200.,&
      .,   200.,   200.,   200.,   275.07, 161.67/)
! Noilhan parameter (N89) [W/m2]
REAL,PARAMETER,DIMENSION(NVEG) :: RGL_NOILHAN1 =&
 (/100.,   100.,   30.,    30.,    30.,    30.,    100.,   100.,   100.,   100.,&
      .,   100.,   100.,   100.,   100.,   60.,    60.,    60.,    100.,   100.,&
      .,   100.,   100.,   100.,   30.,    100.,   100.,   100.,   100.,   100.,&
      .,   100.,   100.,   100.,   100.,   30./)
! Minimum stomatal resistance [s/m]
REAL,PARAMETER,DIMENSION(NVEG) :: RSMIN = &
```

```
 (/120.,   200.,   200.,   200.,   200.,   150.,   200.,   200.,   200.,   200.,&
       .,   200.,   200.,   200.,   200.,   200.,   200.,   200.,   200.,   200.,&
       .,   50.,    100.,   200.,   95.24,  120.,   200.,   200.,   100.,   85.,&
       .,   85.,    100.,   100.,   120.,   200./)
! inverse square root of leaf dimension
REAL,PARAMETER,DIMENSION(NVEG) :: SQRTDI = &
 (/10.,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,&
    .0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,    5.0,&
    .0,    10.0,   5.0,    5.0,    3.16,   3.0,    5.0,    5.0,    5.0,    5.0,&
    .0,    5.0,    5.0,    5.0,    10.0,   5.0/)
! Optimum temperature of electron transport chain [^C]
REAL,PARAMETER,DIMENSION(NVEG) :: TOPTJ1=&
 (/36.3,   34.0,   31.0,   34.0,   33.43,  32.19,  34.0,   34.0,   34.0,   36.30,&
    .0,    ERR_VE, 34.0,   ERR_VE, ERR_VE, 34.0,   34.0,   33.43,  34.0,   ERR_VE,&
    .0,    34.0,   34.0,   34.0,   33.43,  33.43,  ERR_VE, ERR_VE, 34.0,   34.0,&
    .0,    34.0,   34.0,   34.0,   36.3,   33.43/)
! Optimum temperature of Rubisco carboxylation [^C]
REAL,PARAMETER,DIMENSION(NVEG) :: TOPTV1=&
 (/41.24,  39.,    38.57,  39.,    40.37,  37.83,  39.,    39.,    39.,    41.24,&
    .,     ERR_VE, 39.,    ERR_VE, ERR_VE, 39.,    39.,    40.37,  39.,    ERR_VE,&
    .,     39.,    39.,    39.,    40.37,  40.37,  ERR_VE, ERR_VE, 39.,    39.,&
    .,     39.,    39.,    39.,    41.24,  40.37/)
! Rate of Rubisco carboxilation at optimum temperature [umol m-2 s-1]
REAL,PARAMETER,DIMENSION(NVEG) :: VOPT1=&
 (/338.38, 200.,   174.75, 200.,   206.86, 175.81, 200.,   200.,   200.,   338.38,&
       .,  ERR_VE, 200.,   ERR_VE, ERR_VE, 200.,   200.,   206.86, 200.,   ERR_VE,&
       .,  200.,   200.,   200.,   206.86, 206.86, ERR_VE, ERR_VE, 200.,   200.,&
       .,  200.,   200.,   200.,   200.,   206.86/)
!
!--- fixed (do not change in the course of the year) parameters
!---1-------2-------3-------4-------5-------6-------7-------8-------9------10---
! seasonal variation of vegetation cover
REAL,PARAMETER,DIMENSION(NVEG) :: SEASF =&
 (/0.6,    0.1,    0.1,    0.3,    0.3,    0.5,    0.3,    0.,     0.2,    0.6,&
    .1,    0.0,    0.4,    0.0,    0.0,    0.2,    0.3,    0.2,    0.08,   0.,&
    .2,    0.17,   0.95,   0.,     0.3,    0.4,    0.,     0.,     0.1,    0.1,&
    .1,    0.1,    0.1,    0.1,    0.6,    0.3/)
! maximum fractional vegetation cover
REAL,PARAMETER,DIMENSION(NVEG) :: VEGC = &
 (/.85,    0.8,    0.8,    0.8,    0.8,    0.9,    0.8,    0.1,    0.6,    0.95,&
    .35,   0.0,    0.8,    0.,     0.,     0.8,    0.8,    0.8,    0.1,    0.0,&
    .0,    0.17,   1.0,    0.8,    0.95,   0.59,   0.0,    0.0,    0.9,    0.95,&
    .9,    0.95,   0.9,    0.9,    0.85,   0.9/)
! maximum LAI
REAL,PARAMETER,DIMENSION(NVEG) :: XLA = &
 (/6.0,    2.0,    6.0,    6.0,    6.0,    6.0,    6.0,    0.1,    6.0,    6.0,&
    .0,    0.0,    6.0,    0.0,    0.0,    6.0,    6.0,    6.0,    6.0,    0.0,&
    .0,    0.0,    2.0,    2.888,  0.0,    0.0,    0.0,    0.0,    3.0,    3.5,&
    .0,    3.0,    2.0,    3.0,    6.0,    7.0/)
! LAI yearly excursion
REAL,PARAMETER,DIMENSION(NVEG) :: XLAI0  =&
 (/5.5,    1.5,    1.0,    5.0,    5.,     1.0,    5.5,    0.0,    5.5,    5.5,&
    .5,    0.0,    5.5,    0.0,    0.0,    1.0,    5.0,    3.0,    3.0,    0.0,&
    .5,    2.13,   1.5,    2.5,    5.5,    3.5,    0.0,    0.0,    0.5,    0.5,&
    .5,    0.5,    0.5,    0.5,    5.5,    5./)
! C3 pathway
REAL,PARAMETER,DIMENSION(NVEG) :: PVMAX25I_1 =&
 (/35.,    31.,    44.,    45.,    33.,    51.,    31.,    ERR_VE, 31.,    35.,&
   ERR_VE, ERR_VE, 31.,    ERR_VE, ERR_VE, 44.,    31.,    33.,    33.,    ERR_VE,&
     .,    33.,    33.,    33.,    33.,    33.,    ERR_VE, ERR_VE, 33.,    33.,&
     .,    33.,    33.,    33.,    35.,    33./)
! C3 pathway
REAL,PARAMETER,DIMENSION(NVEG) :: MRS1_1 =&
 (/9.,     9.,     6.,     6.,     9.,     9.,     9.,     ERR_VE, 9.,     9.,&
   ERR_VE, ERR_VE, 9.,     ERR_VE, ERR_VE, 9.,     9.,     9.,     9.,     ERR_VE,&
    .,     9.,     9.,     9.,     9.,     9.,     ERR_VE, ERR_VE, 9.,     9.,&
    .,     9.,     9.,     9.,     9.,     9./)
```

```
! C4 pathway
REAL,PARAMETER,DIMENSION(NVEG) :: PVMAX25I_0 =&
 (/35.,    33.,    ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, 33.,    ERR_VE, ERR_VE,&
     .,    ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE,&
   ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, 33.,    ERR_VE, ERR_VE, ERR_VE, ERR_VE,&
   ERR_VE, ERR_VE, ERR_VE, ERR_VE, 35.,    ERR_VE/)
!NB: for vineyards (LVEG=26) it has been set PVMAX25I_0=33 in analogy with short grass
(LVEG=2) to use Medlyn photosynthesys param.
! C4 pathway
REAL,PARAMETER,DIMENSION(NVEG) :: MRS1_0 =&
 (/5.,     5.,     ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, 5.,     ERR_VE, ERR_VE,&
    .,     ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE,&
   ERR_VE, ERR_VE, ERR_VE, ERR_VE, ERR_VE, 5.,     ERR_VE, ERR_VE, ERR_VE, ERR_VE,&
   ERR_VE, ERR_VE, ERR_VE, ERR_VE, 5.,     ERR_VE/)
!NB: for vineyards (LVEG=26) it has been set MRS1_0=5 in analogy with short grass (LVEG=2) to
use Medlyn photosynthesys param.
! C4 pathway index (=1 means no, =0 means yes) - currently only Lveg=1,2,7,8,11,19 (and 26)
can have PTW=0
INTEGER,PARAMETER,DIMENSION(NVEG) :: PTW1 =&
 (/0,      0,      1,      1,      1,      1,      0,      0,      1,      1,&
    ,      1,      1,      1,      1,      1,      1,      1,      0,      1,&
    ,      1,      1,      1,      1,      0,      1,      1,      1,      1,&
    ,      1,      1,      1,      0,      1/)
!NB: for vineyards (LVEG=26) it has been set PTW=0 to use Medlyn photosynthesys param.
LOGICAL,PARAMETER,DIMENSION(NVEG) :: VEGSHR1 =&
 (/.TRUE., .TRUE., .FALSE.,.FALSE.,.FALSE.,.FALSE.,.TRUE., .FALSE.,.TRUE., .TRUE.,&
   .FALSE.,.FALSE.,.TRUE., .FALSE.,.FALSE.,.TRUE., .TRUE., .FALSE.,.FALSE.,.FALSE.,&
   .TRUE., .TRUE., .TRUE., .TRUE., .FALSE.,.TRUE., .FALSE.,.FALSE.,.TRUE., .TRUE.,&
   .TRUE., .TRUE., .TRUE., .TRUE., .TRUE., .FALSE./)
END MODULE
!******************************************************************************
MODULE SOIL_VALUES
!******************************************************************************
! Module containing soil parameters defined according with the U.S. Dept of
! Agriculture soil textural classes assigned to values of index SLTEX
! (for classes 1-11) and adapted from several sources (for classes 12-17)
!
!        1 -- sand              2 -- loamy sand       3 -- sandy loam
!        4 -- silt loam         5 -- loam             6 -- sandy clay loam
!        7 -- silty clay loam   8 -- clay loam        9 -- sandy clay
!       10 -- silty clay       11 -- clay            12 -- peat
!       13 -- ice              14 -- Grugliasco sand 15 -- Asphalt/concrete
!       16 -- pervious concrete 17 -- stone aggregate
!
! Soil Characteristics (see Clapp & Hornberger, 1978; McCumber & Pielke, 1981;
! Pielke, 1984; Tremback & Kessler, 1985)
!******************************************************************************
! Author: C. Cassardo (15-May-2014)
! Last revision : C. Cassardo (15-May-2014)
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
IMPLICIT NONE
SAVE
INTEGER,PARAMETER :: NS=17      ! Number of soil types
! Parameter B depending on type of soil []
REAL,DIMENSION(NS) :: B_VEC = (/4.05, 4.38, 4.90, 5.30, 5.39, 7.12, 7.75, 8.52,&
   .40, 10.40, 11.40, 7.75, .20, 2.00, 0.33, 0.50, 0.50/)
! Hydraulic conductivity [m2s-1]
REAL,DIMENSION(NS) :: CAPPAETA_S_VEC =(/1.76e-2, 1.563e-2, 3.41e-3, 7.2e-4, 7.e-4,&
  .3e-4, 1.7e-4, 2.5e-4, 2.2e-4, 1.0e-4, 1.3e-4, 8.0e-4, 3.179e-2, 5.55e-3,&
  .5e-7, 8.5e-1, 1.2e-3/)
! Soil porosity [m3/m3]
REAL,DIMENSION(NS) :: ETA_S_VEC = (/.395, .410, .435, .485, .451, .420, .477, .476,&
 .426, .492, .482, .863, .355, .4, .05, .20, .40/)
! Wilting point [m3/m3]
REAL,DIMENSION(NS) :: ETA_WI_VEC = (/.0677, .0750, .1142, .1794, .1547, .1749,&
 .2181, .2498, .2193, .2832, .2864, .3947, .0212, .0677, .0001, .0001, .0001/)
REAL,PARAMETER :: PSIFC = -3.4  ! field capacity [m]
```

```
! Soil moisture potential [m]
REAL,DIMENSION(NS) :: PSIS_VEC = (/-12.1, -9.0, -21.8, -78.6, -47.8, -29.9, -35.6,&
 -63.0, -15.3, -49.0, -40.5, -35.6, -4.8, -18., -1.0, -4.0, -4.0/)
! Soil thermal capacity [Jm-3K-1]
REAL,DIMENSION(NS) :: ROCI_VEC = (/1.465, 1.407, 1.344, 1.273, 1.214, 1.177, 1.319,&
  .227, 1.177, 1.151, 1.088, 2.094, 1.911, 1.465, 1.95, 2.10, 2.10/)
END MODULE

MODULE FIELDS
!******************************************************************************
! Module containing all model fields, and their allocation-deallocation routines
! Author: M. Galli (01-Apr-2011)
! Last revision : C. Cassardo (15-May-2014)
!CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
USE CONSTANTS
IMPLICIT NONE
REAL,DIMENSION(:),ALLOCATABLE :: BSOIL
REAL,DIMENSION(:),ALLOCATABLE :: BSOIL1
REAL,DIMENSION(:),ALLOCATABLE :: CAPPAETA
REAL,DIMENSION(:),ALLOCATABLE :: CAPPAETAM
REAL,DIMENSION(:),ALLOCATABLE :: CAPPAETA_S
REAL,DIMENSION(:),ALLOCATABLE :: CAPPAETA_S1
REAL,DIMENSION(:),ALLOCATABLE :: CLAY
REAL,DIMENSION(:),ALLOCATABLE :: DELTAWI
REAL,DIMENSION(:),ALLOCATABLE :: DEP_SOI
REAL,DIMENSION(:),ALLOCATABLE :: DIFFSUM
REAL,DIMENSION(:),ALLOCATABLE :: DIFFSUMM
REAL,DIMENSION(:),ALLOCATABLE :: DIFFVT
REAL,DIMENSION(:),ALLOCATABLE :: DIFFVTM
REAL,DIMENSION(:),ALLOCATABLE :: ETA_FC
REAL,DIMENSION(:),ALLOCATABLE :: ETA_FC1
REAL,DIMENSION(:),ALLOCATABLE :: ETA_I
REAL,DIMENSION(:),ALLOCATABLE :: ETA_S
REAL,DIMENSION(:),ALLOCATABLE :: ETA_S1
REAL,DIMENSION(:),ALLOCATABLE :: ETA_W
REAL,DIMENSION(:),ALLOCATABLE :: ETA_WI
REAL,DIMENSION(:),ALLOCATABLE :: ETA_WI1
REAL,DIMENSION(:),ALLOCATABLE :: EVATRA
REAL,DIMENSION(:),ALLOCATABLE :: FRP_OCC
REAL,DIMENSION(:),ALLOCATABLE :: MID_LAY_DEP
REAL,DIMENSION(:),ALLOCATABLE :: MOIS_POT
REAL,DIMENSION(:),ALLOCATABLE :: MOIS_POT_SAT
REAL,DIMENSION(:),ALLOCATABLE :: MOIS_POT_SAT1
REAL,DIMENSION(:),ALLOCATABLE :: OM
REAL,DIMENSION(:),ALLOCATABLE :: P_ETA_I
REAL,DIMENSION(:),ALLOCATABLE :: P_ETA_W
REAL,DIMENSION(:),ALLOCATABLE :: P_TEM_SOI
REAL,DIMENSION(:),ALLOCATABLE :: PW
REAL,DIMENSION(:),ALLOCATABLE :: PWI
REAL,DIMENSION(:),ALLOCATABLE :: ROC
REAL,DIMENSION(:),ALLOCATABLE :: ROC1
REAL,DIMENSION(:),ALLOCATABLE :: ROCIVOL
REAL,DIMENSION(:),ALLOCATABLE :: SAND
REAL,DIMENSION(:),ALLOCATABLE :: SILT
REAL,DIMENSION(:),ALLOCATABLE :: SOILFLUX
REAL,DIMENSION(:),ALLOCATABLE :: SR_SOI
REAL,DIMENSION(:),ALLOCATABLE :: SR_SOI_I
REAL,DIMENSION(:),ALLOCATABLE :: SSR_FIE_CAPC
REAL,DIMENSION(:),ALLOCATABLE :: TEM_SOI
REAL,DIMENSION(:),ALLOCATABLE :: TEM_SOI_I
REAL,DIMENSION(:),ALLOCATABLE :: THEDIFM
INTEGER,DIMENSION(MDAT_IN)::IC
INTEGER,DIMENSION(:),ALLOCATABLE :: NSO
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_CAPPAETA
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_CAPPAETAM
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_DIFFSUM
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_DIFFSUMM
```

```
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_DIFFVT
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_DIFFVTM
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_ETA_I
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_ETA_W
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_EVATRA
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_FRP_OCC
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_MOIS_POT
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_SOILFLUX
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_ROC
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_ROCIVOL
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_SR_SOI
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_TEM_SOI
LOGICAL,DIMENSION(:),ALLOCATABLE :: LAYERS_OUT_THEDIFM
INTEGER,PARAMETER::ALLOCATE_REALS=111
INTEGER,PARAMETER::ALLOCATE_INTEGERS=112
INTEGER,PARAMETER::ALLOCATE_LOGICALS=113
INTEGER,PARAMETER::DEALLOCATE_REALS=ALLOCATE_REALS
INTEGER,PARAMETER::DEALLOCATE_INTEGERS=ALLOCATE_INTEGERS
INTEGER,PARAMETER::DEALLOCATE_LOGICALS=ALLOCATE_LOGICALS

CONTAINS

SUBROUTINE ALLOCATE_FIELDS(ALLOCATE_WHAT,ALLOCATE_HOWMUCH,INDERR)

IMPLICIT NONE
INTEGER,INTENT(IN)::ALLOCATE_WHAT,ALLOCATE_HOWMUCH
INTEGER,INTENT(OUT)::INDERR

INDERR=0

IF(ALLOCATE_WHAT==ALLOCATE_REALS)THEN
 ALLOCATE(BSOIL(ALLOCATE_HOWMUCH),&
          BSOIL1(ALLOCATE_HOWMUCH),&
          CAPPAETA(ALLOCATE_HOWMUCH),&
          CAPPAETAM(ALLOCATE_HOWMUCH),&
          CAPPAETA_S(ALLOCATE_HOWMUCH),&
          CAPPAETA_S1(ALLOCATE_HOWMUCH),&
          CLAY(ALLOCATE_HOWMUCH),&
          DELTAWI(ALLOCATE_HOWMUCH),&
          DEP_SOI(ALLOCATE_HOWMUCH),&
          DIFFSUM(ALLOCATE_HOWMUCH),&
          DIFFSUMM(ALLOCATE_HOWMUCH),&
          DIFFVT(ALLOCATE_HOWMUCH),&
          DIFFVTM(ALLOCATE_HOWMUCH),&
          ETA_FC(ALLOCATE_HOWMUCH),&
          ETA_FC1(ALLOCATE_HOWMUCH),&
          ETA_I(ALLOCATE_HOWMUCH),&
          ETA_S(ALLOCATE_HOWMUCH),&
          ETA_S1(ALLOCATE_HOWMUCH),&
          ETA_W(ALLOCATE_HOWMUCH),&
          ETA_WI(ALLOCATE_HOWMUCH),&
          ETA_WI1(ALLOCATE_HOWMUCH),&
          EVATRA(ALLOCATE_HOWMUCH),&
          FRP_OCC(ALLOCATE_HOWMUCH),&
          MID_LAY_DEP(ALLOCATE_HOWMUCH),&
          MOIS_POT(ALLOCATE_HOWMUCH),&
          MOIS_POT_SAT(ALLOCATE_HOWMUCH),&
          MOIS_POT_SAT1(ALLOCATE_HOWMUCH),&
          OM(ALLOCATE_HOWMUCH),&
          P_ETA_I(ALLOCATE_HOWMUCH),&
          P_ETA_W(ALLOCATE_HOWMUCH),&
          P_TEM_SOI(ALLOCATE_HOWMUCH),&
          PW(ALLOCATE_HOWMUCH),&
          PWI(ALLOCATE_HOWMUCH),&
          ROC(ALLOCATE_HOWMUCH),&
          ROC1(ALLOCATE_HOWMUCH),&
          ROCIVOL(ALLOCATE_HOWMUCH),&
```

```
          SAND(ALLOCATE_HOWMUCH),&
          SILT(ALLOCATE_HOWMUCH),&
          SOILFLUX(ALLOCATE_HOWMUCH),&
          SR_SOI(ALLOCATE_HOWMUCH),&
          SR_SOI_I(ALLOCATE_HOWMUCH),&
          SSR_FIE_CAPC(ALLOCATE_HOWMUCH),&
          TEM_SOI(ALLOCATE_HOWMUCH),&
          TEM_SOI_I(ALLOCATE_HOWMUCH),&
          THEDIFM(ALLOCATE_HOWMUCH),&
          STAT=INDERR)
ELSE IF(ALLOCATE_WHAT==ALLOCATE_INTEGERS)THEN
 ALLOCATE(NSO(ALLOCATE_HOWMUCH),STAT=INDERR)
ELSE IF(ALLOCATE_WHAT==ALLOCATE_LOGICALS)THEN
 ALLOCATE(LAYERS_OUT_CAPPAETA(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_CAPPAETAM(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_DIFFSUM(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_DIFFSUMM(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_DIFFVT(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_DIFFVTM(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_ETA_I(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_ETA_W(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_EVATRA(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_FRP_OCC(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_MOIS_POT(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_ROC(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_ROCIVOL(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_SOILFLUX(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_SR_SOI(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_TEM_SOI(ALLOCATE_HOWMUCH),&
          LAYERS_OUT_THEDIFM(ALLOCATE_HOWMUCH),&
               STAT=INDERR)
ELSE
 PRINT *,'SUBROUTINE ALLOCATE_FIELDS : invalid allocation to be performed'
 PRINT *,'ALLOCATE_WHAT = ',ALLOCATE_WHAT
 PRINT *,'different from any'
 PRINT *,'ALLOCATE_REALS    = ',ALLOCATE_REALS
 PRINT *,'ALLOCATE_INTEGERS = ',ALLOCATE_INTEGERS
 PRINT *,'ALLOCATE_LOGICALS = ',ALLOCATE_LOGICALS
 STOP
END IF

END SUBROUTINE ALLOCATE_FIELDS

SUBROUTINE DEALLOCATE_FIELDS(DEALLOCATE_WHAT,INDERR)
IMPLICIT NONE
INTEGER,INTENT(IN)::DEALLOCATE_WHAT
INTEGER,INTENT(OUT)::INDERR

IF(DEALLOCATE_WHAT==DEALLOCATE_REALS)THEN

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 DEALLOCATE(BSOIL,         BSOIL1,         CAPPAETA,      CAPPAETAM,    CAPPAETA_S,&
            CAPPAETA_S1,   CLAY,           DELTAWI,       DEP_SOI,      DIFFSUM,&
            DIFFSUMM,      DIFFVT,         DIFFVTM,       ETA_FC,       ETA_FC1,&
            ETA_I,         ETA_S,          ETA_S1,        ETA_W,        ETA_WI,&
            ETA_WI1,       EVATRA,         FRP_OCC,       MID_LAY_DEP,  MOIS_POT,&
            MOIS_POT_SAT,  MOIS_POT_SAT1,  OM,            P_ETA_I,      P_ETA_W,&
            P_TEM_SOI,     PW,             PWI,           ROC,          ROC1,&
            ROCIVOL,       SAND,           SILT,          SOILFLUX,     SR_SOI,&
            SR_SOI_I,      SSR_FIE_CAPC,   TEM_SOI_I,     TEM_SOI,      THEDIFM,&
            STAT=INDERR)
ELSE IF(DEALLOCATE_WHAT==DEALLOCATE_INTEGERS)THEN
 DEALLOCATE(NSO,STAT=INDERR)
ELSE IF(DEALLOCATE_WHAT==DEALLOCATE_LOGICALS)THEN
 DEALLOCATE(LAYERS_OUT_CAPPAETA,  LAYERS_OUT_CAPPAETAM,  LAYERS_OUT_DIFFSUM,&
            LAYERS_OUT_DIFFSUMM,  LAYERS_OUT_DIFFVT,     LAYERS_OUT_DIFFVTM,&
            LAYERS_OUT_ETA_I,     LAYERS_OUT_ETA_W,      LAYERS_OUT_EVATRA,&
```

```
            LAYERS_OUT_FRP_OCC,   LAYERS_OUT_MOIS_POT,   LAYERS_OUT_ROC,&
            LAYERS_OUT_ROCIVOL,   LAYERS_OUT_SOILFLUX,   LAYERS_OUT_SR_SOI,&
            LAYERS_OUT_TEM_SOI,   LAYERS_OUT_THEDIFM,&
            STAT=INDERR)
ELSE
 PRINT *,'SUBROUTINE DEALLOCATE_FIELDS : invalid deallocation to be performed'
 PRINT *,'DEALLOCATE_WHAT = ',DEALLOCATE_WHAT
 PRINT *,'different from any'
 PRINT *,'DEALLOCATE_REALS    = ',DEALLOCATE_REALS
 PRINT *,'DEALLOCATE_INTEGERS = ',DEALLOCATE_INTEGERS
 PRINT *,'DEALLOCATE_LOGICALS = ',DEALLOCATE_LOGICALS
 STOP
END IF

END SUBROUTINE DEALLOCATE_FIELDS

END MODULE FIELDS

!*******************************************************************************
! Main model program
PROGRAM UTOPIA
!*******************************************************************************
USE ECOCLIMAP_CONSTANTS
USE CONSTANTS
USE FIELDS
IMPLICIT NONE

! parameters
INTEGER :: MITER ! number of time steps among two input records

! real variable declarations
REAL :: AIR_DENS
REAL :: ALAT
REAL :: ALB_FH
REAL :: ALB_SD
REAL :: ALB_SN
REAL :: ALB_SNT
REAL :: ALB_TOT
REAL :: ALON
REAL :: BAL_FLU_F
REAL :: BAL_FLU_G
REAL :: BAL_FLU_SN
REAL :: BAL_FLU_TOT
REAL :: C_DREN
REAL :: CAPPAMIXF
REAL :: CAPPAMIXG
REAL :: CL_HIG_STEP
REAL :: CL_LOW_CUR
REAL :: CL_LOW_NEW
REAL :: CL_LOW_OLD
REAL :: CL_LOW_STEP
REAL :: CL_TOT_CUR
REAL :: CL_TOT_NEW
REAL :: CL_TOT_OLD
REAL :: CL_TOT_STEP
REAL :: CLEG
REAL :: CO2_CON_STEP
REAL :: CO2_CON_NEW
REAL :: CO2_CON_CUR
REAL :: COND_F_DRY
REAL :: COND_F_WET
REAL :: COND_G_DRY
REAL :: COND_G_WET
REAL :: CONDUCT_AH
REAL :: CONDUCT_AM
REAL :: CONDUCT_AV
REAL :: CONDUCT_B
```

```
REAL :: CONDUCT_D
REAL :: CONDUCT_D_SN
REAL :: CONDUCT_F
REAL :: CONF_SN_F
REAL :: CONF_SN_G
REAL :: CONF_SN_SRF
REAL :: COVER_SN
REAL :: COVER_SN_F
REAL :: COVER_SN_G
REAL :: CUMTOT
REAL :: D0_VEGPAR
REAL :: DELT
REAL :: DELTAZ
REAL :: DENS_SN
REAL :: DF ! varies between 0.9 and 1.3, default 1.0
REAL :: DRAG_H
REAL :: DRAG_M
REAL :: DRAG_V
REAL :: DTOT
REAL :: EPS
REAL :: EPSF
REAL :: EPSG
REAL :: ESSE
REAL :: EVA_A
REAL :: EVA_F
REAL :: EVA_F_D
REAL :: EVA_F_W
REAL :: EVA_G
REAL :: EVA_G_D
REAL :: EVA_G_W
REAL :: EVA_SN
REAL :: EVA_SN_F
REAL :: EVA_SN_G
REAL :: GAMMA
REAL :: GLORAD_NEW
REAL :: GLORAD_CUR
REAL :: GLORAD_OLD
REAL :: GR_THEO
REAL :: GRO_CAR_ASS_RAT ! gross rate of carbon assimilation by photosynthesis [umol/m2s]
REAL :: H_OBS_ALL
REAL :: H_OBS_WIND
REAL :: HAJ
REAL :: HAV
REAL :: HAZE_IND
REAL :: HDJ
REAL :: HDV
REAL :: HEI_SN
REAL :: HM_SN
REAL :: HW_SN
REAL,DIMENSION(MDAT_IN) :: I_INT
REAL,DIMENSION(MDAT_IN) :: I_SLO
REAL :: JOPT
REAL :: LAI_CUR
REAL :: LAI_NEW
REAL :: LAI_OLD
REAL :: LAI_STEP
REAL :: LAI_STEP0
REAL :: LEA_WAT_CON
REAL :: LHF_A
REAL :: LHF_F
REAL :: LHF_F_D
REAL :: LHF_F_W
REAL :: LHF_G
REAL :: LHF_G_D
REAL :: LHF_G_W
REAL :: LHF_SN
REAL :: LHF_SN_F
```

```
REAL :: LHF_SN_G
REAL :: LITUFA2
REAL :: LONRAD_CUR
REAL :: LONRAD_NEW
REAL :: MRS
REAL :: NET_CAR_ASS_RAT ! net rate of carbon assimilation by photosynthesis [umol/m2s]
REAL :: NR_A
REAL :: NR_F
REAL :: NR_G
REAL :: NR_L_F_D
REAL :: NR_L_F_U
REAL :: NR_L_SN_D
REAL :: NR_L_SN_U
REAL :: NR_SN
REAL :: P_HM_SN
REAL :: P_HW_SN
REAL :: P_LEA_WAT_CON
REAL :: P_SRF_ICE_CON
REAL :: P_SRF_WAT_CON
REAL :: P_TEMP_F_STEP
REAL :: P_TEMP_SN
REAL :: PREC_F
REAL :: PREC_G
REAL :: PREC_STEP
REAL :: PREC_NEW
REAL :: PREC_CUR
REAL :: PREC_OLD
REAL :: PRECSN
REAL :: PRES_A_STEP
REAL :: PRS_AIR_CUR
REAL :: PRS_AIR_NEW
REAL :: PRS_AIR_OLD
REAL :: PVMAX25
REAL :: Q_A_F
REAL :: Q_AIR_CUR
REAL :: Q_AIR_NEW
REAL :: Q_AIR_OLD
REAL :: Q_SP_A_STEP
REAL :: Q_RAIN_F
REAL :: Q_RAIN_G
REAL :: QUOTA
REAL :: QSF
REAL :: QSG
REAL :: QSG_RH
REAL :: RAD_L_D
REAL :: RAD_L_F_D
REAL :: RAD_L_F_U
REAL :: RAD_L_FG_D
REAL :: RAD_L_FG_U
REAL :: RAD_L_G_D
REAL :: RAD_L_G_TOT_D
REAL :: RAD_L_G_TOT_U
REAL :: RAD_L_G_U
REAL :: RAD_L_SN_F_D
REAL :: RAD_L_SN_F_U
REAL :: RAD_L_SN_G_D
REAL :: RAD_L_SN_G_U
REAL :: RAD_L_SN_U
REAL :: RAD_L_U
REAL :: RAD_S_D
REAL :: RAD_S_F_D
REAL :: RAD_S_F_U
REAL :: RAD_S_G_D
REAL :: RAD_S_G_U
REAL :: RAD_S_SN_D
REAL :: RAD_S_SN_F_D
REAL :: RAD_S_SN_F_U
```

```
REAL :: RAD_S_SN_G_D
REAL :: RAD_S_SN_G_U
REAL :: RAD_S_SN_U
REAL :: RAD_S_U
REAL :: REL_LEA_WAT_CON
REAL :: REL_SRF_ICE_CON
REAL :: REL_SRF_WAT_CON
REAL :: RES_RAT          ! mitocondrial respiration [umol/m2s]
REAL :: RESIST_AH
REAL :: RESIST_AM
REAL :: RESIST_AV
REAL :: RESIST_B
REAL :: RESIST_D
REAL :: RESIST_D_SN
REAL :: RESIST_F
REAL :: RESIST_F_MIN
REAL :: RESIST_F_PHOTO
REAL :: RESIST_LEAF
REAL :: RESIST_SRF
REAL :: RESIST_SRF_SN
REAL :: RGL_NOILHAN
REAL :: RH_AIR_CUR
REAL :: RH_AIR_NEW
REAL :: RH_AIR_OLD
REAL :: RH_STEP
REAL :: ROOT_DEP
REAL :: RW ! varies between 0 and 0.7 g/g, default 0.0
REAL :: SHF_A
REAL :: SHF_F
REAL :: SHF_G
REAL :: SHF_SN
REAL :: SHF_SN_F
REAL :: SHF_SN_G
REAL :: SO_SRAT_RO_DE
REAL :: SOL_ALFA
REAL :: SRF_ICE_CON
REAL :: SRF_WAT_CON
REAL :: SSR_FIE_CAP
REAL :: SSR_WI
REAL :: SUM_DRAI
REAL :: SUM_DRAI_TMP
REAL :: SUM_EVAP
REAL :: SUM_EVAP_TMP
REAL :: SUM_MIN_LW_TMP
REAL :: SUM_PREC
REAL :: SUM_PREC_SN
REAL :: SUM_PREC_TMP
REAL :: SUM_RUNO
REAL :: SUM_RUNO_TMP
REAL :: SUM_SOI_HEA_STO
REAL :: TEM_AIR_CUR
REAL :: TEM_AIR_NEW
REAL :: TEM_AIR_OLD
REAL :: TEMP_A_F
REAL :: TEMP_A_STEP
REAL :: TEMP_DEW_A_STEP
REAL :: TEMP_F_STEP
REAL :: TEMP_ROOT
REAL :: TEMP_SN
REAL(KIND=DBL_PREC) :: TIM_E
REAL :: TIMESTEP_SEC
REAL :: TOPTJ
REAL :: TOPTV
REAL :: TORTUOS
REAL :: UPWP
REAL :: USTAR
REAL :: VEG_COV_CUR
```

```
REAL :: VEG_COV_NEW
REAL :: VEG_COV_OLD
REAL :: VEG_COV_STEP
REAL :: VEG_COV_STEP0
REAL :: VEG_HEI_CUR
REAL :: VEG_HEI_NEW
REAL :: VEG_HEI_OLD
REAL :: VEG_HEI_STEP
REAL :: VEG_HEI_STEP0
REAL :: VEL_MOD_CUR
REAL :: VEL_MOD_NEW
REAL :: VEL_MOD_STEP
REAL :: VEL_U_CUR
REAL :: VEL_U_NEW
REAL :: VEL_U_OLD
REAL :: VEL_U_STEP
REAL :: VEL_V_CUR
REAL :: VEL_V_NEW
REAL :: VEL_V_OLD
REAL :: VEL_V_STEP
REAL :: VOPT
REAL :: VPWP
REAL :: Z0_H
REAL :: Z0_H_ECO
REAL :: Z0_M
REAL :: Z0_M_ECO
REAL :: Z0_SN
REAL :: Z0_V
REAL :: ZER_DIS_LEV

! integer variable declarations
INTEGER :: BINARY_RECORD_LENGTH_INPUT
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT1
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT2
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT3
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT4
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT5
INTEGER :: BINARY_RECORD_LENGTH_OUTPUT6
INTEGER,DIMENSION(3) :: DATA_INI
INTEGER,DIMENSION(3) :: DATA_CUR
INTEGER :: I
INTEGER :: IAN
INTEGER :: IANNEW
INTEGER,DIMENSION(MDAT_IN) :: IC_IN
INTEGER :: IER
INTEGER :: IGI
INTEGER :: IGINEW
INTEGER :: IME
INTEGER :: IMENEW
INTEGER :: IMI
INTEGER :: IMINEW
INTEGER :: INDERR
INTEGER :: INN1
INTEGER :: INNI
INTEGER :: INNO
INTEGER :: INP_SOIL
INTEGER :: IORA
INTEGER :: IORANEW
INTEGER :: JCOUNT
INTEGER :: KDATA
INTEGER :: LVEG
INTEGER :: MIN_FREEZE
INTEGER :: N1
INTEGER :: NCODFILE
INTEGER :: NDATI
INTEGER :: NDECADE
INTEGER :: NGDECADE
```

```
INTEGER :: NGDECADETOT
INTEGER :: JU_DAY
INTEGER :: NLAT
INTEGER :: NLON
INTEGER :: NUM_SOI_LAY
INTEGER :: NTIPOSUOLO
INTEGER :: NTIPOSUOLOECO
INTEGER,DIMENSION(2) :: ORA_INI
INTEGER :: PTW

! logical variable declarations
LOGICAL :: ECO_CHOICE
LOGICAL :: ILOGALB
LOGICAL :: ILOGFUN
LOGICAL :: ILOGCOE
LOGICAL :: L360DAYS_CALENDAR
LOGICAL :: LDATA
LOGICAL :: LFREEZE
LOGICAL :: LLONGWFLAG   !Set by READ_INP, becomes true if long wave data are avaiable
LOGICAL :: LMEDLYN ! If true, it activates Medlyn parameterization
LOGICAL :: LPHOTO  ! if true, it activates photosynthesis parameterization
LOGICAL :: LSNOW
LOGICAL :: LSNOW2
LOGICAL :: LTCAN_NEW
LOGICAL :: LUTCFLAG !Added by Marco Galli, (19/7/2007)
LOGICAL :: LVAP
LOGICAL,DIMENSION(NSUDYN) :: OUTPUT_VARIABLES
LOGICAL :: LNAMELIST_PAR
LOGICAL :: VEGFLAG !Set by READ_INP,becomes true if H data are avaiable
LOGICAL :: VEGSHR ! flag for vegetation shrinking during winter season

! character variable declarations
CHARACTER(len=50) :: ACCESSO  !Added by Marco Galli, 9 october 2008
CHARACTER(len=80),DIMENSION(12) :: &
 CME = (/'January   ', 'February  ', 'March     ', 'April     ',&
         'May       ', 'June      ', 'July      ', 'August    ',&
         'September ', 'October   ', 'November  ', 'December  '/)
CHARACTER(len=100) :: DIRIN
CHARACTER(len=100) :: DIROU
CHARACTER(len=100) :: DIRPA
CHARACTER(len=300) :: FILEERR
CHARACTER(len=300) :: FILEFZ1
CHARACTER(len=300) :: FILEFZ2
CHARACTER(len=300) :: FILEFZ3
CHARACTER(len=300) :: FILEIN
CHARACTER(len=300) :: FILEOU1
CHARACTER(len=300) :: FILEOU2
CHARACTER(len=300) :: FILEOU3
CHARACTER(len=300) :: FILEOU4
CHARACTER(len=300) :: FILEOU5
CHARACTER(len=300) :: FILEOU6
CHARACTER(len=300) :: FILEPA
CHARACTER(len=300) :: FILENML
CHARACTER(len=50) :: FORMATO
CHARACTER(len=5000) :: FORMAT_OUTPUT
CHARACTER(len=90) :: PREF
CHARACTER(len=300) :: STRING
CHARACTER(len=100),DIMENSION(NSUDYN) :: VARIABLE_NAME
CHARACTER(len=30) :: CATEGORY

! variables for CPUtime and runtime
INTEGER :: CLCK_COUNTS_BEG, CLCK_COUNTS_END, CLCK_RATE
REAL ::  BEG_CPU_TIME, END_CPU_TIME

! initialize runtime and CPUtine counters
CALL SYSTEM_CLOCK ( CLCK_COUNTS_BEG, CLCK_RATE )
CALL CPU_TIME (BEG_CPU_TIME)
```

```
LMEDLYN=.TRUE.
LPHOTO=.FALSE.
print *,'LMEDLYN set to true at beginning of UTOPIA'

! initializations
DIRPA='../par/' ! the default directory of the parameter file
! if the name of the .par file is not specified, the default name will be used
CALL GETARG(1,STRING)

IF(LEN_TRIM(STRING)==0)THEN
 FILEPA = 'utopia.par'  !default name of parameter file
 FILENML = 'utopia.nml' !default name of parameter namelist file
 LNAMELIST_PAR=.TRUE.
ELSE
 IF(STRING(LEN_TRIM(STRING)-2:LEN_TRIM(STRING))=='nml'.OR.&
    STRING(LEN_TRIM(STRING)-2:LEN_TRIM(STRING))=='NML')THEN
  READ(STRING,'(A)')FILENML
  LNAMELIST_PAR=.TRUE.
 ELSE
  READ(STRING,'(A)')FILEPA
  LNAMELIST_PAR=.FALSE.
 END IF
END IF

! initializations of Ecoclimap variables
VAR_ECO_VAR(:,:)=ERR_ECO
VAR_ECO_FIX(:)=ERR_ECO

! screen writings
PRINT *,"University of Torino model of land Processes Interaction with Atmosphere"

!management of a namelist as a configuration file
IF(LNAMELIST_PAR)THEN
 INQUIRE(FILE=TRIM(FILENML),EXIST=LNAMELIST_PAR)
 IF(.NOT.LNAMELIST_PAR)THEN
  INQUIRE(FILE=TRIM(DIRPA)//TRIM(FILENML),EXIST=LNAMELIST_PAR)
  IF(LNAMELIST_PAR)THEN
   FILENML=TRIM(DIRPA)//TRIM(FILENML)
  END IF
 END IF
END IF

filepar_read:IF(LNAMELIST_PAR)THEN

 CALL SETUP_NAMELIST_OUTPUT(FILENML,ACCESSO,DIRIN,DIROU,FORMATO,PREF,ECO_CHOICE,&
  ILOGALB,ILOGCOE,ILOGFUN,L360DAYS_CALENDAR,LDATA,LFREEZE,LUTCFLAG,LTCAN_NEW,&
  LVAP,INP_SOIL,KDATA,LVEG,MIN_FREEZE,MITER,NCODFILE,NDATI,NUM_SOI_LAY,&
  DATA_INI,ORA_INI,ALB_FH,ALAT,ALB_SD,ALON,ESSE,GAMMA,C_DREN,CLEG,D0_VEGPAR,ROOT_DEP,&
  EPSF,EPSG,VEG_HEI_STEP,HEI_SN,LAI_STEP,QUOTA,RESIST_F_MIN,LITUFA2,VEG_COV_STEP,TORTUOS,
  H_OBS_ALL,H_OBS_WIND,&
  INDERR,PTW)

  IF(INDERR/=0)THEN
   PRINT *,'Errors occurred in processing file'
   PRINT *,TRIM(FILENML)
   PRINT *,'to setup UTOPIA with namelist-formatted input.'
   STOP
  END IF

ELSE filepar_read

!Set default values. DO NOT EDIT THESE VALUES; specify these variables in .par file
ILOGALB = .false. !.true. means soil albedo is fixed, no call to ALBEDO
ILOGFUN = .true. ! .true. means haze function is activated, call to HAZE
LDATA = .false. ! .false. means date changes every day, .true. meanx date fixed
LVAP = .true. ! .true. means water vapor effects in the soil will be evaluated
```

```
L360DAYS_CALENDAR=.FALSE. !.FALSE. means normal 365/366 days calendar is used
LTCAN_NEW=.TRUE.! .true. means new vegetation temperature scheme is used
MITER=120   ! number of iterations within a model time step
PTW=1
CO2_CON_STEP=ERR_VE
SUM_PREC_SN=0.

! .par file opening and reading of parameters
N1=LEN_TRIM(DIRPA)
IF(LEN_TRIM(STRING)==0)THEN
 OPEN(UNIT=1,FILE=DIRPA(1:N1)//FILEPA,STATUS='OLD',FORM='FORMATTED',IOSTAT=INDERR)
ELSE
 OPEN(UNIT=1,FILE=FILEPA,STATUS='OLD',FORM='FORMATTED',IOSTAT=INDERR)
END IF
IF (inderr /=0 ) THEN
 PRINT *,"An error occurred opening file ",TRIM(FILEPA)
 STOP 'Check parameter file FILEPA'
ENDIF

! read the parameters and the initial data
READ(1,*,ERR=9999)

! read the directory in which input data file (dirpin/filein.INP) is stored
READ(1,'(A)',ERR=9999)DIRIN

! when a series of "**********" is encountered, it jumps to the end
IF (INDEX(DIRIN(1:10),'*') /= 0) GOTO 9999
READ(1,*)

! read the directory in which output data file (dirou/fileou*) will be stored
READ(1,'(A)')DIROU
READ(1,*)

! file name (without extensions) name used for FILEIN and FILEOU*
READ(1,'(A)')PREF
READ(1,*)

! LFREEZE: logical variable, if true will create the "freezing file"
! MIN_FREEZE: minutes between which two different "frozen files" are written
READ(1,*) LFREEZE, MIN_FREEZE
READ(1,*)

! KDATA: number of observations in one hour - the time interval, in minutes per
! hour, between two consecutive observations is 60/KDATA
! NCODFILE: code for the format and the type (see routine MEMORYOUT) of output
! file data:
!           1) possible formats are: NCODFILE<0 -> unformatted, >0 -> formatted
!           2) |NCODFILE|=0 -> no output files
!              |NCODFILE|=1 -> 5 output files with extensions: _OUT.INP,
!                              _OUT.RAD, _OUT.THE, _OUT.HYD, _OUT.RUM
!              |NCODFILE|=2 -> 6 output files with extensions: _OUT.INP,
!                      _OUT.RAD, _OUT.THE, _OUT.HYD, _OUT.RUM, _OUT.LAM
!              |NCODFILE|=3 -> 1 output file  with extension:  _OUT.LAM
!Marco, Riccardo: (17/4/2009): added the possibility to set MITER. Check of
!valid values of KDATA and NCODFILE

READ(1,'(A)')STRING
KDATA=-999
NCODFILE=-999
READ(STRING,*,IOSTAT=INDERR) KDATA, NCODFILE, MITER
IF(INDERR/=0)THEN
 IF(KDATA<=0.OR.NCODFILE<-3.OR.NCODFILE>3.OR.NCODFILE==0.OR.MITER<=0)THEN
  PRINT *,"Invalid line in the .PAR file; check KDATA, NCODFILE, MITER"
  STOP
 END IF
END IF
```

```
IF (NCODFILE < 0)THEN  !file coding
 FORMATO='UNFORMATTED'
 ACCESSO='DIRECT'
ELSE
 FORMATO='FORMATTED'
 ACCESSO='SEQUENTIAL'
END IF

! check about consistency of KDATA and MIN_FREEZE
IF (MIN_FREEZE > 0) THEN
 IF (MOD(MIN_FREEZE,(60/KDATA)) /= 0) THEN
  WRITE(90,*) 'MIN_FREEZE=',MIN_FREEZE,' inconsistent with KDATA=',KDATA,&
              ' and set to ZERO'
  MIN_FREEZE=0
 ENDIF
ENDIF
READ(1,*)

! ILOGCOE = if true, vegetation parameters from VEGPAR* routines will be used;
! LVEG = vegetation parameter code (extracted from BATS, see W&HS);
! if ILOGCOE is false, the following vegetation parameters will be read
!   VEG_COV_STEP = vegetation cover [0-1];
!   D0_VEGPAR = dimension of second axis of leaf (m);
!   ALB_FH = vegetation albedo [0-1];
!   RESIST_F_MIN = minimum stomatal resistance (s/m);
!   LAI_STEP = leaf area index (m2/m2);
!   VEG_HEI_STEP = vegetation heigth (m);
!   EPSF = vegetation emissivity [0-1];
!   ROOT_DEP = root depth (m);
READ(1,*) ILOGCOE,LVEG

IF (.NOT.ILOGCOE) READ(1,*) VEG_COV_STEP0,D0_VEGPAR,ALB_FH,RESIST_F_MIN,LAI_STEP0,
VEG_HEI_STEP0,EPSF,&
 ROOT_DEP

READ(1,*)
READ(1,*) INP_SOIL  ! INP_SOIL - What soil scheme to use (see utopia2010.par)
READ(1,*)

DF = 1.0 ! density factor, from 0.9 to 1.3, default 1.0
RW = 0.0 ! gravel factor, from 0.0 to 0.7, default 0.0
! read number of soil layers NUM_SOI_LAY and previous two factors
! check that NUM_SOI_LAY is not be undersized
READ(1,'(A)') STRING
READ(STRING,*,IOSTAT=INDERR) NUM_SOI_LAY, DF, RW
!! check DF
IF (DF < 0.9 .OR. DF > 1.3) THEN
 PRINT *,'Error: DF out of interval 0.9-1.3'
 PRINT *,'DF = ', DF
 STOP 'Please correct DF'
ENDIF
! check RW
IF (RW > 0.7) THEN
 PRINT *,'Error: RW > 0.7'
 PRINT *,'RW = ', RW
 STOP 'Please correct RW'
ENDIF
! check that NUM_SOI_LAY is not be undersized
IF (NUM_SOI_LAY > NSOILMAX) THEN
 PRINT *,'Error: NUM_SOI_LAY > NSOILMAX'
 PRINT *,'NUM_SOI_LAY = ', NUM_SOI_LAY, ' - NSOILMAX = ', NSOILMAX
 STOP 'Please correct NSOILMAX and NUM_SOI_LAY'
ENDIF
READ(1,*)
! eventual deallocation and reallocation of soil variables (useful when FILEPA
! contains more than one site)
IF (ALLOCATED(BSOIL)) DEALLOCATE(BSOIL)
```

```
ALLOCATE(BSOIL(NUM_SOI_LAY))
IF (ALLOCATED(BSOIL1)) DEALLOCATE(BSOIL1)
ALLOCATE(BSOIL1(NUM_SOI_LAY))
IF (ALLOCATED(CAPPAETA)) DEALLOCATE(CAPPAETA)
ALLOCATE(CAPPAETA(NUM_SOI_LAY))
IF (ALLOCATED(CAPPAETAM)) DEALLOCATE(CAPPAETAM)
ALLOCATE(CAPPAETAM(NUM_SOI_LAY))
IF (ALLOCATED(CAPPAETA_S)) DEALLOCATE(CAPPAETA_S)
ALLOCATE(CAPPAETA_S(NUM_SOI_LAY))
IF (ALLOCATED(CAPPAETA_S1)) DEALLOCATE(CAPPAETA_S1)
ALLOCATE(CAPPAETA_S1(NUM_SOI_LAY))
IF (ALLOCATED(CLAY)) DEALLOCATE(CLAY)
ALLOCATE(CLAY(NUM_SOI_LAY))
IF (ALLOCATED(DELTAWI)) DEALLOCATE(DELTAWI)
ALLOCATE(DELTAWI(NUM_SOI_LAY)) !...soil freezing
IF (ALLOCATED(DEP_SOI)) DEALLOCATE(DEP_SOI)
ALLOCATE(DEP_SOI(NUM_SOI_LAY))
IF (ALLOCATED(DIFFSUM)) DEALLOCATE(DIFFSUM)
ALLOCATE(DIFFSUM(NUM_SOI_LAY))
IF (ALLOCATED(DIFFSUMM)) DEALLOCATE(DIFFSUMM)
ALLOCATE(DIFFSUMM(NUM_SOI_LAY))
IF (ALLOCATED(DIFFVT)) DEALLOCATE(DIFFVT)
ALLOCATE(DIFFVT(NUM_SOI_LAY))
IF (ALLOCATED(DIFFVTM)) DEALLOCATE(DIFFVTM)
ALLOCATE(DIFFVTM(NUM_SOI_LAY))
IF (ALLOCATED(ETA_FC)) DEALLOCATE(ETA_FC)
ALLOCATE(ETA_FC(NUM_SOI_LAY))
IF (ALLOCATED(ETA_FC1)) DEALLOCATE(ETA_FC1)
ALLOCATE(ETA_FC1(NUM_SOI_LAY))
IF (ALLOCATED(ETA_I)) DEALLOCATE(ETA_I)
ALLOCATE(ETA_I(NUM_SOI_LAY))
IF (ALLOCATED(ETA_S)) DEALLOCATE(ETA_S)
ALLOCATE(ETA_S(NUM_SOI_LAY))
IF (ALLOCATED(ETA_S1)) DEALLOCATE(ETA_S1)
ALLOCATE(ETA_S1(NUM_SOI_LAY))
IF (ALLOCATED(ETA_W)) DEALLOCATE(ETA_W) !...soil freezing
ALLOCATE(ETA_W(NUM_SOI_LAY))    !...soil freezing
IF (ALLOCATED(ETA_WI)) DEALLOCATE(ETA_WI)
ALLOCATE(ETA_WI(NUM_SOI_LAY))
IF (ALLOCATED(ETA_WI1)) DEALLOCATE(ETA_WI1)
ALLOCATE(ETA_WI1(NUM_SOI_LAY))
IF (ALLOCATED(EVATRA)) DEALLOCATE(EVATRA)
ALLOCATE(EVATRA(NUM_SOI_LAY))
IF (ALLOCATED(FRP_OCC)) DEALLOCATE(FRP_OCC)
ALLOCATE(FRP_OCC(NUM_SOI_LAY))
IF (ALLOCATED(MID_LAY_DEP)) DEALLOCATE(MID_LAY_DEP)
ALLOCATE(MID_LAY_DEP(NUM_SOI_LAY))
IF (ALLOCATED(MOIS_POT)) DEALLOCATE(MOIS_POT)
ALLOCATE(MOIS_POT(NUM_SOI_LAY))
IF (ALLOCATED(MOIS_POT_SAT)) DEALLOCATE(MOIS_POT_SAT)
ALLOCATE(MOIS_POT_SAT(NUM_SOI_LAY))
IF (ALLOCATED(MOIS_POT_SAT1)) DEALLOCATE(MOIS_POT_SAT1)
ALLOCATE(MOIS_POT_SAT1(NUM_SOI_LAY))
IF (ALLOCATED(NSO)) DEALLOCATE(NSO)
ALLOCATE(NSO(NUM_SOI_LAY))
IF (ALLOCATED(OM)) DEALLOCATE(OM)
ALLOCATE(OM(NUM_SOI_LAY))
IF (ALLOCATED(P_ETA_I)) DEALLOCATE(P_ETA_I)
ALLOCATE(P_ETA_I(NUM_SOI_LAY))
IF (ALLOCATED(P_ETA_W)) DEALLOCATE(P_ETA_W) !...soil freezing
ALLOCATE(P_ETA_W(NUM_SOI_LAY))   !...soil freezing
IF (ALLOCATED(P_TEM_SOI)) DEALLOCATE(P_TEM_SOI)
ALLOCATE(P_TEM_SOI(NUM_SOI_LAY))
IF (ALLOCATED(PW)) DEALLOCATE(PW)
ALLOCATE(PW(NUM_SOI_LAY))      !...soil freezing
IF (ALLOCATED(PWI)) DEALLOCATE(PWI)
ALLOCATE(PWI(NUM_SOI_LAY))     !...soil freezing
```

```
IF (ALLOCATED(ROC)) DEALLOCATE(ROC)
ALLOCATE(ROC(NUM_SOI_LAY))
IF (ALLOCATED(ROC1)) DEALLOCATE(ROC1)
ALLOCATE(ROC1(NUM_SOI_LAY))
IF (ALLOCATED(ROCIVOL)) DEALLOCATE(ROCIVOL)
ALLOCATE(ROCIVOL(NUM_SOI_LAY))
IF (ALLOCATED(SAND)) DEALLOCATE(SAND)
ALLOCATE(SAND(NUM_SOI_LAY))
IF (ALLOCATED(SILT)) DEALLOCATE(SILT)
ALLOCATE(SILT(NUM_SOI_LAY))
IF (ALLOCATED(SOILFLUX)) DEALLOCATE(SOILFLUX)
ALLOCATE(SOILFLUX(NUM_SOI_LAY))
IF (ALLOCATED(SSR_FIE_CAPC)) DEALLOCATE(SSR_FIE_CAPC)
ALLOCATE(SSR_FIE_CAPC(NUM_SOI_LAY))
IF (ALLOCATED(SR_SOI_I)) DEALLOCATE(SR_SOI)
ALLOCATE(SR_SOI(NUM_SOI_LAY))
IF (ALLOCATED(SR_SOI_I)) DEALLOCATE(SR_SOI_I)
ALLOCATE(SR_SOI_I(NUM_SOI_LAY))
IF (ALLOCATED(TEM_SOI)) DEALLOCATE(TEM_SOI)
ALLOCATE(TEM_SOI(NUM_SOI_LAY))
IF (ALLOCATED(TEM_SOI_I)) DEALLOCATE(TEM_SOI_I)
ALLOCATE(TEM_SOI_I(NUM_SOI_LAY))
IF (ALLOCATED(THEDIFM)) DEALLOCATE(THEDIFM)
ALLOCATE(THEDIFM(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_CAPPAETA)) DEALLOCATE(LAYERS_OUT_CAPPAETA)
ALLOCATE(LAYERS_OUT_CAPPAETA(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_CAPPAETAM)) DEALLOCATE(LAYERS_OUT_CAPPAETAM)
ALLOCATE(LAYERS_OUT_CAPPAETAM(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_DIFFSUM)) DEALLOCATE(LAYERS_OUT_DIFFSUM)
ALLOCATE(LAYERS_OUT_DIFFSUM(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_DIFFSUMM)) DEALLOCATE(LAYERS_OUT_DIFFSUMM)
ALLOCATE(LAYERS_OUT_DIFFSUMM(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_DIFFVT)) DEALLOCATE(LAYERS_OUT_DIFFVT)
ALLOCATE(LAYERS_OUT_DIFFVT(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_DIFFVTM)) DEALLOCATE(LAYERS_OUT_DIFFVTM)
ALLOCATE(LAYERS_OUT_DIFFVTM(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_ETA_I)) DEALLOCATE(LAYERS_OUT_ETA_I)
ALLOCATE(LAYERS_OUT_ETA_I(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_ETA_W)) DEALLOCATE(LAYERS_OUT_ETA_W)
ALLOCATE(LAYERS_OUT_ETA_W(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_EVATRA)) DEALLOCATE(LAYERS_OUT_EVATRA)
ALLOCATE(LAYERS_OUT_EVATRA(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_FRP_OCC)) DEALLOCATE(LAYERS_OUT_FRP_OCC)
ALLOCATE(LAYERS_OUT_FRP_OCC(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_MOIS_POT)) DEALLOCATE(LAYERS_OUT_MOIS_POT)
ALLOCATE(LAYERS_OUT_MOIS_POT(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_ROC)) DEALLOCATE(LAYERS_OUT_ROC)
ALLOCATE(LAYERS_OUT_ROC(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_ROCIVOL)) DEALLOCATE(LAYERS_OUT_ROCIVOL)
ALLOCATE(LAYERS_OUT_ROCIVOL(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_SOILFLUX)) DEALLOCATE(LAYERS_OUT_SOILFLUX)
ALLOCATE(LAYERS_OUT_SOILFLUX(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_SR_SOI)) DEALLOCATE(LAYERS_OUT_SR_SOI)
ALLOCATE(LAYERS_OUT_SR_SOI(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_TEM_SOI)) DEALLOCATE(LAYERS_OUT_TEM_SOI)
ALLOCATE(LAYERS_OUT_TEM_SOI(NUM_SOI_LAY))
IF (ALLOCATED(LAYERS_OUT_THEDIFM)) DEALLOCATE(LAYERS_OUT_THEDIFM)
ALLOCATE(LAYERS_OUT_THEDIFM(NUM_SOI_LAY))

! For each soil layer:
! NSO = soil layer type according to the (modified) C&H78 table, see also routine
!       SELECT_SOIL; if NSO<0, soil types defined in the .PAR file are keeped
! TEM_SOI_I = initial soil temperature (K);
! SR_SOI_I = initial soil moisture (kgvoid/kgsoil); soil moisture beginning with Q is
!  expressed in ratio between volumetric soil moisture and porosity
! DEP_SOI = soil depth (m);
```

```
! The user has the possibility to set the following soil parameters: ETA_S1,
! MOIS_POT_SAT1,CAPPAETA_S1,BSOIL1,ETA_WI1,ROC1

ETA_S1=ERR_VE
ETA_FC1=ERR_VE
CAPPAETA_S1=ERR_VE
BSOIL1=ERR_VE
ETA_WI1=ERR_VE
ROC1=ERR_VE
CLAY=ERR_VE
SAND=ERR_VE
SILT=ERR_VE
OM=ERR_VE
DO I=1,NUM_SOI_LAY
 READ(1,'(A)') STRING
 READ(STRING,*,IOSTAT=INDERR) &
  NSO(I),TEM_SOI_I(I),SR_SOI_I(I),DEP_SOI(I),CLAY(I),SAND(I),OM(I),ETA_S1(I),&
  ETA_FC1(I),CAPPAETA_S1(I),BSOIL1(I),ETA_WI1(I),ROC1(I)
!Check if temperature data are in Celsius or in Kelvin and eventual correction
 IF(TEM_SOI_I(I)<100.)THEN
  TEM_SOI_I(I)=TEM_SOI_I(I)+DTK
 END IF
END DO
READ(1,*)

! ALB_SD: bare soil albedo, EPSG bare soil emissivity
! TORTUOS: tortuosity coefficient
! C_DREN: drainage parameter; values: 0 = non-permeable soil, 1 = permeable soil

EPSG=ERR_VE
READ(1,'(A)') STRING
READ(STRING,*,IOSTAT=INDERR)  ALB_SD, TORTUOS, C_DREN, EPSG

READ(1,*)

! DATA_INI: initial date, in the format year, month, day
! ORA_INI: initial time, in the format hour, minutes
! CLEG: time difference from the local hour and the reference time of the time
!       zone (to be set 0 unless there is summertime --> CLEG=1);
! for compatibility between old .par files (with no LUTCFLAG variable) and new .par
!  files, allows to run the model in UTC reference time.
LUTCFLAG=.FALSE.
READ(1,'(A)') STRING
READ(STRING,*,IOSTAT=INDERR)DATA_INI,ORA_INI,CLEG,LUTCFLAG
READ(1,*)

! ALON = station longitude (in degrees and decimals);
! ALAT = station latitude (in degrees and decimals);
! QUOTA = station heigth (in m above sea level);
! H_OBS_ALL = heigth of the observations of temp., humid., rad. and prec.;
! LITUFA2 = turbidity factor (see routine RADIATION);
! ESSE = azimuth of the slope factor;
! GAMMA = zenith of the slope factor;
! H_OBS_WIND = height of the place in which observations of wind are carried out;
READ(1,*) ALON,ALAT,QUOTA,H_OBS_ALL,LITUFA2,ESSE,GAMMA,H_OBS_WIND
READ(1,*)

! if .true. will take vegetation and soil parameters from ECOCLIMAP database
! (which must be installed externally);
READ(1,*) ECO_CHOICE
READ(1,*)

! initial snow heigth
READ(1,*) HEI_SN

READ(1,*,IOSTAT=INDERR)
IF(INDERR==0)THEN
```

```
 READ(1,*,IOSTAT=INDERR)L360DAYS_CALENDAR,ILOGALB,ILOGFUN,LDATA,LVAP,LTCAN_NEW,LPHOTO
END IF

! all parameters have been read from FILEPA
CLOSE(1)

END IF filepar_read

! if the station is out of the ECOCLIMAP grid, ECO_CHOICE is forced FALSE
IF (ECO_CHOICE) THEN
 IF (ALON<ZMINLON .OR. ALON>ZMAXLON .OR. ALAT<ZMINLAT .OR. ALAT>ZMAXLAT) THEN
  PRINT *, 'ATTENTION: POINT OUT OF THE ECOCLIMAP GRID',&
           ' - ECO_CHOICE IMPOSED ** FALSE **'
  ECO_CHOICE = .FALSE.
 END IF
END IF

IF (ECO_CHOICE) THEN

! determination of grid point - regarding latitude, scale is ROTATED!!!
 NLON = NINT((ALON-ZMINLON)*REAL(NRES)/REAL(NSCALE))
 NLAT = NINT((ZMAXLAT-ALAT)*REAL(NRES)/REAL(NSCALE))

!---check of vector bounds
 IF (NLON < 1 .OR. NLON > NI_AVG .OR. NLAT < 1 .OR. NLAT > NJ_AVG) THEN
  PRINT *, 'ATTENTION: POINT OUT OF THE ECOCLIMAP GRID',&
           ' - ECO_CHOICE IMPOSED ** FALSE **'
  ECO_CHOICE = .FALSE.
 END IF
END IF

IF(LUTCFLAG)THEN
 CLEG=CLEG-REAL(FLOOR(ALON/15.))
 IF(MOD(ALON,15.)<EPSIL)THEN
  CLEG=CLEG+1.
 END IF
END IF

! calculation of snow water equiv. (only initial step, with approx. DENS_SN=DENS_SN_MIN)
HM_SN=HEI_SN*DENS_SN_MIN/DENS_WAT

! air density, depending on the altitude of the station (reference=?)
AIR_DENS=DENS_DRY_AIR*exp(-BETA*QUOTA)

!Configuration routine for reading input data files
CALL MEMORYIN_CONFIG(DIRPA,IC_IN,I_INT,I_SLO,NDATI,INDERR)
IF(INDERR/=0)THEN
 PRINT *,"Problems occurred in MEMORYIN_CONFIG. Check its configuration file."
 PRINT *,"INDERR = ",INDERR
 PRINT *,"UTOPIA execution stopped"
 STOP
END IF

!Configuration of the subroutine for writing the output results
CALL MEMORYOUT_CONFIG(DIRPA,FORMAT_OUTPUT,LAYERS_OUT_SR_SOI,&
 LAYERS_OUT_TEM_SOI,LAYERS_OUT_CAPPAETA,LAYERS_OUT_CAPPAETAM,&
 LAYERS_OUT_THEDIFM,LAYERS_OUT_FRP_OCC,LAYERS_OUT_DIFFSUMM,&
 LAYERS_OUT_DIFFSUM,LAYERS_OUT_DIFFVT,LAYERS_OUT_DIFFVTM,&
 LAYERS_OUT_EVATRA,LAYERS_OUT_MOIS_POT,LAYERS_OUT_SOILFLUX,LAYERS_OUT_ROC,&
 LAYERS_OUT_ROCIVOL,LAYERS_OUT_ETA_W,LAYERS_OUT_ETA_I,&
 OUTPUT_VARIABLES,NUM_SOI_LAY,VARIABLE_NAME,INDERR)
IF(INDERR/=0)THEN
 PRINT *,"Problems occurred in MEMORYOUT_CONFIG. Check its configuration file."
 PRINT *,"INDERR = ",INDERR
 PRINT *,"UTOPIA execution stopped"
 STOP
END IF
```

```
! calculation of the dimension of the record to be read in case of binary input
! I just need to reserve the space for 5 integers and NDATI-1 real values;
! ESSE in this context is just a placeholder for a generic real scalar variable
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_INPUT)(NDATI,I=1,5),(ESSE,I=1,NDATI-1)
!Same for output files
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT1)(NDATI,I=1,5),(ESSE,I=1,NSU1)
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT2)(NDATI,I=1,5),(ESSE,I=1,NSU2)
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT3)(NDATI,I=1,5),(ESSE,I=1,NSU3)
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT4)(NDATI,I=1,5),(ESSE,I=1,NSU4)
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT5)(NDATI,I=1,5),(ESSE,I=1,NSU5)
INQUIRE(IOLENGTH=BINARY_RECORD_LENGTH_OUTPUT6)(NDATI,I=1,5),(ESSE,I=1,NSU6)

! files management
INN1=INDEX(PREF(2:80),'  ')

IF (INN1 <= 0) STOP 'WRONG_INPUT_FILE'
INNI=INDEX(DIRIN(2:80),'  ')
INNO=INDEX(DIROU(2:80),'  ')
IF (INNI <= 0 .OR. INNO <= 0) THEN
 PRINT *,'Cannot open files: wrong directory'
 PRINT *,'INNI,DIRIN = ',INNI,DIRIN
 PRINT *,'INNO,DIROU = ',INNO,DIROU
 STOP 'WRONG_DIRECTORY'
END IF
FILEOU1=DIROU(1:INNO)//PREF(1:INN1)//'_INP.OUT'
FILEOU2=DIROU(1:INNO)//PREF(1:INN1)//'_RAD.OUT'
FILEOU3=DIROU(1:INNO)//PREF(1:INN1)//'_HYD.OUT'
FILEOU4=DIROU(1:INNO)//PREF(1:INN1)//'_THE.OUT'
FILEOU5=DIROU(1:INNO)//PREF(1:INN1)//'_RUM.OUT'
FILEOU6=DIROU(1:INNO)//PREF(1:INN1)//'_LAM.OUT'
FILEERR=DIROU(1:INNO)//PREF(1:INN1)//'.ERR'
FILEFZ1=DIROU(1:INNO)//PREF(1:INN1)//'.FRZ'
FILEFZ2=DIROU(1:INNO)//PREF(1:INN1)//'.FRZ'
FILEFZ3=DIROU(1:INNO)//PREF(1:INN1)//'_TMP.FRZ'
FILEIN =DIRIN(1:INNI)//PREF(1:INN1)//'.INT'

VEG_HEI_STEP=-1.
! calculation of fixed (yearly not varying) vegetation parameters from W&HS85
IF (ILOGCOE) CALL VEGPAR_FIX(ALB_FH,D0_VEGPAR,ROOT_DEP,EPSF,HAJ,HAV,HDJ,HDV,VEG_HEI_STEP0,&
 JOPT,LVEG,MRS,NTIPOSUOLO,PTW,RGL_NOILHAN,RESIST_F_MIN,TOPTJ,TOPTV,PVMAX25,VOPT,VEGSHR)

! in the case of ice cap/glacier, inland water, ocean and asphalt (VEG_COV_STEP=0.), the
! bare soil albedo and emissivity are set equal to the values of the corresponding
! land-use categories (LVEG=12,14,15,27)
IF(LVEG==12 .OR. LVEG==14 .OR. LVEG==15 .OR. LVEG==20 .OR. LVEG==27 .OR.&
 LVEG==28)THEN
 ALB_SD=ALB_FH
 IF(EPSG ==ERR_VE)THEN
  EPSG=EPSF
 END IF
END IF

!---determination of Ecoclimap non-changing variables
NTIPOSUOLOECO = -9 ! ecoclimap soil type code
IF (ECO_CHOICE) THEN

!---determination decade or month data
 CALL DECADE(DATA_INI,NDECADE,NGDECADE,NGDECADETOT,NTIME,L360DAYS_CALENDAR)

! fixed variables from ECOCLIMAP database will be put in VAR_ECO_FIX(NUM_ECO_FIX)
 CALL ECOCLIMAP(2,NTIME,NSCALE,DIR_ECO,NUM_ECO_VAR,NUM_ECO_FIX,CODVAR,&
  CODFIX,VAR_ECO_VAR,VAR_ECO_FIX,NLON,NLAT,NDECADE,NI_AVG,ERR_ECO)

! check and eventual substitution of ECOCLIMAP fixed variables
 IF (VAR_ECO_FIX(1) /= ERR_ECO) ALB_SD     = VAR_ECO_FIX(1) ! bare soil albedo
 IF (VAR_ECO_FIX(2) /= ERR_ECO) ALB_FH      = VAR_ECO_FIX(2) ! vegetation albedo
```

```
 IF (VAR_ECO_FIX(4) /= ERR_ECO) ROOT_DEP    = VAR_ECO_FIX(4) ! root depth
 !IF (VAR_ECO_FIX(5) /= ERR_ECO) XXXX     = VAR_ECO_FIX(5) ! soil depth - not used
 IF (VAR_ECO_FIX(6) /= ERR_ECO) RESIST_F_MIN  = VAR_ECO_FIX(6) ! min stomatal resist

! determination of the soil code type from sand and clay fractions; ocean points
! have sand and clay fraction equal to zero: this causes NTIPOSUOLO=4 (silt soil)
! thus if SAND=CLAY=0 we check also if root depth = ERR_ECO: in this case we force
! NTIPOSUOLO=-9 (error). Also check if the simulation has to be stopped because
! the site is over water (sea /lake/...)
 IF (VAR_ECO_FIX(3) /= ERR_ECO .AND. VAR_ECO_FIX(7) /= ERR_ECO) THEN
  CALL SOIL_TYPE1(VAR_ECO_FIX(3),VAR_ECO_FIX(7),NTIPOSUOLOECO)
  IF (VAR_ECO_FIX(4) == ERR_ECO) THEN
   IF (NTIPOSUOLOECO == 4 .AND. VAR_ECO_FIX(3) == 0. .AND.&
       VAR_ECO_FIX(7) == 0.) THEN
     NTIPOSUOLOECO=-9
     PRINT *, 'Point over water, I stop here'
     STOP
   ELSE
     IF (NTIPOSUOLOECO /= 4) NTIPOSUOLOECO=-9
     PRINT *, 'Undefined soil type. I will use the .par initialization'
   ENDIF
  ENDIF
 ENDIF

!if the following line are commented, Ecoclimap soil type is not used
 IF (NTIPOSUOLOECO /= -9) NTIPOSUOLO = NTIPOSUOLOECO

! use soil type in .par file if soil types are negative
 DO I=1,NUM_SOI_LAY
  IF (NSO(I) > 0 ) THEN
   NSO(I) = NTIPOSUOLO
  ELSE
   NSO(I) = ABS(NSO(I))
  ENDIF
 END DO
ENDIF !ECO_CHOICE

! Selection of the soil parameters using the C&H78 dataset

DO I=1,NUM_SOI_LAY
 CALL SELECT_SOIL(NSO(I),ETA_S(I),MOIS_POT_SAT(I),CAPPAETA_S(I),BSOIL(I),&
  ETA_WI(I),ROC(I),SSR_FIE_CAPC(I))
ENDDO

! Selection of soil parameters: possibility of using the parameters set by the user
! The program stops if a value of the soil parameters is invalid

DO I=1,NUM_SOI_LAY

 IF(ETA_S1(I) /= ERR_VE)THEN
  IF(ETA_S1(I)>0 .AND. ETA_S1(I)<1.)THEN
   ETA_S(I)=ETA_S1(I)
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check ETA_S, layer ",I
   STOP
  END IF
 END IF

 IF(ETA_FC1(I) /= ERR_VE)THEN
  IF(ETA_FC1(I)>0 .AND. ETA_FC1(I)<=ETA_S(I))THEN
   ETA_FC(I)=ETA_FC1(I)
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check ETA_FC, layer ",I
   STOP
  END IF
 END IF
```

```
 IF(CAPPAETA_S1(I) /= ERR_VE)THEN
  IF(CAPPAETA_S1(I) > 0.)THEN
   CAPPAETA_S(I)=CAPPAETA_S1(I)/100. !data in table are in dm2/s
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check CAPPAETA_S, layer ",I
   STOP
  END IF
 END IF

 IF(BSOIL1(I) /= ERR_VE)THEN
  IF(BSOIL1(I) > 0.)THEN
   BSOIL(I)=BSOIL1(I)
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check BSOIL, layer ",I
   STOP
  END IF
 END IF

 IF(ETA_WI1(I) /= ERR_VE)THEN
  IF(ETA_WI1(I) > 0. .AND. ETA_WI1(I)<=ETA_S(I))THEN
   ETA_WI(I)=ETA_WI1(I)
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check ETA_WI, layer ",I
   STOP
  END IF
 END IF

 IF(ROC1(I) /= ERR_VE)THEN
  IF(ROC1(I) > 0.)THEN
   ROC(I)=ROC1(I)*1.E6 ! data in table are in uJm-3K-1
  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value in .PAR file; check ROCI, layer ",I
   STOP
  END IF
 END IF

 IF(ETA_S1(I) /= ERR_VE .OR. ETA_FC1(I) /= ERR_VE .OR. BSOIL1(I) &
  /= ERR_VE) THEN
  MOIS_POT_SAT(I)=(-3.4)*(ETA_FC(I)/ETA_S(I))**BSOIL(I)
  SSR_FIE_CAPC(I)=(MOIS_POT_SAT(I)/(-3.4))**(1./BSOIL(I))
 END IF

!sjlim
   OPEN(UNIT=37,FILE=TRIM(DIROU)//TRIM(PREF)//'_PAMORG.TXT',STATUS='UNKNOWN',FORM='FORMATTED',
   ACTION='WRITE',IOSTAT=IER)
   WRITE(37,'(7A)') '========= BSOIL   CAPPAETA_S   ETA_S   MOIS_POT_SAT   ETA_WI   ETA_FC
   ROC'
   WRITE(37,'(A,I1.1,A,F6.2,A,ES10.3,A,F6.4,A,ES10.3,A,F6.4,A,F6.4,A,ES10.3,A,A)') 'Layer ',I,
    ' ',&
    BSOIL(I),' ',CAPPAETA_S(I),' ',ETA_S(I),' ',MOIS_POT_SAT(I),' ',ETA_WI(I),' ',SSR_FIE_CAPC
    (I)*ETA_S(I),' ',ROC(I)

END DO

! CLAY,SAND,OM: routine SAXTON is used to calculate the soil parameters from soil
! soil texture. If some parameter is alredy set, it is not calculated by the
! routine and, if necessary, other soil parameters are evaluated
DO I=1,NUM_SOI_LAY
 IF(CLAY(I)/=ERR_VE .AND. SAND(I)/=ERR_VE .AND. OM(I)/=ERR_VE)THEN

! Check to avoid negative value of soil CLAY,SAND and OM
  IF(CLAY(I)<0. .OR. SAND(I)<0. .OR. OM(I)<0)THEN
   WRITE(*,'(A,I1.1)')"Check CLAY, SAND or OM <0 in .PAR file, layer ",I
   STOP
  END IF

!The sum of CLAY SAND and OM percentages must be less or equal to 100
```

```
  IF(CLAY(I)+SAND(I)+OM(I)<=100.)THEN

!In Saxton routine OM<8% and CLAY<60%: in this case use SELECT_SOIL values
   IF(OM(I)>8. .OR.CLAY(I)>60.)THEN

    IF(OM(I)>8.)THEN
     WRITE(*,'(A,I1.1,A)') 'Error in SAXTON: OM>8% in layer ',I
    END IF

    IF(CLAY(I)>60.)THEN
     WRITE(*,'(A,I1.1,A)') 'Error in SAXTON: CLAY>60% in layer ',I
    END IF

    WRITE(*,'(A)') 'Soil parameters will be taken from SELECT_SOIL'

   END IF

! Evaluation of percentage of silt in the soil layer
   SILT(I)=100.-CLAY(I)-SAND(I)-OM(I)

!Use of empirical relation of Saxton if OM<=8% .AND.CLAY<=60%
   IF(OM(I)<=8. .AND.CLAY(I)<=60.)THEN

    CALL SAXTON(CLAY(I),SAND(I),SILT(I),OM(I),ETA_S1(I),ETA_FC1(I),&
     CAPPAETA_S1(I),BSOIL1(I),ETA_WI1(I),ROC1(I),BSOIL(I),CAPPAETA_S(I),&
     ETA_S(I),MOIS_POT_SAT(I),ETA_WI(I),SSR_FIE_CAPC(I),ROC(I),CATEGORY,DF,RW)
    WRITE(*,'(A,I1.1,A,A,A)') 'Layer ',I,' ',TRIM(CATEGORY),'. Soil'//&
     ' water characteristic taken by empirical relations of SAXTON'
!sjlim
   OPEN(UNIT=36,FILE=TRIM(DIROU)//TRIM(PREF)//'_PAM.TXT',STATUS='UNKNOWN',FORM='FORMATTED',
   ACTION='WRITE',IOSTAT=IER)
   WRITE(36,'(F6.3,A,F6.3,A,F6.3)') CLAY(I),' ',SAND(I),' ',SILT(I)
   WRITE(36,'(A)') '============== BSOIL   CAPPAETA_S   ETA_S   MOIS_POT_SAT   ETA_WI
   ETA_FC   ROC'
   WRITE(36,'(A,I1.1,A,A,A,F6.2,A,ES10.3,A,F6.4,A,ES10.3,A,F6.4,A,F6.4,A,ES10.3,A,A)') 'Layer
   ',I, ' ',TRIM(CATEGORY), ' ',&
    BSOIL(I),' ',CAPPAETA_S(I),' ',ETA_S(I),' ',MOIS_POT_SAT(I),' ',ETA_WI(I),' ',SSR_FIE_CAPC
    (I)*ETA_S(I),' ',ROC(I)
   END IF

  ELSE
   WRITE(*,'(A,I1.1)')"Invalid value, .PAR file: CLAY+SAND+OM>100, layer ",I
   STOP
  END IF

 ELSE IF(CLAY(I)/=ERR_VE .OR. SAND(I)/=ERR_VE .OR. OM(I)/=ERR_VE)THEN
  WRITE(*,'(A,I1.1)')"Invalid CLAY,SAND,OM in .PAR file, layer ",I
  STOP
 END IF
END DO


! model time step determination (KDATA= number of observations in one hour)
DELT=REAL(60/KDATA) ! DELT = time   ! interval between 2 obs in minutes
TIMESTEP_SEC=60./REAL(KDATA)*60./REAL(MITER)    ! TIMESTEP_SEC=DELT/MITER*60 = time btw 2 obs
[s]

! open input/output files
OPEN(UNIT=99,FILE=FILEERR,STATUS='UNKNOWN',FORM='FORMATTED',IOSTAT=IER)
IF(IER/=0)THEN
 PRINT *,'ERROR : problems in opening output files. Check par file'
 PRINT *,'DIRPA  = ',TRIM(DIRPA)
 PRINT *,'FILEPA = ',TRIM(FILEPA)
 STOP
END IF
IF(NCODFILE>0)THEN
 OPEN(UNIT=71,FILE=FILEIN,STATUS='OLD',FORM=FORMATO,ACTION='READ',IOSTAT=IER)
```

```
 IF(IER/=0)THEN
  PRINT *,'ERROR: cannot open input file'
  PRINT *,TRIM(FILEIN)
  STOP
 END IF

 IF (ABS(NCODFILE) >= 1 .AND. ABS(NCODFILE) <= 2) THEN ! standard
  OPEN(UNIT=91,FILE=FILEOU1,STATUS='UNKNOWN',FORM=FORMATO)
  OPEN(UNIT=92,FILE=FILEOU2,STATUS='UNKNOWN',FORM=FORMATO)
  OPEN(UNIT=93,FILE=FILEOU3,STATUS='UNKNOWN',FORM=FORMATO)
  OPEN(UNIT=94,FILE=FILEOU4,STATUS='UNKNOWN',FORM=FORMATO)
  OPEN(UNIT=95,FILE=FILEOU5,STATUS='UNKNOWN',FORM=FORMATO)
 ENDIF

 IF (ABS(NCODFILE) >= 2 .AND. ABS(NCODFILE) <= 3) THEN ! LAM
  OPEN(UNIT=96,FILE=FILEOU6,STATUS='UNKNOWN',FORM=FORMATO)
 ENDIF

ELSE

 OPEN(UNIT=71,FILE=FILEIN,STATUS='OLD',FORM=FORMATO,ACCESS=ACCESSO,&
  RECL=BINARY_RECORD_LENGTH_INPUT,ACTION='READ',IOSTAT=IER)
 IF(IER/=0)THEN
  PRINT *,'ERROR: cannot open input file'
  PRINT *,TRIM(FILEIN)
  STOP
 END IF

 IF (ABS(NCODFILE) >= 1 .AND. ABS(NCODFILE) <= 2) THEN ! standard
  OPEN(UNIT=91,FILE=FILEOU1,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT1)
  OPEN(UNIT=92,FILE=FILEOU2,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT2)
  OPEN(UNIT=93,FILE=FILEOU3,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT3)
  OPEN(UNIT=94,FILE=FILEOU4,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT4)
  OPEN(UNIT=95,FILE=FILEOU5,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT5)
 ENDIF

 IF (ABS(NCODFILE) >= 2 .AND. ABS(NCODFILE) <= 3) THEN ! LAM
  OPEN(UNIT=96,FILE=FILEOU6,STATUS='UNKNOWN',FORM=FORMATO,ACCESS=ACCESSO,&
   RECL=BINARY_RECORD_LENGTH_OUTPUT6)
 ENDIF
END IF

OPEN(UNIT=97,FILE=TRIM(DIROU)//TRIM(PREF)//'_MEM.TXT',STATUS='UNKNOWN',&
 FORM='FORMATTED',ACTION='WRITE')

! initialization of all parameters used during simulation
CALL INIT(ALB_TOT,NUM_SOI_LAY,ALAT,ALON,ESSE,SUM_DRAI,SUM_RUNO,SUM_PREC,SUM_EVAP,&
 SUM_SOI_HEA_STO,DELTAZ,DTOT,SSR_FIE_CAP,SSR_WI,TEMP_F_STEP,TEMP_ROOT,Q_A_F,LEA_WAT_CON,
 SRF_WAT_CON,&
 SRF_ICE_CON,ZER_DIS_LEV,PREC_OLD,PRS_AIR_OLD,GLORAD_OLD,VEL_U_OLD,VEL_V_OLD,TEM_AIR_OLD,
 RH_AIR_OLD,CL_TOT_OLD,CL_LOW_OLD,Q_AIR_OLD,&
 PREC_CUR,PRS_AIR_CUR,GLORAD_CUR,VEL_U_CUR,VEL_V_CUR,TEM_AIR_CUR,RH_AIR_CUR,CL_TOT_CUR,
 CL_LOW_CUR,Q_AIR_CUR,PREC_NEW,PRS_AIR_NEW,GLORAD_NEW,VEL_U_NEW,&
 VEL_V_NEW,TEM_AIR_NEW,RH_AIR_NEW,CL_TOT_NEW,CL_LOW_NEW,Q_AIR_NEW,LSNOW,P_HM_SN,P_HW_SN,
 TEMP_SN,GAMMA,ROOT_DEP,&
 VEG_HEI_STEP,HM_SN,DEP_SOI,MID_LAY_DEP,FRP_OCC,TEM_SOI,P_TEM_SOI,SR_SOI,SSR_FIE_CAPC,ETA_WI,
 ETA_S,&
 TEM_SOI_I,SR_SOI_I,USTAR,Z0_SN,Z0_M,ETA_W,P_ETA_W,ETA_I,HW_SN,DENS_SN,ERR_ECO,Z0_M_ECO,
 Z0_H_ECO,&
 C_DREN,PWI,P_ETA_I,CL_HIG_STEP,EVA_G,EVA_G_W,EVA_F_W,TIM_E,VEG_HEI_OLD,VEG_HEI_CUR,
 VEG_HEI_NEW,LAI_OLD,&
 LAI_CUR,LAI_NEW,VEG_COV_OLD,VEG_COV_CUR,VEG_COV_NEW,CO2_CON_STEP,RESIST_B,CUMTOT)
```

```
! defreezing - in this case variables initialized in INIT are overwritten in FREEZE
IF (LFREEZE) CALL FREEZE(1,FILEFZ1,IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,SRF_WAT_CON
,&
 HW_SN,HM_SN,NUM_SOI_LAY,P_LEA_WAT_CON,P_SRF_WAT_CON,P_HM_SN,P_HW_SN,P_TEMP_F_STEP,&
 P_TEMP_SN,REL_LEA_WAT_CON,REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,SUM_EVAP_TMP,
 SUM_RUNO,SUM_RUNO_TMP,&
 SUM_DRAI,SUM_DRAI_TMP,SUM_SOI_HEA_STO,TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,
 TEMP_ROOT,Q_A_F,Z0_M,&
 ZER_DIS_LEV,ILOGFUN,LSNOW,SR_SOI,P_TEM_SOI,TEM_SOI,ETA_W,P_ETA_W,ETA_I,P_ETA_I,PWI)

!initialization counter (useful to determine the number of every step)
JCOUNT=0

DO !loop for a single observation - main loop
 JCOUNT=JCOUNT+1 ! counter update

! calculation of yearly varying parameters for vegetation from W&HS85 database
 IF (ILOGCOE) CALL VEGPAR_VAR(LVEG,TEMP_ROOT,VEG_COV_STEP0,LAI_STEP0)

! only at the first instant, two input data must be read: NM and NEW
 IF (JCOUNT == 1) THEN

  CALL READ_INP(ALB_TOT,ESSE,GAMMA,NDATI,QUOTA,ALON,ALAT,CLEG,LITUFA2,JU_DAY,&
   TEM_AIR_CUR,PRS_AIR_CUR,Q_AIR_CUR,CL_TOT_CUR,VEL_U_CUR,VEL_V_CUR,PREC_CUR,CL_LOW_CUR,
   GLORAD_CUR,RH_AIR_CUR,LONRAD_CUR,VEG_HEI_CUR,LAI_CUR,&
   VEG_COV_CUR,CO2_CON_CUR,IAN,IME,IGI,IORA,IMI,IER,TIM_E,GR_THEO,KDATA,&
   LLONGWFLAG,L360DAYS_CALENDAR,JCOUNT,ACCESSO,VEGFLAG,IC_IN,I_INT,I_SLO)

! print a message on the screen
  WRITE(*,"(A,I4.4,2('-',I2.2),'; Time: ',I2.2,':',I2.2)") &
                     'Starting time : ',IAN,IME,IGI,IORA,IMI
  JCOUNT=JCOUNT+1
  IF (IER < 0) EXIT ! End or error of file FILEIN

! read the 2nd observation --> variables with suffix NEW
  CALL READ_INP(ALB_TOT,ESSE,GAMMA,NDATI,QUOTA,ALON,ALAT,CLEG,LITUFA2,JU_DAY,&
   TEM_AIR_NEW,PRS_AIR_NEW,Q_AIR_NEW,CL_TOT_NEW,VEL_U_NEW,VEL_V_NEW,PREC_NEW,CL_LOW_NEW,
   GLORAD_NEW,RH_AIR_NEW,LONRAD_NEW,&
   VEG_HEI_NEW,LAI_NEW,VEG_COV_NEW,CO2_CON_NEW,IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,IER,&
   TIM_E,GR_THEO,KDATA,LLONGWFLAG,L360DAYS_CALENDAR,JCOUNT,ACCESSO,VEGFLAG,&
   IC_IN,I_INT,I_SLO)

! data update
  DATA_CUR(1)=IAN
  DATA_CUR(2)=IME
  DATA_CUR(3)=IGI
  IF (IER < 0) EXIT ! End or error of file FILEIN

 ELSE IF (JCOUNT/=1) THEN

  CALL AGGIORNA_NEW(TEM_AIR_OLD,PRS_AIR_OLD,Q_AIR_OLD,CL_TOT_OLD,VEL_U_OLD,VEL_V_OLD,PREC_OLD,
  CL_LOW_OLD,GLORAD_OLD,&
   RH_AIR_OLD,IAN,IME,IGI,IORA,IMI,TEM_AIR_CUR,PRS_AIR_CUR,Q_AIR_CUR,CL_TOT_CUR,VEL_U_CUR,
   VEL_V_CUR,PREC_CUR,CL_LOW_CUR,&
   GLORAD_CUR,RH_AIR_CUR,LONRAD_CUR,VEG_HEI_CUR,LAI_CUR,VEG_COV_CUR,CO2_CON_CUR,TEM_AIR_NEW,
   PRS_AIR_NEW,Q_AIR_NEW,&
   CL_TOT_NEW,VEL_U_NEW,VEL_V_NEW,PREC_NEW,CL_LOW_NEW,GLORAD_NEW,RH_AIR_NEW,LONRAD_NEW,
   VEG_HEI_NEW,LAI_NEW,&
   VEG_COV_NEW,CO2_CON_NEW,IANNEW,IMENEW,IGINEW,IORANEW,IMINEW)

! data update
  DATA_CUR(1)=IAN
  DATA_CUR(2)=IME
  DATA_CUR(3)=IGI

! read the ith observation --> variables with suffix NEW
```

```
  CALL READ_INP(ALB_TOT,ESSE,GAMMA,NDATI,QUOTA,ALON,ALAT,CLEG,LITUFA2,JU_DAY,&
   TEM_AIR_NEW,PRS_AIR_NEW,Q_AIR_NEW,CL_TOT_NEW,VEL_U_NEW,VEL_V_NEW,PREC_NEW,CL_LOW_NEW,
   GLORAD_NEW,RH_AIR_NEW,LONRAD_NEW,&
   VEG_HEI_NEW,LAI_NEW,VEG_COV_NEW,CO2_CON_NEW,IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,IER,&
   TIM_E,GR_THEO,KDATA,LLONGWFLAG,L360DAYS_CALENDAR,JCOUNT,ACCESSO,VEGFLAG,&
   IC_IN,I_INT,I_SLO)
  IF (IER < 0) EXIT ! End or error of file FILEIN

 ENDIF

! if LDATA is .TRUE. force data to not change (simulation of an infinite day)
 IF (LDATA) THEN
  IAN=DATA_INI(1)
  IME=DATA_INI(2)
  IGI=DATA_INI(3)
  IANNEW=DATA_INI(1)
  IMENEW=DATA_INI(2)
  IGINEW=DATA_INI(3)
 ENDIF

! *********** MODEL ******* STARTING ******* POINT
! storage of above calculated parameters prior to calling Ecoclimap
! (in case of re-use later) - suffix "1"
! EPS1=VEG_COV_STEP*EPSF+(1.-VEG_COV_STEP)*EPSG ! weighted mean of emissivity
! EPS2=0.90+0.18*ETA_S(1)*SR_SOI(1) ! Braud et al. (1995)

! update of ecoclimap variables
 IF (ECO_CHOICE) THEN
  CALL DECADE(DATA_CUR,NDECADE,NGDECADE,NGDECADETOT,NTIME,L360DAYS_CALENDAR)

! put fixed variables from ECOCLIMAP database into VAR_ECO_FIX(3,NUM_ECO_FIX)
  CALL ECOCLIMAP(1,NTIME,NSCALE,DIR_ECO,NUM_ECO_VAR,NUM_ECO_FIX,CODVAR,&
   CODFIX,VAR_ECO_VAR,VAR_ECO_FIX,NLON,NLAT,NDECADE,NI_AVG,ERR_ECO)

! linear time interpolation of Ecoclimap parameters at different times (decades)
  CALL INTLIN(NGDECADE,NGDECADETOT,VAR_ECO_VAR,NUM_ECO_VAR,ERR_ECO)

! check and eventual substitution of ECOCLIMAP variable variables
  IF (VAR_ECO_VAR(3,1) /= ERR_ECO) EPS     = VAR_ECO_VAR(3,1) ! emissivity
  !IF (VAR_ECO_VAR(3,2) /= ERR_ECO) XXXX   = VAR_ECO_VAR(3,2) ! greeness, unused
  IF (VAR_ECO_VAR(3,3) /= ERR_ECO) LAI_STEP0= VAR_ECO_VAR(3,3) ! leaf area index
  IF (VAR_ECO_VAR(3,4) /= ERR_ECO) VEG_COV_STEP0 = VAR_ECO_VAR(3,4) ! vegetation cover
  IF (VAR_ECO_VAR(3,5) /= ERR_ECO) Z0_M_ECO  = VAR_ECO_VAR(3,5) ! mom. roughn. len.
  IF (VAR_ECO_VAR(3,6) /= ERR_ECO) Z0_H_ECO  = VAR_ECO_VAR(3,6) ! heat roughn. len.

! check of Ecoclimap Z0_M<= 0.1 m to avoid too large aerodynamic resistances
  IF (Z0_M_ECO > 0.1) THEN
   WRITE(99,*)'Original Ecoclimap value of Z0_M = ',Z0_M_ECO,' changed in 0.1m'
   Z0_M_ECO=0.1
  ENDIF

! check of Ecoclimap Z0_H<=0.1/7.4m to avoid too large aerodynamic resistance
  IF (Z0_H_ECO > 0.0135) THEN
   WRITE(99,*)'Original Ecoclimap value of Z0_H = ',Z0_H_ECO,' changed in 0.0135m'
   Z0_H_ECO=0.0135
  ENDIF

! derivation of UTOPIA variables absent in Ecoclimap: veg. height from Z0_M=0.13 h
  IF (Z0_M_ECO /= ERR_ECO) VEG_HEI_STEP0 = Z0_M_ECO / 0.13

! clever initial value to EPS in any case
 ELSE
  EPS=ERR_ECO
 ENDIF

! readjust vegetation
 IF (VEG_HEI_STEP <0.) THEN
```

```
  VEG_HEI_STEP=VEG_HEI_STEP0
  VEG_COV_STEP=VEG_COV_STEP0
  LAI_STEP=LAI_STEP0
 ENDIF

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! the core of the model is launched
 CALL UTOPIA_DRIVER(IMI, INP_SOIL, IORA, JU_DAY, LVEG, MITER, NUM_SOI_LAY, PTW, &
 AIR_DENS, ALAT, ALB_FH, ALON, &
 C_DREN, CL_LOW_CUR, CL_LOW_NEW, CL_TOT_CUR, CL_TOT_NEW, CLEG, CO2_CON_CUR, CO2_CON_NEW, &
 D0_VEGPAR, DELT, DELTAZ, EPS, ERR_ECO, ESSE, &
 GAMMA, GLORAD_NEW, GLORAD_CUR, &
 HAJ, HAV, HDJ, HDV, JOPT, &
 LAI_CUR, LAI_NEW, LONRAD_CUR, LONRAD_NEW, MRS, &
 PREC_NEW, PREC_CUR, PRS_AIR_CUR, PRS_AIR_NEW, PVMAX25, &
 Q_AIR_CUR, Q_AIR_NEW, &
 RGL_NOILHAN, RESIST_F_MIN, RH_AIR_CUR, RH_AIR_NEW, &
 SSR_FIE_CAP, SSR_WI, &
 TEM_AIR_NEW, TEM_AIR_CUR, TIMESTEP_SEC, TOPTJ, TOPTV, TORTUOS, &
 VEG_COV_CUR, VEG_COV_NEW, VEG_HEI_CUR, VEG_HEI_NEW, VEL_U_CUR, VEL_U_NEW, VEL_V_CUR,
 VEL_V_NEW, &
 VOPT, Z0_H_ECO, Z0_M_ECO, &
 ALB_SD, ALB_SN, ALB_SNT, CO2_CON_STEP, CUMTOT, &
 DENS_SN, EPSF, EPSG, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, &
 EVA_SN_F, EVA_SN_G, H_OBS_ALL, H_OBS_WIND, HEI_SN, HM_SN, HW_SN, &
 LAI_STEP, LAI_STEP0, LEA_WAT_CON, &
 P_HM_SN, P_HW_SN, P_LEA_WAT_CON, P_SRF_ICE_CON, P_SRF_WAT_CON, P_TEMP_SN, &
 SHF_SN_F, SHF_SN_G, SRF_ICE_CON, SRF_WAT_CON, &
 SUM_DRAI, SUM_EVAP, SUM_PREC, SUM_RUNO, SUM_SOI_HEA_STO, &
 TEMP_F_STEP, TEMP_SN, USTAR, &
 VEG_COV_STEP, VEG_COV_STEP0, VEG_HEI_STEP, VEG_HEI_STEP0, &
 Z0_H, Z0_M, Z0_SN, Z0_V, ZER_DIS_LEV, &
 ALB_TOT, BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT, &
 CAPPAMIXF, CAPPAMIXG, &
 CL_HIG_STEP, CL_LOW_STEP, CL_TOT_STEP, COND_F_DRY, COND_F_WET, COND_G_DRY, COND_G_WET, &
 CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F, &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, &
 DRAG_H, DRAG_M, DRAG_V, GRO_CAR_ASS_RAT, HAZE_IND, &
 LHF_A, LHF_F, LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, &
 NET_CAR_ASS_RAT, NR_A, NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, &
 P_TEMP_F_STEP, PREC_F, PREC_G, PRECSN, PREC_STEP, PRES_A_STEP, &
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, &
 RAD_L_D, RAD_L_F_D, RAD_L_F_U, RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D,
 RAD_L_G_TOT_U, &
 RAD_L_G_U, RAD_L_SN_F_D, RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, &
 RAD_S_D, RAD_S_F_D, RAD_S_F_U, RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U,
 &
 RAD_S_SN_G_D, RAD_S_SN_G_U, RAD_S_SN_U, RAD_S_U, &
 REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, &
 RESIST_AH, RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, &
 RESIST_LEAF, RESIST_SRF, RESIST_SRF_SN, RH_STEP, &
 SHF_A, SHF_F, SHF_G, SHF_SN, SO_SRAT_RO_DE, &
 SOL_ALFA, SUM_DRAI_TMP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, SUM_PREC_TMP, SUM_PREC_SN, SUM_RUNO_TMP
 , &
 TEMP_A_F, TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_ROOT, UPWP, &
 VEL_MOD_CUR, VEL_MOD_NEW, VEL_MOD_STEP, VEL_U_STEP, VEL_V_STEP, VPWP, &
 ILOGALB, ILOGFUN, L360DAYS_CALENDAR, LLONGWFLAG, LMEDLYN, LPHOTO, &
 LTCAN_NEW, VEGSHR, &
 LSNOW, LVAP, &
 LSNOW2, &
 NSO, &
 BSOIL, CAPPAETA_S, DEP_SOI, ETA_S, ETA_WI, FRP_OCC,&
 MID_LAY_DEP, MOIS_POT_SAT, ROC, SSR_FIE_CAPC, &
 ETA_I, ETA_W, P_ETA_I, P_ETA_W, P_TEM_SOI,&
 TEM_SOI,&
 CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, DIFFVT, &
 DIFFVTM, EVATRA, MOIS_POT, ROCIVOL, SOILFLUX, SR_SOI, THEDIFM)
```

```
! eventual frezing
 IF (MIN_FREEZE > 0 .AND. MOD(INT(TIM_E),MIN_FREEZE)==0) THEN
  CALL FREEZE(3,FILEFZ3,IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,SRF_WAT_CON,&
   HW_SN,HM_SN,NUM_SOI_LAY,P_LEA_WAT_CON,P_SRF_WAT_CON,P_HM_SN,P_HW_SN,P_TEMP_F_STEP,P_TEMP_SN
   ,&
   REL_LEA_WAT_CON,REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,SUM_EVAP_TMP,SUM_RUNO,
   SUM_RUNO_TMP,&
   SUM_DRAI,SUM_DRAI_TMP,SUM_SOI_HEA_STO,TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,
   TEMP_ROOT,&
   Q_A_F,Z0_M,ZER_DIS_LEV,ILOGFUN,LSNOW,SR_SOI,P_TEM_SOI,TEM_SOI,ETA_W,P_ETA_W,&
   ETA_I,P_ETA_I,PWI)
 ENDIF

! storage of results
 CALL DYNAMIC_MEMORYOUT(IANNEW, IGINEW, IMENEW, IMINEW, IORANEW, JCOUNT, NCODFILE, NUM_SOI_LAY
 ,&
 CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, &
 DIFFVT, DIFFVTM, ETA_I, ETA_W, EVATRA, FRP_OCC, MOIS_POT, ROC, ROCIVOL, SOILFLUX, SR_SOI, &
 TEM_SOI, THEDIFM,&
 ALB_SD, ALB_SN, ALB_TOT, BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT,&
 CAPPAMIXF, CAPPAMIXG, CL_LOW_STEP, CL_TOT_STEP, CO2_CON_STEP, COND_F_DRY, COND_F_WET,
 COND_G_DRY,&
 COND_G_WET, CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F
 , &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, DENS_SN, DRAG_H, DRAG_M,
  &
 DRAG_V, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, EVA_SN_F, EVA_SN_G,
 &
 GR_THEO, GRO_CAR_ASS_RAT, HAZE_IND, HEI_SN, HM_SN, HW_SN, LAI_STEP, LEA_WAT_CON, LHF_A, LHF_F
 , &
 LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, NET_CAR_ASS_RAT, NR_A,
  &
 NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, PREC_F, PREC_G, PRES_A_STEP,&
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, RAD_L_D, RAD_L_F_D, RAD_L_F_U, &
 RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D, RAD_L_G_TOT_U, RAD_L_G_U, RAD_L_SN_F_D, &
 RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, RAD_S_F_D, RAD_S_D, RAD_S_F_U,
  &
 RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U, RAD_S_SN_G_D, RAD_S_SN_G_U, &
 RAD_S_SN_U, RAD_S_U, REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, RESIST_AH, &
 RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, RESIST_LEAF,
  &
 RESIST_SRF, RESIST_SRF_SN, RH_STEP, SHF_A, SHF_F, SHF_G, SHF_SN, SHF_SN_F, SHF_SN_G, SOL_ALFA
 , &
 SRF_ICE_CON, SRF_WAT_CON, SUM_DRAI, SUM_DRAI_TMP, SUM_EVAP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, &
 SUM_PREC, SUM_PREC_SN, SUM_PREC_TMP, SUM_RUNO, SUM_RUNO_TMP, SUM_SOI_HEA_STO, TEMP_SN,
 TEMP_A_F, &
 TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_F_STEP, TEMP_ROOT, UPWP, USTAR, VPWP, VEG_COV_STEP, &
 VEG_HEI_STEP, VEL_MOD_STEP,&
 FORMAT_OUTPUT, VARIABLE_NAME, &
 LAYERS_OUT_CAPPAETA, LAYERS_OUT_CAPPAETAM, &
 LAYERS_OUT_DIFFSUM, LAYERS_OUT_DIFFSUMM, LAYERS_OUT_DIFFVT, LAYERS_OUT_DIFFVTM, &
 LAYERS_OUT_ETA_I, LAYERS_OUT_ETA_W, LAYERS_OUT_EVATRA, LAYERS_OUT_FRP_OCC, &
 LAYERS_OUT_MOIS_POT, LAYERS_OUT_SR_SOI, LAYERS_OUT_ROC, LAYERS_OUT_ROCIVOL, &
 LAYERS_OUT_SOILFLUX, LAYERS_OUT_TEM_SOI, LAYERS_OUT_THEDIFM,&
 OUTPUT_VARIABLES)

! write time on screen
 IF ((IORANEW+IMINEW)==0)THEN
  WRITE(*,'(a,i4,a,a3,i3,a,I3)') 'Station '//PREF(1:INN1)//' ',&
   IANNEW,' ',CME(IMENEW)(1:3),IGINEW
 END IF

END DO ! end of main loop

! final freezing
CALL FREEZE(2,FILEFZ2,IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,SRF_WAT_CON,HW_SN,&
```

```
 HM_SN,NUM_SOI_LAY,P_LEA_WAT_CON,P_SRF_WAT_CON,P_HM_SN,P_HW_SN,P_TEMP_F_STEP,P_TEMP_SN,
 REL_LEA_WAT_CON,&
 REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,SUM_EVAP_TMP,SUM_RUNO,SUM_RUNO_TMP,SUM_DRAI,&
 SUM_DRAI_TMP,SUM_SOI_HEA_STO,TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,TEMP_ROOT,Q_A_F,
 Z0_M,&
 ZER_DIS_LEV,ILOGFUN,LSNOW,SR_SOI,P_TEM_SOI,TEM_SOI,ETA_W,P_ETA_W,ETA_I,P_ETA_I,PWI)

! closing of input/output files
CLOSE (UNIT=71)
CLOSE (UNIT=96)
IF (ABS(NCODFILE) >= 1 .AND. ABS(NCODFILE) <= 2) THEN ! standard
 CLOSE(UNIT=91)
 CLOSE(UNIT=92)
 CLOSE(UNIT=93)
 CLOSE(UNIT=94)
 CLOSE(UNIT=95)
ENDIF
IF (ABS(NCODFILE) >= 2 .AND. ABS(NCODFILE) <= 3) THEN ! LAM
 CLOSE(UNIT=96)
ENDIF

PRINT *, "End of simulation"
GOTO 10000
     CONTINUE
PRINT *, "Error in FILEPA"
PRINT *, "Non regular termination"

      CONTINUE
! finalize runtime and CPUtine counters
CALL SYSTEM_CLOCK ( CLCK_COUNTS_END, CLCK_RATE )
CALL CPU_TIME (END_CPU_TIME)

! print runtime and CPUtine counters
PRINT *, "RUN time = ", (CLCK_COUNTS_END - CLCK_COUNTS_BEG) / REAL (CLCK_RATE)
PRINT *, "CPU time", END_CPU_TIME - BEG_CPU_TIME

END PROGRAM UTOPIA

!*************************************************************************
SUBROUTINE ABCI(NUM_SOI_LAY,RAD_S_D,RAD_S_U,TIMESTEP_SEC,P_TEM_SOI,DEP_SOI,DIFFWATER,SRHS,
SRHS0,&
 WAT_SUR_FLU,RAD_L_D,RAD_L_U,LHF_G,SHF_A,MID_LAY_DEP,AA,BB,CC,DD)
!*************************************************************************
! Routine to diagonalize water temperature
!*************************************************************************
! Calls: ICEDIFF, SOLARHS
! - Author : M.W. Qian (28-Nov-2002)
! - Last revision : C. Cassardo (15-May-2014)
!*************************************************************************
USE CONSTANTS
IMPLICIT NONE
INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: AA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: BB
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFWATER
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: LHF_G
REAL,INTENT(IN) :: SHF_A
INTEGER :: I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: RAD_L_D
REAL,INTENT(IN) :: RAD_L_U
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U
```

```
REAL,DIMENSION(NUM_SOI_LAY) :: S
REAL,INTENT(OUT) :: SRHS0
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: SRHS
REAL,INTENT(OUT) :: WAT_SUR_FLU
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP
REAL :: WATERSURFLUX

WAT_SUR_FLU = WATERSURFLUX(RAD_S_D,RAD_S_U,RAD_L_D,RAD_L_U,LHF_G,SHF_A)
CALL SOLARHS(NUM_SOI_LAY,DEP_SOI,RAD_S_D,RAD_S_U,SRHS,SRHS0,MID_LAY_DEP)
CALL ICEDIFF(DIFFWATER,NUM_SOI_LAY)

DO I=1,NUM_SOI_LAY-1
 S(I) = DIFFWATER(I) / (DEP_SOI(I)+DEP_SOI(I+1))
END DO

AA(1)=0.
BB(1)=DEP_SOI(1)/TIMESTEP_SEC+S(1)
CC(1)=-S(1)
DD(1)=DEP_SOI(1)/TIMESTEP_SEC*P_TEM_SOI(1)+WAT_SUR_FLU/ROCI_I& !BAL_FLU_G/ROCI_I
      -S(1)*(P_TEM_SOI(1)-P_TEM_SOI(2))+SRHS(1)/ROCI_I

DO I=2,NUM_SOI_LAY-1
 AA(I)=-S(I-1)
 BB(I)=DEP_SOI(I)/TIMESTEP_SEC+S(I-1)+S(I)
 CC(I)=-S(I)
 DD(I)=DEP_SOI(I)/TIMESTEP_SEC*P_TEM_SOI(I)+S(I-1)*(P_TEM_SOI(I-1)-P_TEM_SOI(I))-&
       S(I)*(P_TEM_SOI(I)-P_TEM_SOI(I+1))+SRHS(I)/ROCI_I
END DO

AA(NUM_SOI_LAY)=-S(NUM_SOI_LAY-1)
BB(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC+S(NUM_SOI_LAY-1)
CC(NUM_SOI_LAY)=0.
DD(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC*P_TEM_SOI(NUM_SOI_LAY)+S(NUM_SOI_LAY-1)*&
 (P_TEM_SOI(NUM_SOI_LAY-1)-P_TEM_SOI(NUM_SOI_LAY))+SRHS(NUM_SOI_LAY)/ROCI_I

RETURN
END SUBROUTINE ABCI

!*****************************************************************************
SUBROUTINE ABCQ(NUM_SOI_LAY,AA,BB,C_DREN,CAPPAETA,CAPPAETAM,CC,DD,&
 DIFFSUMM,DIFFVTM,DEP_SOI,TIMESTEP_SEC,EVA_G,EVATRA,P_INFILTR,P_ETA_W)
!*****************************************************************************
! Routine to evaluate soil moisture
!*****************************************************************************
! Calls: none
! - Author : M. W. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE
INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: AA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: BB
REAL,INTENT(IN) :: C_DREN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETAM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DIFFSUMM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DIFFVTM
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_G
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: EVATRA
INTEGER :: I
REAL,INTENT(IN) :: P_INFILTR
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_ETA_W
```

```
REAL,DIMENSION(NUM_SOI_LAY) :: S
!REAL,DIMENSION(NUM_SOI_LAY) :: S2

DO I=1,NUM_SOI_LAY-1
 S(I)=DIFFSUMM(I)/(DEP_SOI(I)+DEP_SOI(I+1))
! S2(I)=DIFFVTM(I)/(DEP_SOI(I)+DEP_SOI(I+1))
END DO

AA(1)=0.
BB(1)=DEP_SOI(1)/TIMESTEP_SEC+S(1)
CC(1)=-S(1)
DD(1)=DEP_SOI(1)/TIMESTEP_SEC*P_ETA_W(1)-S(1)*(P_ETA_W(1)-P_ETA_W(2))-CAPPAETAM(1)*1.+&
      P_INFILTR-(EVA_G+EVATRA(1))/DENS_WAT !&
!original part with thermal diffusivity
!               -2*S2(1)*(P_TEM_SOI(1)-P_TEM_SOI(2))+CAPPAETAM(1)*DESERTO+&
!                P_INFILTR-(EVA_G+EVATRA(1))/DENS_WAT

DO I=2,NUM_SOI_LAY-1
 AA(I)=-S(I-1)
 BB(I)=DEP_SOI(I)/TIMESTEP_SEC+S(I-1)+S(I)
 CC(I)=-S(I)
 DD(I)=DEP_SOI(I)/TIMESTEP_SEC*P_ETA_W(I)+S(I-1)*(P_ETA_W(I-1)-P_ETA_W(I))-&
       S(I)*(P_ETA_W(I)-P_ETA_W(I+1))+(CAPPAETAM(I-1)-CAPPAETAM(I))*&
        .-EVATRA(I)/DENS_WAT !&
!original part with thermal diffusivity ??? to be checked
!          +2*S2(I-1)*(P_TEM_SOI(i-1)-P_TEM_SOI(I))-2*S2(I)*(P_TEM_SOI(I)-P_TEM_SOI(I+1))
END DO

AA(NUM_SOI_LAY)=-S(NUM_SOI_LAY-1)
BB(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC+S(NUM_SOI_LAY-1)
CC(NUM_SOI_LAY)=0.
DD(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC*P_ETA_W(NUM_SOI_LAY)+&
              S(NUM_SOI_LAY-1)*(P_ETA_W(NUM_SOI_LAY-1)-P_ETA_W(NUM_SOI_LAY))-&
!!!claudioooo              C_DREN*CAPPAETA(NUM_SOI_LAY)+CAPPAETA(NUM_SOI_LAY-1)*1.-&
              C_DREN*CAPPAETA(NUM_SOI_LAY)+CAPPAETAM(NUM_SOI_LAY-1)*1.-&
              EVATRA(NUM_SOI_LAY)/DENS_WAT !&
!original part with thermal diffusivity
!               +2*S2(NUM_SOI_LAY-1)*(P_TEM_SOI(NUM_SOI_LAY-1)-P_TEM_SOI(NUM_SOI_LAY))

RETURN
END SUBROUTINE ABCQ

!************************************************************************
SUBROUTINE ABCT(NUM_SOI_LAY,TIMESTEP_SEC,BAL_FLU_G,BAL_FLU_G_BOT,THEDIFM,DEP_SOI,ROCIVOL,&
 P_TEM_SOI,AA,BB,CC,DD)
!************************************************************************
! Routine to evaluate soil temperature
!************************************************************************
! Calls: none
! - Author : M. W. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!************************************************************************
USE CONSTANTS
IMPLICIT NONE
INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: AA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: BB
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: THEDIFM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,INTENT(IN) :: TIMESTEP_SEC
INTEGER :: I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: BAL_FLU_G
REAL,INTENT(IN) :: BAL_FLU_G_BOT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ROCIVOL
```

```
REAL,DIMENSION(NUM_SOI_LAY) :: S

DO I=1,NUM_SOI_LAY-1
 S(I)=THEDIFM(I)/(DEP_SOI(I)+DEP_SOI(I+1))
END DO

AA(1)=0.
BB(1)=ROCIVOL(1)/TIMESTEP_SEC+S(1)
CC(1)=-S(1)
DD(1)=ROCIVOL(1)/TIMESTEP_SEC*P_TEM_SOI(1)+BAL_FLU_G-S(1)*(P_TEM_SOI(1)-P_TEM_SOI(2))

DO I=2,NUM_SOI_LAY-1
 AA(I)=-S(I-1)
 BB(I)=ROCIVOL(I)/TIMESTEP_SEC+S(I-1)+S(I)
 CC(I)=-S(I)
 DD(I)=ROCIVOL(I)/TIMESTEP_SEC*P_TEM_SOI(I)+S(I-1)*(P_TEM_SOI(I-1)-P_TEM_SOI(I))-&
       S(I)*(P_TEM_SOI(I)-P_TEM_SOI(I+1))
END DO

AA(NUM_SOI_LAY)=-S(NUM_SOI_LAY-1)
BB(NUM_SOI_LAY)=ROCIVOL(NUM_SOI_LAY)/TIMESTEP_SEC+S(NUM_SOI_LAY-1)
CC(NUM_SOI_LAY)=0.
DD(NUM_SOI_LAY)=ROCIVOL(NUM_SOI_LAY)/TIMESTEP_SEC*P_TEM_SOI(NUM_SOI_LAY)+&
    S(NUM_SOI_LAY-1)*(P_TEM_SOI(NUM_SOI_LAY-1)-P_TEM_SOI(NUM_SOI_LAY))-BAL_FLU_G_BOT

RETURN
END SUBROUTINE ABCT

!*****************************************************************************
SUBROUTINE ABCW(NUM_SOI_LAY,VEL_MOD_STEP,ALAT,DEP_SOI,DIFFWATER,P_TEM_SOI,MID_LAY_DEP,SRHS,&
                SRHS0,WAT_SUR_FLU,RAD_S_D,RAD_S_U,RAD_L_D,RAD_L_U,LHF_G,SHF_A,TIMESTEP_SEC,&
                AA,BB,CC,DD)
!*****************************************************************************
! Routine to evaluate water temperature
!*****************************************************************************
! Calls: SOLARHS, WATERDIFF
! - Author : M. W. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE
INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: AA
REAL,INTENT(IN) :: ALAT
REAL,INTENT(IN) :: VEL_MOD_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: BB
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFWATER
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: LHF_G
REAL,INTENT(IN) :: SHF_A
INTEGER :: I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U
REAL,INTENT(IN) :: RAD_L_D
REAL,INTENT(IN) :: RAD_L_U
REAL,DIMENSION(NUM_SOI_LAY) :: S
REAL,INTENT(OUT) :: SRHS0
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: SRHS
REAL :: WATERSURFLUX
REAL,INTENT(OUT) :: WAT_SUR_FLU
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP

WAT_SUR_FLU = WATERSURFLUX(RAD_S_D,RAD_S_U,RAD_L_D,RAD_L_U,LHF_G,SHF_A)
```

```
CALL SOLARHS(NUM_SOI_LAY,DEP_SOI,RAD_S_D,RAD_S_U,SRHS,SRHS0,MID_LAY_DEP)

CALL WATERDIFF( NUM_SOI_LAY,VEL_MOD_STEP,ALAT,DEP_SOI,DIFFWATER,P_TEM_SOI,MID_LAY_DEP)

DO I=1,NUM_SOI_LAY-1
 S(I)=DIFFWATER(I)/(DEP_SOI(I)+DEP_SOI(I+1))
END DO

AA(1)=0.
BB(1)=DEP_SOI(1)/TIMESTEP_SEC+S(1)
CC(1)=-S(1)
DD(1)=DEP_SOI(1)/TIMESTEP_SEC*P_TEM_SOI(1)+WAT_SUR_FLU/ROCI_W-S(1)*(P_TEM_SOI(1)-P_TEM_SOI(2
))+&
      (SRHS0-SRHS(1))/ROCI_W

DO I=2,NUM_SOI_LAY-1
 AA(I)=-S(I-1)
 BB(I)=DEP_SOI(I)/TIMESTEP_SEC+S(I-1)+S(I)
 CC(I)=-S(I)
 DD(I)=DEP_SOI(I)/TIMESTEP_SEC*P_TEM_SOI(I)+S(I-1)*(P_TEM_SOI(I-1)-P_TEM_SOI(I))-S(I)*&
       (P_TEM_SOI(I)-P_TEM_SOI(I+1))+(SRHS(I-1)-SRHS(I))/ROCI_W
END DO

AA(NUM_SOI_LAY)=-S(NUM_SOI_LAY-1)
BB(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC+S(NUM_SOI_LAY-1)
CC(NUM_SOI_LAY)=0.
DD(NUM_SOI_LAY)=DEP_SOI(NUM_SOI_LAY)/TIMESTEP_SEC*P_TEM_SOI(NUM_SOI_LAY)+&
              S(NUM_SOI_LAY-1)*(P_TEM_SOI(NUM_SOI_LAY-1)-P_TEM_SOI(NUM_SOI_LAY))+&
              (SRHS(NUM_SOI_LAY-1)-SRHS(NUM_SOI_LAY))/ROCI_W

RETURN
END SUBROUTINE ABCW

!************************************************************************
SUBROUTINE AGGIORNA_NEW(TEM_AIR_OLD,PRS_AIR_OLD,Q_AIR_OLD,CL_TOT_OLD,VEL_U_OLD,VEL_V_OLD,
PREC_OLD,&
 CL_LOW_OLD,GLORAD_OLD,RH_AIR_OLD,IAN,IME,IGI,IORA,IMI,TEM_AIR_CUR,PRS_AIR_CUR,Q_AIR_CUR,
 CL_TOT_CUR,&
 VEL_U_CUR,VEL_V_CUR,PREC_CUR,CL_LOW_CUR,GLORAD_CUR,RH_AIR_CUR,LONRAD_CUR,VEG_HEI_CUR,LAI_CUR,
 &
 VEG_COV_CUR,CO2_CON_CUR,TEM_AIR_NEW,PRS_AIR_NEW,Q_AIR_NEW,CL_TOT_NEW,VEL_U_NEW,VEL_V_NEW,&
 PREC_NEW,CL_LOW_NEW,GLORAD_NEW,RH_AIR_NEW,LONRAD_NEW,VEG_HEI_NEW,LAI_NEW,&
 VEG_COV_NEW,CO2_CON_NEW,IANNEW,IMENEW,IGINEW,IORANEW,IMINEW)
!************************************************************************
! Routine to store current values of input variables
!************************************************************************
! Calls: none
! - Author : C.Cassardo (11-Jun-1994)
! - Last revision: C. Cassardo (15-May-2014)
!************************************************************************
IMPLICIT NONE
REAL,INTENT(IN) :: PRS_AIR_NEW
REAL,INTENT(INOUT) :: PRS_AIR_CUR
REAL,INTENT(OUT) :: PRS_AIR_OLD
REAL,INTENT(IN) :: CL_LOW_NEW
REAL,INTENT(INOUT) :: CL_LOW_CUR
REAL,INTENT(OUT) :: CL_LOW_OLD
REAL,INTENT(IN) :: CL_TOT_NEW
REAL,INTENT(INOUT) :: CL_TOT_CUR
REAL,INTENT(OUT) :: CL_TOT_OLD
REAL,INTENT(IN) :: CO2_CON_NEW
REAL,INTENT(INOUT) :: CO2_CON_CUR
REAL,INTENT(IN) :: GLORAD_NEW
REAL,INTENT(INOUT) :: GLORAD_CUR
REAL,INTENT(OUT) :: GLORAD_OLD
REAL,INTENT(IN) :: VEG_HEI_NEW
```

```
REAL,INTENT(INOUT) :: VEG_HEI_CUR
INTEGER,INTENT(OUT) :: IAN
INTEGER,INTENT(IN) :: IANNEW
INTEGER,INTENT(OUT) :: IGI
INTEGER,INTENT(IN) :: IGINEW
INTEGER,INTENT(OUT) :: IME
INTEGER,INTENT(IN) :: IMENEW
INTEGER,INTENT(OUT) :: IMI
INTEGER,INTENT(IN) :: IMINEW
INTEGER,INTENT(OUT) :: IORA
INTEGER,INTENT(IN) :: IORANEW
REAL,INTENT(IN) :: LAI_NEW
REAL,INTENT(INOUT) :: LAI_CUR
REAL,INTENT(IN) :: PREC_NEW
REAL,INTENT(INOUT) :: PREC_CUR
REAL,INTENT(OUT) :: PREC_OLD
REAL,INTENT(IN) :: Q_AIR_NEW
REAL,INTENT(INOUT) :: Q_AIR_CUR
REAL,INTENT(OUT) :: Q_AIR_OLD
REAL,INTENT(IN) :: RH_AIR_NEW
REAL,INTENT(INOUT) :: RH_AIR_CUR
REAL,INTENT(OUT) :: RH_AIR_OLD
REAL,INTENT(IN) :: LONRAD_NEW
REAL,INTENT(INOUT) :: LONRAD_CUR
REAL,INTENT(IN) :: VEG_COV_NEW
REAL,INTENT(INOUT) :: VEG_COV_CUR
REAL,INTENT(IN) :: TEM_AIR_NEW
REAL,INTENT(INOUT) :: TEM_AIR_CUR
REAL,INTENT(OUT) :: TEM_AIR_OLD
REAL,INTENT(IN) :: VEL_U_NEW
REAL,INTENT(INOUT) :: VEL_U_CUR
REAL,INTENT(OUT) :: VEL_U_OLD
REAL,INTENT(IN) :: VEL_V_NEW
REAL,INTENT(INOUT) :: VEL_V_CUR
REAL,INTENT(OUT) :: VEL_V_OLD

TEM_AIR_OLD=TEM_AIR_CUR     ! 1
PRS_AIR_OLD=PRS_AIR_CUR   ! 2
Q_AIR_OLD=Q_AIR_CUR     ! 3
CL_TOT_OLD=CL_TOT_CUR     ! 4
VEL_U_OLD=VEL_U_CUR       ! 5
VEL_V_OLD=VEL_V_CUR       ! 6
PREC_OLD=PREC_CUR       ! 7
CL_LOW_OLD=CL_LOW_CUR   ! 8
GLORAD_OLD=GLORAD_CUR     ! 9
RH_AIR_OLD=RH_AIR_CUR !10

TEM_AIR_CUR=TEM_AIR_NEW     ! 1
PRS_AIR_CUR=PRS_AIR_NEW   ! 2
Q_AIR_CUR=Q_AIR_NEW     ! 3
CL_TOT_CUR=CL_TOT_NEW     ! 4
VEL_U_CUR=VEL_U_NEW       ! 5
VEL_V_CUR=VEL_V_NEW       ! 6
PREC_CUR=PREC_NEW       ! 7
CL_LOW_CUR=CL_LOW_NEW   ! 8
GLORAD_CUR=GLORAD_NEW     ! 9
RH_AIR_CUR=RH_AIR_NEW !10
LONRAD_CUR=LONRAD_NEW
VEG_HEI_CUR=VEG_HEI_NEW
LAI_CUR=LAI_NEW
VEG_COV_CUR=VEG_COV_NEW
CO2_CON_CUR=CO2_CON_NEW

IAN=IANNEW
IME=IMENEW
IGI=IGINEW
IORA=IORANEW
```

```
IMI=IMINEW

RETURN
END SUBROUTINE AGGIORNA_NEW

!******************************************************************************
SUBROUTINE ALBEDO(ALB_F,ALB_FH,ALB_G,ALB_SD,ALB_SN,ALB_SNT,ALAT,ALB_TOT,&
 ALON,ESSE,GAMMA,CLEG,TIMESTEP_SEC,HEI_SN,ILOGALB,IORA,&
 IMINUTI,L360DAYS_CALENDAR,LSNOW,JU_DAY,NSOIL,PRECSN,SR_SOI,&
 P_TEM_SOI,P_TEMP_SN,VEG_COV_STEP,SOL_ALFA,SOL_ZEN,COVER_SN)
!******************************************************************************
! This routine calculates all albedoes (bare soil, vegetation, snow)
!******************************************************************************
! Calls: SOL_ANG, WEIGHTMEAN
! - Authors: C.Cassardo, J.J.Ji (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE
REAL,PARAMETER ::  TAU1=86400.    ! MeteoFrance formulation's parameters
REAL,PARAMETER ::  TAUF=0.24      ! MeteoFrance formulation's parameters

REAL,INTENT(OUT) :: ALB_F
REAL,INTENT(IN) :: ALB_FH
REAL,INTENT(OUT) :: ALB_G
REAL :: AFH
REAL :: AFZ
REAL,INTENT(IN) :: ALAT
REAL,INTENT(IN) :: ALB_SD
REAL,INTENT(OUT) :: ALB_SN
REAL,INTENT(INOUT) :: ALB_SNT
REAL :: ALB_SNMAX
REAL,INTENT(OUT) :: ALB_TOT
REAL,INTENT(IN) :: ALON
REAL,INTENT(IN) :: ESSE
REAL,INTENT(IN) :: GAMMA
REAL,INTENT(IN) :: CLEG
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: HEI_SN
LOGICAL,INTENT(IN) :: ILOGALB
INTEGER,INTENT(IN) :: IORA
INTEGER,INTENT(IN) :: IMINUTI
LOGICAL,INTENT(IN) :: L360DAYS_CALENDAR
LOGICAL,INTENT(IN) :: LSNOW
INTEGER,INTENT(IN) :: JU_DAY
INTEGER,INTENT(IN) :: NSOIL
REAL,DIMENSION(NSOIL),INTENT(IN) :: SR_SOI
REAL,INTENT(IN) :: PRECSN
REAL,DIMENSION(NSOIL),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: P_TEMP_SN
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN
REAL,INTENT(OUT) :: SOL_ALFA
REAL,INTENT(OUT) :: SOL_ZEN
REAL :: WEIGHTMEAN

! Evaluation of bare soil albedo ALB_G = AFH + AFZ
! AFH is the portion due to soil moisture (color)
AFZ=0.
IF (ILOGALB) THEN
 ALB_G=ALB_SD ! albedo assumed constant and fixed to ALB_SD, read from .PAR
ELSE
! bare soil albedo SET =water albedo if soil relative wetness>50%
 AFH=AFW
! correction for dry soil: orig. value 0.34 in Pielke changed with (ALB_sd-afw)*2
 IF (SR_SOI(1)<0.5) &
  AFH=ALB_SD-2.*(ALB_SD-AFW)*SR_SOI(1)
```

```
! soil frozen appears dry (Cassardo personal observation) - imposed a linear function
! in order to avoid albedo jumps - thresholds are from -5 to 5 ^C
 IF (P_TEM_SOI(1) < (DTK+5.)) THEN
  IF (P_TEM_SOI(1) > (DTK-5.)) THEN
   AFH=ALB_SD+(AFH-ALB_SD)*(P_TEM_SOI(1)-(DTK-5.))/10.
  ELSE
   AFH=ALB_SD
  ENDIF
 ENDIF

! AFZ is the portion due to solar inclination (solar angle): (> slope, > albedo)
 CALL SOL_ANG(JU_DAY,IORA,IMINUTI,ALON,ALAT,CLEG,SOL_ALFA,L360DAYS_CALENDAR)
! Case -3.0 < SOL_ALFA < 0 is considered to take into account that there are about
! 8 min of radiation before sunrise and after sunset; in this moment AFZ is max
 IF (SOL_ALFA>-3.) THEN
  SOL_ZEN=MIN(90.,(90.-SOL_ALFA))
  AFZ=MAX(0.,((EXP(0.003286*(SOL_ZEN**1.5))-1.)/100.)) ! McCumber (1980)
 ELSE
  AFZ=0.
 END IF

 ALB_G=AFZ+AFH ! total albedo for bare soil
END IF

! Evaluation of vegetation albedo: we sum the value AFZ due to the variation
! of albedo with zenith angle, we subtract the value of 0.03
! which represents the daily diurnal mean of AFZ: this substraction is made
! in order to have the daily mean value of ALB_F = ALB_FH
! 0.03 represents the daily mean of AFZ (needs more accurate evaluation)
ALB_F=ALB_FH+AFZ-0.03

! Evaluation of snow albedo (only if there is snow)
IF (LSNOW) THEN

! calculation of maximum snow albedo, function of temperature
 ALB_SNMAX=ALB_SNMAXX+0.05*((MAX(MIN((DTK-P_TEMP_SN),10.),0.))/10.-1.)

! ALB_SNT is portion of albedo due to snow aging: at beginning (at first snowfall,
! when ALBSN is = ERR_VE, is set to its max value, then it will decrease with time
 IF (ALB_SN<=0.) ALB_SNT=ALB_SNMAX ! <=0 includes the case =0. and =ERR_VE

!ALTERNATIVE 1---albedo as ISBA1 (M□t□ofrance formulation)
!  exponential decrement in case of TEMP_SN=0 C (Douville et al., 1995)
! IF (P_TEMP_SN>=DTK) ALB_SNT=(ALB_SNT-ALB_SNMIN)*EXP(-TAUF*TIMESTEP_SEC/TAU1)+ALB_SNMIN !
! linear decrement in case of TEMP_SN<0 C
! IF (P_TEMP_SN<DTK) ALB_SNT=ALB_SNT-TAUA*TIMESTEP_SEC/TAU1

!ALTERNATIVE 2:---albedo as Sun et al., 1999
 IF (HEI_SN>=0.20) THEN
! exponential decrement in case of deep snow according with (Douville et al., 1995)
  ALB_SNT=ALB_SNMIN+(ALB_SNT-ALB_SNMIN)*EXP(-TAUF*TIMESTEP_SEC/TAU1)
 ELSE
! linear decrement in case of shallow snow depth, but different cofficients
! according with snow temperature
  IF (P_TEMP_SN>=DTK) THEN
   ALB_SNT=ALB_SNT-0.071*TIMESTEP_SEC/TAU1
  ELSE
   ALB_SNT=ALB_SNT-0.006*TIMESTEP_SEC/TAU1
  ENDIF
 ENDIF

! if new snow, the new snow whiteness is adapted from Douville et al. (1995) pag 9
 IF (PRECSN>0.) THEN
! imposed a minimum threshold PRECSNWHITE above which new snow albedo prevails
  IF ((PRECSN*TIMESTEP_SEC)>=PRECSNWHITE) THEN
   ALB_SNT=ALB_SNMAX
```

```
  ELSE
! if few snowfall, albedo is weighted average between few fresh snow and threhold
   ALB_SNT=WEIGHTMEAN(ALB_SNT,ALB_SNMAX,1./(PRECSN*TIMESTEP_SEC),(1./PRECSNWHITE))
  END IF
 END IF

 ALB_SNT=MAX(ALB_SNMIN,MIN(ALB_SNT,ALB_SNMAX))

! total snow albedo evaluated as in the case of vegetation with AFZ due to solar
! angle variation, and subtracting 0.03 which is the daily mean value of AFZ
 ALB_SN=ALB_SNT+AFZ-0.03

ELSE

! if there is not snow, snow albedo is fixed = 0.
 ALB_SN=0. ! in case of no snow
ENDIF

! check that all albedoes are included between 0 and 1
ALB_F=MAX(0.,MIN(1.,ALB_F))
ALB_G=MAX(0.,MIN(1.,ALB_G))
IF (LSNOW) ALB_SN=MAX(0.,MIN(1.,ALB_SN))

! total soil albedo (weighted average between all components)
ALB_TOT=COVER_SN*ALB_SN+(1-COVER_SN)*(VEG_COV_STEP*ALB_F+(1.-VEG_COV_STEP)*ALB_G)

RETURN
END SUBROUTINE ALBEDO

!*****************************************************************************
SUBROUTINE AV_NS_MOI(PRES_A_STEP,BSOIL,CAPPAETA,CAPPAETA_S,CAPPAETAM,&
 THEDIFM,CAPPAMIXF,CAPPAMIXG,CAPPAUNO,CAPPASN,DEP_SOI,DENS_SN,&
 DIFFSUM,DIFFSUMM,DIFFVT,DIFFVTM,ETA_S,MOIS_POT,MOIS_POT_SAT,&
 NUM_SOI_LAY,HEI_SN,VEG_HEI_STEP,INP_SOIL,LAI_STEP,LVAP,LVEG,LSNOW,SR_SOI,&
 P_TEM_SOI,ROC,ROCIVOL,RP_TOTAL,TORTUOS,P_ETA_W,SSR_FIE_CAPC,VEG_COV_STEP,P_ETA_I,NSO)
!*****************************************************************************
! Calculates mean and non-saturated values of all var. depending on soil moisture
!*****************************************************************************
! Calls: SPECUM, WEIGHTMEAN, X_WEI_MEA
! - Author : C. Cassardo (26-Nov-1998)
! - Last revision : C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: PRES_A_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: BSOIL
REAL :: BSOIL12
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CAPPAETA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CAPPAETAM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: THEDIFM
REAL,INTENT(OUT) :: CAPPAMIXF
REAL,INTENT(OUT) :: CAPPAMIXG
REAL,INTENT(OUT) :: CAPPASN
REAL,INTENT(OUT) :: CAPPAUNO
REAL :: DCS
REAL :: DDCS
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,INTENT(IN) :: DENS_SN
REAL :: WAT_VAP_DENS
REAL :: DIFFSOIL
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFSUM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFSUMM
REAL :: DIFFVE
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFVT
```

```
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFVTM
REAL :: DIFFWV
REAL :: DIFFUSIV
REAL :: ES
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: MOIS_POT
REAL :: FS12
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MOIS_POT_SAT
REAL,INTENT(IN) :: VEG_HEI_STEP
REAL,INTENT(IN) :: HEI_SN
INTEGER :: I
INTEGER,INTENT(IN) :: INP_SOIL
REAL,INTENT(IN) :: LAI_STEP
LOGICAL,INTENT(IN) :: LSNOW
LOGICAL,INTENT(INOUT) :: LVAP
LOGICAL :: LVDIF
INTEGER, INTENT(IN) :: LVEG
LOGICAL :: LVWT
INTEGER,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: NSO
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_W
REAL :: PF1
REAL :: PF12
REAL :: PIPPO
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SR_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL :: Q12
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SSR_FIE_CAPC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ROC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: ROCIVOL
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: RP_TOTAL
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: TORTUOS
REAL :: X_WEI_MEA
REAL :: WEIGHTMEAN

! calculation of non-saturated variables
DO I=1,NUM_SOI_LAY

! Correction to avoid numerical instability when SR_SOI is close to 0.01. The min
! SR_SOI used to calculate MOIS_POT in dry soils is 0.03. Bonanno (2009) p 88-97
 MOIS_POT(I)=MOIS_POT_SAT(I)/(MAX(SR_SOI(I),0.03)**BSOIL(I))

! modification of thermal capacity to take into account soil freezing
! - ROC = volumetric heat capacity of "dry" unvegetated soil
! - 4.186E3 = volumetric heat capacity of water (=DENS_WAT*CPwater)
! - 1.911E3 = volumetric heat capacity of ice
! - 1.080 = volumetric heat capacity of air
 DCS=(1.-ETA_S(I))*ROC(I)+P_ETA_W(I)*4.186E3*ROW+P_ETA_I(I)*1.911E3*ROI +&
  (ETA_S(I)-P_ETA_W(I)-P_ETA_I(I))*1.030

 IF (INP_SOIL >= 3 .AND. INP_SOIL <= 5) THEN

  IF((P_TEM_SOI(I) >= TFRZ2) .AND. (P_TEM_SOI(I) <= TFRZ1)) THEN

   DDCS = 0.
! Viterbo et al. (1999)
   IF (INP_SOIL == 3 .OR. INP_SOIL == 4) THEN

    DDCS= CLF*ROW*VEG_COV_STEP*SSR_FIE_CAPC(I)*ETA_S(I)*0.5*PI*COS((PI* &
     (P_TEM_SOI(I)-0.5*TFRZ1-0.5*TFRZ2))/(TFRZ1-TFRZ2))/(TFRZ1-TFRZ2)

! Viterbo et al. (1999) modified
   ELSE IF (INP_SOIL == 5) THEN

    DDCS= CLF*ROW*(P_ETA_W(I)+P_ETA_I(I))*0.5*PI*COS((PI*(P_TEM_SOI(I)- &
      .5*TFRZ1-0.5*TFRZ2))/(TFRZ1-TFRZ2))/(TFRZ1-TFRZ2)
```

```
   ENDIF

! P_TEM_SOI(1) out of thresholds for freezing
  ELSE

   DDCS= 0.

  ENDIF

  IF (DDCS >= 0.) THEN
   DCS = DCS + DDCS
  ENDIF

 ENDIF ! INP_SOIL

! ROCIVOL = surface heat capacity of vegetated soil

!...Vegetated soil: adds root heat capacity (roots = water: see VEG_HEA_CAP)
 IF (LVEG/=12 .AND. LVEG/=14 .AND. LVEG/=15)&
  ROCIVOL(I)=(1.-RP_TOTAL(I))*DCS*DEP_SOI(I) + RP_TOTAL(I)*2302.3*LAI_STEP

!...Ice (it is dry!!!)
 IF (LVEG==12) ROCIVOL(I)=ROC(I)*DEP_SOI(I)

!...Water (ocean) 4.186E6 is water heat capacity
 IF (LVEG==14 .OR. LVEG==15) ROCIVOL(I)=4.186E6*DEP_SOI(I)

! soil hydrulic conductivity
! Correction to avoid numerical instability when SR_SOI is close to 0.01. The min
! SR_SOI used to calculate CAPPAETA in dry soils is 0.03. Bonanno (2009) p 88-97
 CAPPAETA(I)=CAPPAETA_S(I)*MAX(SR_SOI(I),0.03)**(2.*BSOIL(I)+3)

! water vapor diffusivity, follows Niu et al. (1997)
 DIFFWV= 2.3E-5*(P_TEM_SOI(I)/273.16)**1.75

! soil surface saturated specific humidity
 CALL SPECUM(100.,P_TEM_SOI(I),PRES_A_STEP,PIPPO,PIPPO,ES)

! water vapor density from state eq. for moist air - factor 100 because ES is in Pa
 WAT_VAP_DENS=(100.*ES)/WAT_VAP_CON/P_TEM_SOI(I)*EXP(GRAVITY*MOIS_POT(I)/(WAT_VAP_CON*&
  P_TEM_SOI(I)))

 DIFFUSIV=-BSOIL(I)/ETA_S(I)/SR_SOI(I)*MOIS_POT(I)
 DIFFSOIL=CAPPAETA(I)*DIFFUSIV

!Correction to avoid numerical instability on soil moisture after thawing
!The water vapor flux is negligible if (P_ETA_W+P_ETA_I) > 0.06. -> LDIF=.false.
!(Niu et al. 1997)

!Control on soil moisture: water vapor transfer is active only for very dry soil
 LVDIF=.TRUE.
 IF ((P_ETA_W(I)+P_ETA_I(I))>0.06 ) THEN
  LVDIF=.FALSE.
 END IF

 IF (LVAP.OR.LVDIF) THEN ! taken from Niu et al. (1997)
  DIFFVE=-DIFFWV*TORTUOS*(1.-SR_SOI(I))*WAT_VAP_DENS*GRAVITY*MOIS_POT_SAT(I)*&
   BSOIL(I)/DENS_WAT/WAT_VAP_CON/P_TEM_SOI(I)/SR_SOI(I)**(BSOIL(I)+1.)
  DIFFVT(I)=-DIFFWV*TORTUOS*ETA_S(I)*(1.-SR_SOI(I))*WAT_VAP_DENS/DENS_WAT*&
   ((1./P_TEM_SOI(I))+(GRAVITY*MOIS_POT(I)/WAT_VAP_CON/P_TEM_SOI(I)**2)-&
   (17.269*273.3/(P_TEM_SOI(I)-35.86)**2))
 ELSE
  DIFFVE=0.
  DIFFVT(I)=0.
 END IF
```

```
!Control on soil temperature: for some freezing schemes, in case of cold soil,
!water transfer is prohibited, so all diffusivities and conductivities in that
!layer are set=0
 LVWT=.TRUE.
 IF ( ( (INP_SOIL==3.OR.INP_SOIL==4.OR.INP_SOIL==5).AND.P_TEM_SOI(I)<TFRZ2) .OR.&
  ( (INP_SOIL==0.OR.INP_SOIL==1.OR.INP_SOIL==2).AND.P_TEM_SOI(I)<DTK) ) THEN
  LVWT=.FALSE.
 END IF

 IF (.NOT.LVWT) THEN
  DIFFSOIL=0.
  DIFFVE=0.
  DIFFVT(I)=0.
  CAPPAETA(I)=0.
 ENDIF

 DIFFSUM(I)=DIFFSOIL+DIFFVE
END DO

! thermal conductivity between snow and 1st soil layer, and snow and canopy
IF (LSNOW) THEN

! soil surface conductivity following Pielke (1984) pag.385
 PF1=LOG10(-100.*MOIS_POT_SAT(1)/SR_SOI(1)**BSOIL(1))
 CAPPAUNO=MAX(0.1722,420.*EXP(-PF1-2.7))    ! [J/(m*s*K)]

! snow thermal conductivity follows Yen (1981)
! Yen Y (1981) Review of thermal properties of snow,ice and seaice, CRREL Rep 81-10
! can be found in J. W. Ren, D. Qin and M. H. Huang (1991). Thermal properties and
temperature distribution of
! snow/firn on the Law Dome ice cap, Antarctica. Antarctic Research, 2(2), 38-46.
 IF (DENS_SN > 0.) THEN ! in case no snow
  CAPPASN=2.22362*(DENS_SN/1000.)**1.885 ! in original formula, DENS_SN was in Mg/m3
 ELSE
  CAPPASN=2.22862*(DENS_SN_MIN/1000.)**1.885
 ENDIF

! canopy thermal conductivity according to G94 (1994), for which canopy equals LAI
! mm of water, i.e. = 0.6 (water cond.) * 10^-3 (mm to m) * LAI_STEP

! mixed conductivity calculated as weighted average
 CAPPAMIXF=WEIGHTMEAN(CAPPASN,6.0E-4*LAI_STEP,HEI_SN,VEG_HEI_STEP)
 CAPPAMIXG=WEIGHTMEAN(CAPPASN,CAPPAUNO,HEI_SN,DEP_SOI(1))

ELSE ! no snow
 CAPPAMIXF=0.
 CAPPAMIXG=0.
END IF

! calculation of averaged-layer variables - adiacent layer average

! all variables 12 for soil thermal conductivity
DO I=1,NUM_SOI_LAY-1
 Q12=X_WEI_MEA(SR_SOI(I),SR_SOI(I+1),1,ERR_VE)
 FS12=X_WEI_MEA(MOIS_POT_SAT(I),MOIS_POT_SAT(I+1),1,ERR_VE)
 BSOIL12 = X_WEI_MEA(BSOIL(I),BSOIL(I+1),1,ERR_VE)
 PF12=LOG10(-100.*FS12/Q12**BSOIL12)

! if no ICE (12), inland (14) or open WATER (15)
 IF (LVEG/=12 .AND. LVEG/=14 .AND. LVEG/=15)THEN

  IF(NSO(I)==15)THEN ! asphalt or concrete
   THEDIFM(I)=0.82 *(1-ETA_S(I))+0.6*P_ETA_W(I)+2.5*P_ETA_I(I)+0.026*&
    (ETA_S(I)-P_ETA_W(I)-P_ETA_I(I))
  ELSE IF(NSO(I)==16)THEN ! pervious concrete
   THEDIFM(I)=2.00*(1-ETA_S(I))+0.6*P_ETA_W(I)+2.5*P_ETA_I(I)+0.026*&
    (ETA_S(I)-P_ETA_W(I)-P_ETA_I(I))  !1.228
```

```
  ELSE IF(NSO(I)==17)THEN ! stone aggregate
   THEDIFM(I)=2.00*(1-ETA_S(I))+0.6*P_ETA_W(I)+2.5*P_ETA_I(I)+0.026*&
    (ETA_S(I)-P_ETA_W(I)-P_ETA_I(I)) !! 2.10 Masson
  ELSE ! all other soil types
   THEDIFM(I)=MAX(0.1722,420.*EXP(-PF12-2.7))
  END IF

! Ice
 ELSE IF (LVEG==12) THEN
  THEDIFM(I)=2.5 !W/mK, G94 App

! Water (ocean)
 ELSE IF (LVEG==14 .OR. LVEG==15) THEN
  THEDIFM(I)=0.6 !W/mK,G94 p291

 END IF

! weighted averages of hydraulic conductivites and diffusivities at interface layers
 CAPPAETAM(I)=X_WEI_MEA(CAPPAETA(I),CAPPAETA(I+1),1,ERR_VE)
 DIFFSUMM(I)=X_WEI_MEA(DIFFSUM(I),DIFFSUM(I+1),1,ERR_VE)
 DIFFVTM(I)=MAX(0.,X_WEI_MEA(DIFFVT(I),DIFFVT(I+1),1,ERR_VE) ) ! <0 diff. neglected
!???perchè ce ne sono? non dovrebbe succedere mai
END DO

RETURN
END SUBROUTINE AV_NS_MOI

!*******************************************************************************
SUBROUTINE CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI,CO2_CON_STEP,E_CAN
,ES_CAN,F_CI,GAMMA_STAR,&
 J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)
!*******************************************************************************
! Core routine computing carbon fixation due to photosynthesis
!*******************************************************************************
! Calls: none
! - Author: Ines Cerenzia (06-Feb-2012)
! - Last revision : C. Cassardo (15-May-2014)
!*******************************************************************************
IMPLICIT NONE

! parameter

! input
REAL, INTENT(IN) :: APR          ! [Pa]
REAL, INTENT(IN) :: CO2_CON_STEP        ! [Pa]
REAL, INTENT(IN) :: E_CAN        ! [Pa]
REAL, INTENT(IN) :: ES_CAN       ! [Pa]
REAL, INTENT(IN) :: GAMMA_STAR   ! [Pa]
REAL, INTENT(IN) :: KC           ! [Pa]
REAL, INTENT(IN) :: KO           ! [KPa]
REAL, INTENT(IN) :: J            ! [umol/m2s]
REAL, INTENT(IN) :: MRS
REAL, INTENT(IN) :: O2CONT       ! [KPa]
INTEGER, INTENT(IN) :: PTW
REAL, INTENT(IN) :: SSR_FIE_CAP
REAL, INTENT(IN) :: SO_SRAT_RO_DE
REAL, INTENT(IN) :: SSR_WI
REAL, INTENT(IN) :: RB           ! [m2s/umol]
REAL, INTENT(IN) :: RES_RAT      ! mitocondrial respiration [umol/m2s]
REAL, INTENT(IN) :: TEMP_F_STEP ! [K]
REAL, INTENT(IN) :: PVMAX        ! [umol/m2s]

! input/output
REAL, INTENT(INOUT) :: CI        ! [Pa]

! output
REAL, INTENT(OUT) :: GRO_CAR_ASS_RAT    ! gross rate of carbon assimilation by photosynthesis
```

```
[umol/m2s]
REAL, INTENT(OUT) :: AV                 ! [umol/m2s]
REAL, INTENT(OUT) :: AJ                 ! [umol/m2s]
REAL, INTENT(OUT) :: AC                 ! [umol/m2s]
REAL, INTENT(OUT) :: NET_CAR_ASS_RAT    ! net rate of carbon assimilation by photosynthesis
[umol/m2s]
REAL, INTENT(OUT) :: F_CI               ! [Pa]
REAL, INTENT (OUT) :: RS                ! [umol/m2s]

! dummy
REAL :: ATMP
REAL :: BTMP
REAL :: CI_NEW                  ! [Pa]
REAL :: CS                      ! [Pa]
REAL :: CTMP
REAL :: EP
REAL :: FF2
REAL :: RS1
REAL :: RS2

AV=PVMAX*MAX((CI-GAMMA_STAR),0.)/(CI+KC*(1.+(O2CONT/KO)))*REAL(PTW)+PVMAX*REAL(1.-PTW)
!umol/m2s
AC=PVMAX/2.*REAL(PTW)+4.e3*PVMAX*CI/APR*REAL(1.-PTW)                                    !umol/m2s
!NBNBNB: coeff di PVMAX qui e' 4.e3 ma nella prova viti era 1.8e4????????
AJ=J*MAX((CI-GAMMA_STAR),0.)/(4.*(CI+2.*GAMMA_STAR))*REAL(PTW)+J*REAL(1.-PTW)           !umol/m2s

! gross rate of carbon assimilation by photosynthesis
GRO_CAR_ASS_RAT = MIN(AV,AJ,AC)                                                         !umol/m2s
! net rate of carbon assimilation (excludes the respiration RB rate)
NET_CAR_ASS_RAT = GRO_CAR_ASS_RAT - RES_RAT                                             !umol/m2s

CS=CO2_CON_STEP-GRO_CAR_ASS_RAT*APR*1.37*RB
EP=MAX(MIN(E_CAN,ES_CAN),(0.25*REAL(PTW)+0.4*REAL(1-PTW))*ES_CAN)                       !Pa

ATMP=MRS*GRO_CAR_ASS_RAT*APR*EP/(CS*ES_CAN)+2000.
BTMP=MRS*GRO_CAR_ASS_RAT*APR*RB/CS+2000.*RB-1.
!NBNBNB: MRS=9 per C4 e 5 per C3 ma nella prova vale 3.13 per C3?????
CTMP=-RB

RS1=(-BTMP+SQRT(BTMP**2-4.*ATMP*CTMP))/(2.*ATMP)
RS2=(-BTMP-SQRT(BTMP**2-4.*ATMP*CTMP))/(2.*ATMP)
RS=MAX(RS1,RS2)

!Water stress
!FF2=MIN(1.,(10*(SO_SRAT_RO_DE-SSR_WI))/(3.*(SSR_FIE_CAP-SSR_WI))) ! Da Gollan et al. 1986
FF2=MIN(1.,MAX(0.01,(10*(SO_SRAT_RO_DE-SSR_WI))/(3.*(SSR_FIE_CAP-SSR_WI)))) ! Da Gollan et
al. 1986
! above formula modified forbidding negative values (posed MIN value = 0.01)
RS=RS/FF2

CI_NEW=CS-1.65*GRO_CAR_ASS_RAT*APR*RS
F_CI=CI_NEW-CI

END SUBROUTINE CALC_CAR_ASS_RAT

!*******************************************************************************
REAL FUNCTION LAT_HEA_EVA (T)
!*******************************************************************************
! Calculates latent evaporation/sublimation heat flux
!*******************************************************************************
! Calls: none
! - Author: C. Cassardo (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*******************************************************************************
USE CONSTANTS
IMPLICIT NONE
```

```
REAL, INTENT(IN) :: T                    !temperature in Kelvin degree

IF (T >= DTK)THEN
 LAT_HEA_EVA = (2500.-2.39*(T - DTK))*1000. ! J/kg  pag 201 Longhetto
ELSE
 LAT_HEA_EVA = (2824.-0.2*(T - DTK))*1000.  ! J/kg  pag 201 Longhetto
END IF

RETURN
END FUNCTION LAT_HEA_EVA

!*******************************************************************************
REAL FUNCTION DEWPOINT(Q_SP_A_STEP,PRES_A_STEP)
!*******************************************************************************
! Calculates dew point temperature [K]
!*******************************************************************************
! Calls: none
! - Author: M. Trevisan (15-Mar-2004)
! - Last revision: C. Cassardo (15-May-2014)
!*******************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL :: AA
REAL,INTENT(IN) :: Q_SP_A_STEP              ! Specific humidity [kg/kg]
REAL,INTENT(IN) :: PRES_A_STEP              ! Pressure [hPa]
REAL :: E                                ! vapor pressure [hPa]

E=PRES_A_STEP*Q_SP_A_STEP/(A5+A6*Q_SP_A_STEP)          ! non sat vapor pressure (hPa)
AA=1./A2*ALOG(E/A1)
DEWPOINT=MAX(1.,((AA*A4-A3)/(AA-1)))    ! Clausius-Clapeyron invert. (natural log.)

RETURN
END FUNCTION DEWPOINT

!*******************************************************************************
SUBROUTINE DIAG3(A,B,C,D,CO,M)
!*******************************************************************************
! Diagonalize a vector
!*******************************************************************************
! Calls: none
! - Author: M. W. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!*******************************************************************************
IMPLICIT NONE
INTEGER,INTENT(IN) :: M
REAL,DIMENSION(M),INTENT(INOUT) :: A
REAL,DIMENSION(M),INTENT(INOUT)  :: B
REAL,DIMENSION(M),INTENT(INOUT)  :: C
REAL,DIMENSION(M),INTENT(INOUT)  :: D
REAL,DIMENSION(M),INTENT(OUT)  :: CO
INTEGER :: I

DO I=1,M-1
 C(I)=C(I)/B(I)
 D(I)=D(I)/B(I)
 B(I+1)=B(I+1)-A(I+1)*C(I)
 D(I+1)=D(I+1)-A(I+1)*D(I)
END DO

CO(M)=D(M)/B(M)

DO I=M-1,1,-1
 CO(I)=D(I)-C(I)*CO(I+1)
END DO

RETURN
```

```
END SUBROUTINE DIAG3

!******************************************************************************
SUBROUTINE DRAG(LSNOW,LVEG,PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,VEL_MOD_STEP,DRAG_H,DRAG_M,
DRAG_V,&
 ZER_DIS_LEV,VEG_HEI_STEP,P_TEMP_F_STEP,P_TEM_SOI,REYNOLDS,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,
 &
 COVER_SN,UAF,USTAR,Z0_F_H,Z0_F_M,Z0_G_H,Z0_G_M,Z0_G_V,Z0_H,Z0_M,Z0_SN,Z0_V,H_OBS_ALL,
 ACT_VEG_HEI,&
 H_OBS_WIND,ERR_ECO,Z0_M_ECO,Z0_H_ECO)
!******************************************************************************
! Evaluation of drag coefficients using the formulation proposed by Andre
! (from Louis, 1979). This routine calculates also Richardson number RIC_NUM and
! the wind velocity UAF in the vegetation.
!******************************************************************************
! Calls: LOUIS, POTENTIAL
! - Authors: C. Cassardo, J.J. Ji, P. Ramieri (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL, INTENT(IN) :: PRES_A_STEP !pressure at H_OBS_ALL
REAL, INTENT(IN) :: Q_SP_A_STEP !specific humidity at H_OBS_ALL
REAL, INTENT(IN) :: TEMP_A_STEP !air temperature at H_OBS_ALL
REAL, INTENT(IN) :: VEL_MOD_STEP    !horizontal wind speed at H_OBS_WIND
REAL, INTENT(OUT) :: DRAG_H !heat transfer coefficient
REAL :: DRAG_H_N                !neutral heat transfer coefficient
REAL, INTENT(OUT) :: DRAG_M !momentum drag coefficient
REAL :: DRAG_M_N                !neutral momentum drag coefficient
REAL, INTENT(OUT) :: DRAG_V     !water vapor bulk transfer coefficient
REAL :: DRAG_V_N                !neutral water vapor bulk transfer coefficient
REAL, INTENT(OUT) :: ZER_DIS_LEV    !zero displacement level
REAL, INTENT(IN) :: ERR_ECO
REAL, INTENT(IN) :: VEG_HEI_STEP    !height of canopy
REAL :: LOUIS
LOGICAL, INTENT(IN) :: LSNOW!logical snow presence
INTEGER, INTENT(IN) :: LVEG !vegetation type code
REAL :: NU
REAL, INTENT(IN) :: P_TEMP_F_STEP       !current canopy temperature
REAL, INTENT(IN) :: P_TEM_SOI   !soil temperature (previous step)
REAL, INTENT(OUT) :: REYNOLDS   !Reynolds number
REAL :: RIC_NUM             !Richardson number
REAL, INTENT(IN) :: VEG_COV_STEP    !vegetation cover
REAL, INTENT(IN) :: COVER_SN_F  !snow cover over vegetation
REAL, INTENT(IN) :: COVER_SN_G  !snow cover over bare soil
REAL, INTENT(IN) :: COVER_SN    !snow cover raction
REAL :: THETA_V
REAL :: THETAA
REAL :: THETA_S
REAL :: TS                  !surface temperature
REAL, INTENT(OUT) :: UAF    !wind velocity within the canopy
REAL, INTENT(IN) :: USTAR   !surface friction velocity
REAL, INTENT(OUT) :: Z0_F_H !canopy heat aerodynamic roughness lenght
REAL, INTENT(OUT) :: Z0_F_M !canopy momentum aerodynamic roughness lenght
REAL, INTENT(OUT) :: Z0_G_H !bare soil heat aerodynamic roughness lenght
REAL, INTENT(OUT) :: Z0_G_M !bare soil momentum aerodynamic roughness lenght
REAL, INTENT(OUT) :: Z0_G_V !bare soil w. vapor aerodynamic roughness lenght
REAL, INTENT(INOUT) :: Z0_H !heat mean aerodynamic roughness lenght
REAL, INTENT(IN) :: Z0_H_ECO
REAL, INTENT(INOUT) :: Z0_M !momentum mean aerodynamic roughness lenght
REAL, INTENT(IN) :: Z0_M_ECO
REAL, INTENT (OUT) :: Z0_SN !snow roughness length
REAL, INTENT(INOUT) :: Z0_V !w. vapor mean aerodynamic roughness lenght
REAL, INTENT(INOUT) :: H_OBS_ALL    !screen level height for all obs but wind
REAL, INTENT(OUT) :: ACT_VEG_HEI    !active surface heigth (mean canopy heigth)
REAL, INTENT(INOUT) :: H_OBS_WIND   !screen level height for wind
```

```
! evaluation of zero displacement level (p. 86 G94)
ZER_DIS_LEV = VEG_COV_STEP * 0.67 * VEG_HEI_STEP

NU = 1.3432E-5 + 9.3571E-8 * (TEMP_A_STEP - DTK)     ! air kinematic viscosity

! automatic correction of screen level height if Z < D
IF (H_OBS_ALL <= VEG_HEI_STEP)  THEN
 PRINT *,"H_OBS_ALL, VEG_HEI_STEP = ",H_OBS_ALL,VEG_HEI_STEP
 PRINT *,"Found H_OBS_ALL <= VEG_HEI_STEP - using automatic correction"
! H_OBS_ALL is imposed 2m higher than VEG_HEI_STEP
 H_OBS_ALL=VEG_HEI_STEP+2.
! H_OBS_WIND is imposed equal to H_OBS_ALL
 H_OBS_WIND=H_OBS_ALL
 PRINT *,"Imposed H_OBS_ALL,H_OBS_WIND = ",H_OBS_ALL,H_OBS_WIND
END IF

IF (LVEG /= 14. .AND. LVEG /= 15. .AND. LVEG /= 12.) THEN
! all land use types except water and ice

 Z0_G_M = 0.005         !table A6 pag.290 G94

 IF(LVEG==27 .OR. LVEG==28) Z0_G_M=0.01 !concrete, asphalt

 Z0_G_H=Z0_G_M/7.4          ! G94 p. 244
 Z0_G_V=Z0_G_M/7.4          ! G94 p. 244
 Z0_F_M = 0.13 * VEG_HEI_STEP   !eq.4.4 G94:Z0_F_M now is only for canopy and not all soil
 Z0_F_H = Z0_F_M/7.4        ! adapted from G94

! in case of snow presence, Z0_M=0.005 x snow pag.190 Dingman
 IF (LSNOW)  THEN
  Z0_SN=(0.135*NU/USTAR)+0.035*USTAR*USTAR/GRAVITY*(1+5.*&
   EXP(-((USTAR-0.18)/0.10)**2))
 END IF

 IF (Z0_M_ECO == ERR_ECO) THEN

! weighted average between canopy, bare soil and snow
  Z0_M = (1.- COVER_SN_F)*VEG_COV_STEP*Z0_F_M + (1.-VEG_COV_STEP)*(1.-COVER_SN_G)*Z0_G_M +&
   COVER_SN*Z0_SN
 ELSE

! use Ecoclimap that does not consider snow, so it add snow contribution to Z0_M_ECO
  Z0_M = Z0_M_ECO * (1 - COVER_SN) + COVER_SN * Z0_SN
 ENDIF

 IF (Z0_H_ECO == ERR_ECO) THEN

  Z0_H = Z0_M / 7.4              !pag.244 G94
 ELSE

! use Ecoclimap that does not consider snow, so it add snow contribution to Z0_H_ECO
  Z0_H = Z0_M / 7.4 * (1 - COVER_SN) + COVER_SN * Z0_SN / 7.4
 ENDIF

 Z0_V=Z0_H

ELSE IF (LVEG == 12)   THEN ! over ice

 Z0_G_M = 0.005 !same as for snow (ref?)
 Z0_G_H = Z0_G_M / 7.4
 Z0_G_V = Z0_G_H
 Z0_F_H = Z0_G_H ! temporary because it is used elsewhere

! assumed Z0_M = Z0_G_M supposing that there is not vegetation over ice
 Z0_M = Z0_G_M
 Z0_H = Z0_G_H
```

```
 IF (Z0_M_ECO /= ERR_ECO) Z0_M = Z0_M_ECO
 IF (Z0_H_ECO /= ERR_ECO) Z0_H = Z0_H_ECO
 Z0_V = Z0_H

ELSE ! over ocean or insland water

 IF (USTAR <= 0.23) THEN ! smooth flow pag.98,102 G94

  Z0_G_M = 0.11 * NU / USTAR              !eq. 4.3 G94
  Z0_G_H = 0.20 * NU / USTAR              !eq. 4.26a G94
  Z0_G_V = 0.30 * NU / USTAR              !eq. 4.26b G94
  Z0_M = Z0_G_M
  Z0_H = Z0_G_H
  IF (Z0_M_ECO /= ERR_ECO) Z0_M = Z0_M_ECO
  IF (Z0_H_ECO /= ERR_ECO) Z0_H = Z0_H_ECO
  Z0_V = Z0_H

 ELSE ! rough flow pag.98,102 G94

  Z0_G_M = CHARNOOK * USTAR * USTAR / GRAVITY          ! eq.4.5 G94
  REYNOLDS = USTAR * Z0_G_M / NU                       ! pag.92 G94
  Z0_G_H = Z0_G_M * EXP(2.- 2.48 * (REYNOLDS**0.25))     ! eq. 4.27 G94
  Z0_G_V = Z0_G_M * EXP(2.- 2.28 * (REYNOLDS**0.25))     ! eq. 4.28 G94
  Z0_M = Z0_G_M
  Z0_H = Z0_G_H
  IF (Z0_M_ECO /= ERR_ECO) Z0_M = Z0_M_ECO
  IF (Z0_H_ECO /= ERR_ECO) Z0_H = Z0_H_ECO
  Z0_V = Z0_H

 END IF

 Z0_F_H = Z0_G_H ! temporary because it is used elsewhere

END IF

! neutral drag coefficients evaluation
DRAG_M_N = VON_KARMAN**2/(ALOG((H_OBS_ALL-ZER_DIS_LEV)/Z0_M))**2                    ! eq.3.43
G94
DRAG_H_N = VON_KARMAN**2/ALOG((H_OBS_ALL-ZER_DIS_LEV)/Z0_M)/ALOG((H_OBS_ALL-ZER_DIS_LEV)/Z0_H)
 ! eq.3.48 G94
DRAG_V_N = VON_KARMAN**2/ALOG((H_OBS_ALL-ZER_DIS_LEV)/Z0_M)/ALOG((H_OBS_ALL-ZER_DIS_LEV)/Z0_V)
 ! "  " "

! evaluation of active surface heigth and its temperature (for calculating Richardson number)
ACT_VEG_HEI = VEG_COV_STEP * VEG_HEI_STEP
TS = VEG_COV_STEP * P_TEMP_F_STEP + (1.- VEG_COV_STEP)* P_TEM_SOI

! determination of pot. temperatures corresp. to TEMP_A_STEP and TS for Richardson number
CALL POTENTIAL(PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,THETAA,THETA_V)
CALL POTENTIAL(PRES_A_STEP,Q_SP_A_STEP,TS,THETA_S,THETA_V)

! Richardson number, G94 (1994) eq. 3.45
RIC_NUM = GRAVITY*(H_OBS_ALL-ACT_VEG_HEI)*(THETAA-THETA_S)/(THETA_S * VEL_MOD_STEP**2)

! corrections for stability effects in drag coefficients (according to G94)
DRAG_M = LOUIS(DRAG_M_N,(H_OBS_ALL-ZER_DIS_LEV)/Z0_M,RIC_NUM,1)
DRAG_H = LOUIS(DRAG_H_N,(H_OBS_ALL-ZER_DIS_LEV)/Z0_H,RIC_NUM,2)
DRAG_V = LOUIS(DRAG_V_N,(H_OBS_ALL-ZER_DIS_LEV)/Z0_V,RIC_NUM,3)

! wind speed within canopy
UAF=VEL_MOD_STEP*(1.0-VEG_COV_STEP*(1.0-SQRT(DRAG_M)))
UAF = MAX(UAF,0.05)      !see eq 4.6 pag 32 CLM
! UAF_TOP = (USTAR/VON_KARMAN)*ALOG((VEG_HEI_STEP-ZER_DIS_LEV)/Z0_M) !parametr. modified from
p.63 bonan

RETURN
```

```
END SUBROUTINE DRAG

!*****************************************************************************
REAL FUNCTION D_SPEC_UM(AT,APR)
!*****************************************************************************
! Derived from SPEC_UM, calculates derivative of SPEC_UM with respect to temp.
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Mar-1994)
! - Last revision: C. Cssardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE
REAL, INTENT(IN) :: APR
REAL, INTENT(IN) :: AT
REAL :: COST
REAL :: ES
REAL :: QS

ES=A1*EXP(A2*(AT-A3)/(AT-A4))   ! sat water vapor pressure (hPa)
QS=A5*ES/(APR-A6*ES)            ! sat specific humidity (kg/kg)
COST=A2*(A3-A4)
D_SPEC_UM=COST*APR*QS/(APR-A6*ES)/(AT-A4)/(AT-A4)

RETURN
END FUNCTION D_SPEC_UM

!*****************************************************************************
SUBROUTINE DYNAMIC_MEMORYOUT(IANNEW, IGINEW, IMENEW, IMINEW, IORANEW, JCOUNT, NCOD,
NUM_SOI_LAY,&
 CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, &
 DIFFVT, DIFFVTM, ETA_I, ETA_W, EVATRA, FRP_OCC, MOIS_POT, ROC, ROCIVOL, SOILFLUX, SR_SOI, &
 TEM_SOI, THEDIFM,&
 ALB_SD, ALB_SN, ALB_TOT, BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT,&
 CAPPAMIXF, CAPPAMIXG, CL_LOW_STEP, CL_TOT_STEP, CO2_CON_STEP, COND_F_DRY, COND_F_WET,
 COND_G_DRY,&
 COND_G_WET, CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F
 , &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, DENS_SN, DRAG_H, DRAG_M,
  &
 DRAG_V, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, EVA_SN_F, EVA_SN_G,
 &
 GR_THEO, GRO_CAR_ASS_RAT, HAZE_IND, HEI_SN, HM_SN, HW_SN, LAI_STEP, LEA_WAT_CON, LHF_A, LHF_F
 , &
 LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, NET_CAR_ASS_RAT, NR_A,
  &
 NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, PREC_F, PREC_G, PRES_A_STEP,&
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, RAD_L_D, RAD_L_F_D, RAD_L_F_U, &
 RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D, RAD_L_G_TOT_U, RAD_L_G_U, RAD_L_SN_F_D, &
 RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, RAD_S_F_D, RAD_S_D, RAD_S_F_U,
  &
 RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U, RAD_S_SN_G_D, RAD_S_SN_G_U, &
 RAD_S_SN_U, RAD_S_U, REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, RESIST_AH, &
 RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, RESIST_LEAF,
  &
 RESIST_SRF, RESIST_SRF_SN, RH_STEP, SHF_A, SHF_F, SHF_G, SHF_SN, SHF_SN_F, SHF_SN_G, SOL_ALFA
 , &
 SRF_ICE_CON, SRF_WAT_CON, SUM_DRAI, SUM_DRAI_TMP, SUM_EVAP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, &
 SUM_PREC, SUM_PREC_SN, SUM_PREC_TMP, SUM_RUNO, SUM_RUNO_TMP, SUM_SOI_HEA_STO, TEMP_SN,
 TEMP_A_F, &
 TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_F_STEP, TEMP_ROOT, UPWP, USTAR, VPWP, VEG_COV_STEP, &
 VEG_HEI_STEP, VEL_MOD_STEP,&
 FORMAT_OUTPUT, VARIABLE_NAME, &
 LAYERS_OUT_CAPPAETA, LAYERS_OUT_CAPPAETAM, &
 LAYERS_OUT_DIFFSUM, LAYERS_OUT_DIFFSUMM, LAYERS_OUT_DIFFVT, LAYERS_OUT_DIFFVTM, &
 LAYERS_OUT_ETA_I, LAYERS_OUT_ETA_W, LAYERS_OUT_EVATRA, LAYERS_OUT_FRP_OCC, &
 LAYERS_OUT_MOIS_POT, LAYERS_OUT_SR_SOI, LAYERS_OUT_ROC, LAYERS_OUT_ROCIVOL, &
```

```
 LAYERS_OUT_SOILFLUX, LAYERS_OUT_TEM_SOI, LAYERS_OUT_THEDIFM,&
 OUTPUT_VARIABLES)
!******************************************************************************
! Writes output variables on apropriate output files.
!******************************************************************************
! Calls: none
! - Authors: M. Galli, R. Bonanno (23-Apr-2009)
! - Last revision : C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: IANNEW, IGINEW, IMENEW, IMINEW, IORANEW, JCOUNT, NCOD, NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, &
 DIFFVT, DIFFVTM, ETA_I, ETA_W, EVATRA, FRP_OCC, MOIS_POT, ROC, ROCIVOL, SOILFLUX, SR_SOI, &
 TEM_SOI, THEDIFM

REAL,INTENT(IN) :: ALB_SD, ALB_SN, ALB_TOT, BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT,&
 CAPPAMIXF, CAPPAMIXG, CL_LOW_STEP, CL_TOT_STEP, CO2_CON_STEP, COND_F_DRY, COND_F_WET,
 COND_G_DRY,&
 COND_G_WET, CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F
 , &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, DENS_SN, DRAG_H, DRAG_M,
  &
 DRAG_V, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, EVA_SN_F, EVA_SN_G,
 &
 GR_THEO, GRO_CAR_ASS_RAT, HAZE_IND, HEI_SN, HM_SN, HW_SN, LAI_STEP, LEA_WAT_CON, LHF_A, LHF_F
 , &
 LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, NET_CAR_ASS_RAT, NR_A,
  &
 NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, PREC_F, PREC_G, PRES_A_STEP,&
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, RAD_L_D, RAD_L_F_D, RAD_L_F_U, &
 RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D, RAD_L_G_TOT_U, RAD_L_G_U, RAD_L_SN_F_D, &
 RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, RAD_S_F_D, RAD_S_D, RAD_S_F_U,
  &
 RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U, RAD_S_SN_G_D, RAD_S_SN_G_U, &
 RAD_S_SN_U, RAD_S_U, REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, RESIST_AH, &
 RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, RESIST_LEAF,
  &
 RESIST_SRF, RESIST_SRF_SN, RH_STEP, SHF_A, SHF_F, SHF_G, SHF_SN, SHF_SN_F, SHF_SN_G, SOL_ALFA
 , &
 SRF_ICE_CON, SRF_WAT_CON, SUM_DRAI, SUM_DRAI_TMP, SUM_EVAP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, &
 SUM_PREC, SUM_PREC_SN, SUM_PREC_TMP, SUM_RUNO, SUM_RUNO_TMP, SUM_SOI_HEA_STO, TEMP_SN,
 TEMP_A_F, &
 TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_F_STEP, TEMP_ROOT, UPWP, USTAR, VPWP, VEG_COV_STEP, &
 VEG_HEI_STEP, VEL_MOD_STEP

CHARACTER(len=*),INTENT(IN) :: FORMAT_OUTPUT
CHARACTER(len=*),DIMENSION(NSUDYN),INTENT(IN) :: VARIABLE_NAME
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: LAYERS_OUT_CAPPAETA, LAYERS_OUT_CAPPAETAM, &
 LAYERS_OUT_DIFFSUM, LAYERS_OUT_DIFFSUMM, LAYERS_OUT_DIFFVT, LAYERS_OUT_DIFFVTM, &
 LAYERS_OUT_ETA_I, LAYERS_OUT_ETA_W, LAYERS_OUT_EVATRA, LAYERS_OUT_FRP_OCC, &
 LAYERS_OUT_MOIS_POT, LAYERS_OUT_SR_SOI, LAYERS_OUT_ROC, LAYERS_OUT_ROCIVOL, &
 LAYERS_OUT_SOILFLUX, LAYERS_OUT_TEM_SOI, LAYERS_OUT_THEDIFM
LOGICAL,DIMENSION(NSUDYN),INTENT(IN) :: OUTPUT_VARIABLES

REAL,DIMENSION(NSU1) :: SU1
REAL,DIMENSION(NSU2) :: SU2
REAL,DIMENSION(NSU3) :: SU3
REAL,DIMENSION(NSU4) :: SU4
REAL,DIMENSION(NSU5) :: SU5
REAL,DIMENSION(NSU6) :: SU6
REAL,DIMENSION(NSUDYN) :: SU_POSSIBILITIES

INTEGER :: J
INTEGER :: I_POSSIB
INTEGER :: J_SU
```

```
INTEGER :: K_NUM_SOI_LAY
INTEGER,SAVE :: IJK=0
REAL,EXTERNAL :: LAT_HEA_EVA

IJK=IJK+1
!*********************************************
!* output data storage after calculation cycle *
!*********************************************
!
!...1: input data (1-10)
!
SU1( 1)=SUM_PREC_TMP*1000            ! precipitation [mm]
SU1( 2)=VEL_MOD_STEP                    ! screen wind speed [m/s]
SU1( 3)=TEMP_A_STEP-DTK                ! screen air temperature [C]
SU1( 4)=RH_STEP                     ! relative air humidity [%]
SU1( 5)=CL_TOT_STEP                    ! total cloudiness
SU1( 6)=CL_LOW_STEP                       ! low cloudiness
SU1( 7)=Q_SP_A_STEP                   ! Specific humidity
SU1( 8)=PRES_A_STEP                   ! Atmospheric pressure   (hPa)
SU1( 9)=RAD_S_D                    ! global radiation [Wm-2]
!
!...2: radiation data (7-33)
!
SU2( 1)=RAD_S_D                     ! global radiation [Wm-2]
SU2( 2)=NR_A                     ! net radiation [Wm-2]
SU2( 3)=SHF_A                     ! sensible heat flux [Wm-2]
SU2( 4)=LHF_A                     ! latent heat flux [Wm-2]
SU2( 5)=BAL_FLU_F+BAL_FLU_G+CONF_SN_SRF         ! atmosphere-soil-canopy flux [Wm-2]
SU2( 6)=BAL_FLU_G                     ! effective downward soil flux [Wm-2]
SU2( 7)=LHF_F_D                     ! transpiration flux [Wm-2]
SU2( 8)=NR_F                     ! [Wm-2]
SU2( 9)=NR_G                     ! [Wm-2]
SU2(10)=NR_SN                     ! [Wm-2]
SU2(10)=RAD_S_D                        ! [Wm-2]
SU2(11)=RAD_S_U                        ! [Wm-2]
SU2(12)=RAD_L_FG_D                       ! [Wm-2]
SU2(13)=RAD_L_FG_U                       ! [Wm-2]
SU2(14)=RAD_L_D                    ! [Wm-2]
SU2(15)=RAD_L_U                        ! [Wm-2]
SU2(16)=RAD_S_F_D                       ! [Wm-2]
SU2(17)=RAD_S_F_U                       ! [Wm-2]
SU2(18)=RAD_S_G_D                       ! [Wm-2]
SU2(19)=RAD_S_G_U                       ! [Wm-2]
SU2(20)=NR_L_F_D                      ! [Wm-2]
SU2(21)=NR_L_F_U                      ! [Wm-2]
SU2(22)=RAD_L_G_TOT_D                          ! [Wm-2]
SU2(23)=RAD_L_G_TOT_U                          ! [Wm-2]
SU2(24)=SOILFLUX(1)                     ! [Wm-2]
SU2(25)=SOILFLUX(2)                     ! [Wm-2]
SU2(26)=VEG_COV_STEP                     ! []
SU2(27)=LAI_STEP                   ! []
!
!...3: hydrological data (34-53)
!
SU3( 1)=SR_SOI(1)                  ! 1th soil layer humidity [%]
SU3( 2)=SR_SOI(2)                  ! 2nd soil layer humidity [%]
SU3( 3)=SR_SOI(3)                  ! 3rd soil layer humidity [%]
SU3( 4)=SR_SOI(4)                  ! 4th soil layer humidity [%]
SU3( 5)=SR_SOI(4)                  ! 5th soil layer humidity [%]
!SU3( 5)=SR_SOI(5)                  ! 5th soil layer humidity [%]
SU3( 6)=Q_A_F*1000.                    ! air-canopy humidity [g/kg]
SU3( 7)=SUM_EVAP                       ! cumulated evaporation [m]
SU3( 8)=SUM_PREC                       ! cumulated precipitation [m]
SU3( 9)=SUM_DRAI                    ! cumulated surface runoff [m]
SU3(10)=SUM_RUNO                    ! cumulated drainage [m]
SU3(11)=SUM_EVAP_TMP*1000.             ! inst. cumulated evaporation [mm]
SU3(12)=SUM_PREC_TMP*1000.             ! inst. cumulated precipitation [mm]
```

```
SU3(13)=SUM_DRAI_TMP*1000.          ! inst. cumulated surface runoff [mm]
SU3(14)=SUM_RUNO_TMP*1000.          ! inst. cumulated drainage [mm]
SU3(15)=LHF_A                     ! evapotranspiration [W m-2]
SU3(16)=LHF_F                     ! latent heat flux from canopy [W m-2]
!SU3(17)=EVA_F_W*LAT_HEA_EVA(P_TEMP_F_STEP)          ! canopy evaporation [W m-2]
SU3(18)=LHF_G                     ! soil evaporation/DENS_WAT [W m-2]
SU3(19)=COVER_SN                    ! snow fraction []
SU3(20)=HEI_SN                      ! snow height [m]
!
!...4: thermal data (54-60)
!
SU4( 1)=TEM_SOI(1)-DTK              ! 1th soil layer temp. [C]
SU4( 2)=TEM_SOI(2)-DTK              ! 2nd soil layer temp. [C]
SU4( 3)=TEM_SOI(3)-DTK              ! 3rd soil layer temp. [C]
SU4( 4)=TEM_SOI(4)-DTK              ! 4th soil layer temp. [C]
SU4( 5)=TEM_SOI(4)-DTK              ! 5th soil layer temp. [C]
SU4( 6)=TEM_SOI(4)-DTK              ! 4th soil layer temp. [C]
!SU4( 5)=TEM_SOI(5)-DTK              ! 5th soil layer temp. [C]
!SU4( 6)=TEM_SOI(6)-DTK              ! 4th soil layer temp. [C]
SU4( 7)=0.!TEM_SOI(7)-DTK             ! 5th soil layer temp. [C]
SU4( 8)=TEMP_F_STEP-DTK             ! canopy temperature [C]
SU4( 9)=TEMP_A_F-DTK              ! air-canopy temperature [C]
!
!...5: rumenta data (61-75)
!
SU5( 1)=REL_LEA_WAT_CON               ! percentage of canopy wetness [%]
SU5( 2)=HAZE_IND*100.         ! percentage of haze probability [%]
SU5( 3)=GR_THEO             ! Clear sky radiation flux [W/m2]
SU5( 4)=CL_TOT_STEP*100.          ! Total cloudiness [%]
SU5( 5)=SUM_MIN_LW_TMP          ! Minuti di bagnatura []
SU5( 6)=LAI_STEP              ! leaf area index
SU5( 7)=SUM_PREC_SN*1000.     ! snow precipitation [mm]
SU5( 8)=HEI_SN                  ! snow height [m]
SU5( 9)=COVER_SN               ! snow fraction []
SU5(10)=HM_SN                 ! snow water equivalent [m]
SU5(11)=HW_SN                 ! water into snow [m]
SU5(12)=NET_CAR_ASS_RAT                 ! net rate of carbon assimilation [micromol/m2s]
SU5(13)=RESIST_F               ! canopy resistance  [s/m]
SU5(14)=RESIST_LEAF               ! leaf resistance   [s/m]
SU5(15)=RESIST_F_PHOTO               ! stomatal resistance  [s/m]  -from photosynthesis calc.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!...SU_POSSIBILITIES filling -- DO NOT EDIT how this vector is filled
!                    vector containing all the possible SCALAR output vars
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
! shortwave radiation
SU_POSSIBILITIES(  1)=RAD_S_D             ! downward total solar radiation [Wm-2]
SU_POSSIBILITIES(  2)=RAD_S_U             ! upward total solar radiation [Wm-2]
SU_POSSIBILITIES(  3)=RAD_S_F_D           ! downward solar radiation over vegetation [Wm-2]
SU_POSSIBILITIES(  4)=RAD_S_F_U           ! upward solar radiation from vegetation [Wm-2]
SU_POSSIBILITIES(  5)=RAD_S_G_D           ! downward solar radiation over bare soil [Wm-2]
SU_POSSIBILITIES(  6)=RAD_S_G_U           ! upward solar radiation from bare soil [Wm-2]
SU_POSSIBILITIES(  7)=RAD_S_SN_F_D        ! downward solar radiation over snowy vegetation
[Wm-2]
SU_POSSIBILITIES(  8)=RAD_S_SN_F_U        ! upward solar radiation from snowy vegetation
[Wm-2]
SU_POSSIBILITIES(  9)=RAD_S_SN_G_D        ! downward solar radiation over snowy bare soil
[Wm-2]
SU_POSSIBILITIES( 10)=RAD_S_SN_G_U        ! upward solar radiation from snowy bare soil [Wm-2]
SU_POSSIBILITIES( 11)=RAD_S_SN_D          ! downward solar radiation over snow [Wm-2]
SU_POSSIBILITIES( 12)=RAD_S_SN_U          ! upward solar radiation from snow [Wm-2]
! longwave radiation
SU_POSSIBILITIES( 13)=RAD_L_D             ! downward total longwave radiation [Wm-2]
SU_POSSIBILITIES( 14)=RAD_L_U             ! upward total longwave radiation [Wm-2]
SU_POSSIBILITIES( 15)=RAD_L_F_D           ! downward longwave radiation over vegetation [Wm-2]
SU_POSSIBILITIES( 16)=RAD_L_F_U           ! upward longwave radiation from vegetation [Wm-2]
SU_POSSIBILITIES( 17)=RAD_L_G_D           ! downward longwave radiation over bare soil [Wm-2]
SU_POSSIBILITIES( 18)=RAD_L_G_U           ! upward longwave radiation from bare soil [Wm-2]
```

```
SU_POSSIBILITIES( 19)=RAD_L_FG_D          ! downward longwave radiation from veg. to bare
soil [Wm-2]
SU_POSSIBILITIES( 20)=RAD_L_FG_U          ! upward longwave radiation from bare soil to veg.
[Wm-2]
SU_POSSIBILITIES( 21)=RAD_L_SN_F_D        ! downward longwave radiation over snowy vegetation
[Wm-2]
SU_POSSIBILITIES( 22)=RAD_L_SN_F_U        ! upward longwave radiation from snowy vegetation
[Wm-2]
SU_POSSIBILITIES( 23)=RAD_L_SN_G_D        ! downward longwave radiation over snowy bare soil
[Wm-2]
SU_POSSIBILITIES( 24)=RAD_L_SN_G_U        ! upward longwave radiation from snowy bare soil
[Wm-2]
SU_POSSIBILITIES( 25)=RAD_L_G_TOT_D       ! downward longwave radiation over bare soil from
sky and veg. [Wm-2]
SU_POSSIBILITIES( 26)=RAD_L_G_TOT_U       ! upward longwave radiation from bare soil to sky
and veg. [Wm-2]
SU_POSSIBILITIES( 27)=RAD_L_SN_U          ! upward longwave radiation from snow [Wm-2]
! net radiation
SU_POSSIBILITIES( 28)=NR_A                ! total net radiation [Wm-2]
SU_POSSIBILITIES( 29)=NR_F                ! net radiation above vegetation [Wm-2]
SU_POSSIBILITIES( 30)=NR_G                ! net radiation above bare soil [Wm-2]
SU_POSSIBILITIES( 31)=NR_SN               ! net radiation above snow [Wm-2]
SU_POSSIBILITIES( 32)=NR_L_F_D            ! net downward radiation on vegetation without snow
[Wm-2]
SU_POSSIBILITIES( 33)=NR_L_F_U            ! net upward radiation from vegetation without snow
[Wm-2]
SU_POSSIBILITIES( 34)=NR_L_SN_D           ! total downward radiation on snow [Wm-2]
SU_POSSIBILITIES( 35)=NR_L_SN_U           ! total upward radiation from snow [Wm-2]
! sensible heat fluxes
SU_POSSIBILITIES( 36)=SHF_A               ! total sensible heat flux [Wm-2]
SU_POSSIBILITIES( 37)=SHF_F               ! sensible heat flux from vegetation [Wm-2]
SU_POSSIBILITIES( 38)=SHF_G               ! sensible heat flux from bare soil [Wm-2]
SU_POSSIBILITIES( 39)=SHF_SN              ! sensible heat flux from snow [Wm-2]
SU_POSSIBILITIES( 40)=SHF_SN_F            ! sensible heat flux from snowy vegetation [Wm-2]
SU_POSSIBILITIES( 41)=SHF_SN_G            ! sensible heat flux from snowy bare soil [Wm-2]
! latent heat fluxes
SU_POSSIBILITIES( 42)=LHF_A               ! total latent heat flux [Wm-2]
SU_POSSIBILITIES( 43)=LHF_F               ! latent heat flux from/on vegetation [Wm-2]
SU_POSSIBILITIES( 44)=LHF_F_D             ! transpiration flux [Wm-2]
SU_POSSIBILITIES( 45)=LHF_F_W     !*        ! latent heat flux from/on wet vegetation [Wm-2]
SU_POSSIBILITIES( 46)=LHF_G               ! latent heat flux from/on bare soil [Wm-2]
SU_POSSIBILITIES( 47)=LHF_G_D     !*        ! latent heat flux from/on dry bare soil [Wm-2]
SU_POSSIBILITIES( 48)=LHF_G_W     !*        ! latent heat flux from/on wet bare soil [Wm-2]
SU_POSSIBILITIES( 49)=LHF_SN              ! latent heat flux from/on snow [Wm-2]
SU_POSSIBILITIES( 50)=LHF_SN_F    !*        ! latent heat flux from/on snowy vegetation [Wm-2]
SU_POSSIBILITIES( 51)=LHF_SN_G    !*        ! latent heat flux from/on snowy bare soil [Wm-2]
! evaporation fluxes
SU_POSSIBILITIES( 52)=EVA_A               ! total evapotranspiration flux [kg m-2 s-1]
SU_POSSIBILITIES( 53)=EVA_F               ! evapotranspiration flux from/on vegetation [kg
m-2 s-1]
SU_POSSIBILITIES( 54)=EVA_F_D             ! transpiration flux [kg m-2 s-1]
SU_POSSIBILITIES( 55)=EVA_F_W             ! evaporation flux from/on wet vegetation [kg m-2
s-1]
SU_POSSIBILITIES( 56)=EVA_G               ! evaporation flux from/on bare soil [kg m-2 s-1]
SU_POSSIBILITIES( 57)=EVA_G_D             ! evaporation flux from/on dry bare soil [kg m-2
s-1]
SU_POSSIBILITIES( 58)=EVA_G_W             ! evaporation flux from/on wet bare soil [kg m-2
s-1]
SU_POSSIBILITIES( 59)=EVA_SN              ! evaporation flux from/on snow [kg m-2 s-1]
SU_POSSIBILITIES( 60)=EVA_SN_F            ! evaporation flux from/on snowy vegetation [kg m-2
s-1]
SU_POSSIBILITIES( 61)=EVA_SN_G            ! evaporation flux from/on snowy bare soil [kg m-2
s-1]
SU_POSSIBILITIES( 62)=ERR_VE !Will be filled with EVA_TRA array   ! layered total
transpiration flux [kg m-2 s-1]
! momentum fluxes
SU_POSSIBILITIES( 63)=UPWP                ! horizontal x-axis kinematic momentum flux [m2 s-2]
```

```
SU_POSSIBILITIES( 64)=VPWP                ! horizontal y-axis kinematic momentum flux [m2 s-2]
SU_POSSIBILITIES( 65)=USTAR               ! friction velocity [m s-1]
! conductive and non-radiative/turbulent fluxes
SU_POSSIBILITIES( 66)=BAL_FLU_F           ! effective imbalance of radiative and turbulent
fluxes for vegetation [Wm-2]
SU_POSSIBILITIES( 67)=BAL_FLU_G           ! effective imbalance of radiative and turbulent
fluxes for bare soil [Wm-2]
SU_POSSIBILITIES( 68)=BAL_FLU_SN          ! effective imbalance of radiative and turbulent
fluxes for snow [Wm-2]
SU_POSSIBILITIES( 69)=BAL_FLU_TOT         ! effective imbalance of radiative and turbulent
fluxes [Wm-2]
SU_POSSIBILITIES( 70)=CONF_SN_F           ! conductive heat flux between snow and vegetation
[Wm-2]
SU_POSSIBILITIES( 71)=CONF_SN_G           ! conductive heat flux between snow and bare soil
[Wm-2]
SU_POSSIBILITIES( 72)=CONF_SN_SRF         ! conductive heat flux between snow and
vegetation/bare soil [Wm-2]
SU_POSSIBILITIES( 73)=Q_RAIN_F            ! heat flux over vegetation by rain [Wm-2]
SU_POSSIBILITIES( 74)=Q_RAIN_G            ! heat flux over bare soil by rain [Wm-2]
! vegetation variables
SU_POSSIBILITIES( 75)=TEMP_F_STEP-DTK     ! temperature of vegetation [°C]
SU_POSSIBILITIES( 76)=TEMP_A_F-DTK        ! temperature of air within vegetation [°C]
SU_POSSIBILITIES( 77)=TEMP_ROOT-DTK       ! mean soil temperature in the root zone [°C]
SU_POSSIBILITIES( 78)=Q_A_F*1000.         ! specific humidity of air within vegetation [g/kg]
SU_POSSIBILITIES( 79)=VEG_COV_STEP        ! vegetation cover []
SU_POSSIBILITIES( 80)=LAI_STEP            ! leaf area index []
SU_POSSIBILITIES( 81)=REL_LEA_WAT_CON     ! relative leaf water content []
SU_POSSIBILITIES( 82)=LEA_WAT_CON*1000.   ! leaf water content [mm]
SU_POSSIBILITIES( 83)=VEG_HEI_STEP        ! vegetation height [m]
! soil temperature
SU_POSSIBILITIES( 84)=ERR_VE !Will be filled with TEM_SOI array   ! soil temperature [°C]
! soil moisture and ice
SU_POSSIBILITIES( 85)=ERR_VE !Will be filled with ETA_W array     ! volumetric water soil
content [m3/m3]
SU_POSSIBILITIES( 86)=ERR_VE !Will be filled with ETA_I array     ! volumetric ice soil
content [m3/m3]
SU_POSSIBILITIES( 87)=ERR_VE !Will be filled with SR_SOI array    ! soil saturation ratio for
liquid water [m3/m3]
! snow values
SU_POSSIBILITIES( 88)=COVER_SN            ! snow cover []
SU_POSSIBILITIES( 89)=COVER_SN_F          ! snow cover of vegetation fraction []
SU_POSSIBILITIES( 90)=COVER_SN_G          ! snow cover of bare soil fraction []
SU_POSSIBILITIES( 91)=HEI_SN              ! snow height [m]
SU_POSSIBILITIES( 92)=DENS_SN             ! snow density [kg/m3]
SU_POSSIBILITIES( 93)=HM_SN               ! snow water equivalent [m]
SU_POSSIBILITIES( 94)=HW_SN               ! snow water content [m]
SU_POSSIBILITIES( 95)=TEMP_SN              ! snow temperature [K]
SU_POSSIBILITIES( 96)=ALB_SN              ! snow albedo []
! photosynthesis
SU_POSSIBILITIES( 97)=GRO_CAR_ASS_RAT     ! gross rate of carbon assimilation by
photosynthesis [umol/m2s]
SU_POSSIBILITIES( 98)=NET_CAR_ASS_RAT     ! net rate of carbon assimilation by photosynthesis
[umol/m2s]
SU_POSSIBILITIES( 99)=RES_RAT             ! mitocondrial respiration [umol/m2s]
! cumulative variables
SU_POSSIBILITIES(100)=SUM_PREC            ! cumulated precipitation [m]
SU_POSSIBILITIES(101)=SUM_PREC_SN         ! cumulated snowfall in water equivalent units [m]
SU_POSSIBILITIES(102)=SUM_EVAP            ! cumulated evapotranspiration [m]
SU_POSSIBILITIES(103)=SUM_RUNO            ! cumulated surface runoff [m]
SU_POSSIBILITIES(104)=SUM_DRAI            ! cumulated underground drainage [m]
SU_POSSIBILITIES(105)=SUM_PREC_TMP*1000.  ! instantaneous cumulated precipitation [mm]
SU_POSSIBILITIES(106)=SUM_EVAP_TMP*1000.  ! instantaneous cumulated evapotranspiration [mm]
SU_POSSIBILITIES(107)=SUM_RUNO_TMP*1000.  ! instantaneous cumulated surface runoff [mm]
SU_POSSIBILITIES(108)=SUM_DRAI_TMP*1000.  ! instantaneous cumulated underground drainage [mm]
SU_POSSIBILITIES(109)=SUM_MIN_LW_TMP      ! minutes of leaf wetness in one timestep [min]
SU_POSSIBILITIES(110)=SUM_SOI_HEA_STO     ! heat storage into soil [Wm-2]
! other input values
```

```
SU_POSSIBILITIES(111)=TEMP_A_STEP-DTK     ! screen air temperature [°C]
SU_POSSIBILITIES(112)=VEL_MOD_STEP        ! screen wind speed [m/s]
SU_POSSIBILITIES(113)=RH_STEP             ! relative air humidity [%]
SU_POSSIBILITIES(114)=CL_TOT_STEP         ! total cloudiness []
SU_POSSIBILITIES(115)=CL_LOW_STEP         ! low cloudiness []
SU_POSSIBILITIES(116)=Q_SP_A_STEP         ! Specific humidity [g/kg]
SU_POSSIBILITIES(117)=PRES_A_STEP         ! Atmospheric pressure [hPa]
SU_POSSIBILITIES(118)=TEMP_DEW_A_STEP-DTK ! screen dew point temperature [°C]
SU_POSSIBILITIES(119)=CO2_CON_STEP        ! CO2 concentration [ppm]
! drag coefficient, resistances, and conductances
SU_POSSIBILITIES(120)=DRAG_H              ! bulk drag transfer coefficient for heat []
SU_POSSIBILITIES(121)=DRAG_M              ! bulk drag transfer coefficient for momentum []
SU_POSSIBILITIES(122)=DRAG_V              ! bulk drag transfer coefficient for water vapor []
SU_POSSIBILITIES(123)=RESIST_AH           ! bulk aerodynamic resistance for heat transfer
above vegetation [s/m]
SU_POSSIBILITIES(124)=RESIST_AM           ! bulk aerodynamic resistance for momentum transfer
above vegetation [s/m]
SU_POSSIBILITIES(125)=RESIST_AV           ! bulk aerodynamic resistance for water vapor
transfer above vegetation [s/m]
SU_POSSIBILITIES(126)=RESIST_B            ! laminar boundary layer resistance of air near the
foliage [s/m]
SU_POSSIBILITIES(127)=RESIST_D            ! aerodynamic resistance between vegetation and
soil [s/m]
SU_POSSIBILITIES(128)=RESIST_D_SN         ! aerodynamic resistance between vegetation and
snow [s/m]
SU_POSSIBILITIES(129)=RESIST_F            ! vegetation resistance to transpiration (from
Jarvis) [s/m]
SU_POSSIBILITIES(130)=RESIST_F_PHOTO      ! vegetation resistance to transpiration from
photosynthesis routine (active if LPHOTO) [s/m]
SU_POSSIBILITIES(131)=RESIST_LEAF         ! single leaf resistance to transpiration (from
Jarvis) [s/m]
SU_POSSIBILITIES(132)=RESIST_SRF          ! bare soil resistance to evaporation [s/m]
SU_POSSIBILITIES(133)=RESIST_SRF_SN       ! snow resistance to evaporation [s/m]
SU_POSSIBILITIES(134)=CONDUCT_AH          ! bulk aerodynamic conductance for heat [m/s]
SU_POSSIBILITIES(135)=CONDUCT_AM          ! bulk aerodynamic conductance for momentum [m/s]
SU_POSSIBILITIES(136)=CONDUCT_AV          ! bulk aerodynamic conductance for water vapor [m/s]
SU_POSSIBILITIES(137)=CONDUCT_B           ! laminar boundary layer conductance of air near
the foliage [m/s]
SU_POSSIBILITIES(138)=CONDUCT_D           ! aerodynamic conductance between vegetation and
soil [m/s]
SU_POSSIBILITIES(139)=CONDUCT_D_SN        ! aerodynamic conductance between vegetation and
snow [m/s]
SU_POSSIBILITIES(140)=COND_F_DRY          ! dry vegetation conductance to transpiration (from
Jarvis or photosynthesys) [m/s]
SU_POSSIBILITIES(141)=CONDUCT_F           ! vegetation conductance to transpiration (from
Jarvis or photosynthesys) [m/s]
SU_POSSIBILITIES(142)=COND_F_WET          ! wet vegetation conductance to transpiration (from
Jarvis or photosynthesys) [m/s]
SU_POSSIBILITIES(143)=COND_G_DRY          ! dry bare soil conductance to transpiration [m/s]
SU_POSSIBILITIES(144)=COND_G_WET          ! wet bare soil conductance to transpiration [m/s]
! other values: above soil
SU_POSSIBILITIES(145)=ALB_SD              ! dry bare soil albedo []
SU_POSSIBILITIES(146)=ALB_TOT             ! total surface albedo (vegetation, snow, bare
soil) []
SU_POSSIBILITIES(147)=GR_THEO             ! theoretical incident downward solar radiation []
SU_POSSIBILITIES(148)=HAZE_IND*100.       ! haze index probability(from 0 to 100) [%]
SU_POSSIBILITIES(149)=SOL_ALFA            ! solar elevation angle [deg]
SU_POSSIBILITIES(150)=REL_SRF_WAT_CON     ! relative soil water content []
SU_POSSIBILITIES(151)=REL_SRF_ICE_CON     ! relative soil water content []
SU_POSSIBILITIES(152)=SRF_WAT_CON*1000.   ! soil water content [mm]
SU_POSSIBILITIES(153)=SRF_ICE_CON*1000.   ! soil ice content [mm]
SU_POSSIBILITIES(154)=PREC_F              ! precipitation rate above vegetation []
SU_POSSIBILITIES(155)=PREC_G              ! precipitation rate above bare soil []
SU_POSSIBILITIES(156)=QSF*1000.           ! saturated specific humidity over vegetation [g/kg]
SU_POSSIBILITIES(157)=QSG*1000.           ! saturated specific humidity over bare soil [g/kg]
SU_POSSIBILITIES(158)=QSG_RH*1000.        ! relative humidity over bare soil [g/kg]
SU_POSSIBILITIES(159)=CAPPAMIXF           ! mixed vegetation-snow thermal diffusivity [m2/s]
```

```
SU_POSSIBILITIES(160)=CAPPAMIXG          ! mixed bare soil-snow thermal diffusivity [m2/s]
! other values: into soil
SU_POSSIBILITIES(161)=ERR_VE !Will be filled with CAPPAETA array
SU_POSSIBILITIES(162)=ERR_VE !Will be filled with CAPPAETAM array
SU_POSSIBILITIES(163)=ERR_VE !Will be filled with THEDIFM array
SU_POSSIBILITIES(164)=ERR_VE !Will be filled with MOIS_POT array
SU_POSSIBILITIES(165)=ERR_VE !Will be filled with FRP_OCC array
SU_POSSIBILITIES(166)=ERR_VE !Will be filled with ROC array
SU_POSSIBILITIES(167)=ERR_VE !Will be filled with ROCIVOL array
SU_POSSIBILITIES(168)=ERR_VE !Will be filled with SOILFLUX array
SU_POSSIBILITIES(169)=ERR_VE !Will be filled with DIFFSUMM array
SU_POSSIBILITIES(170)=ERR_VE !Will be filled with DIFFSUM array
SU_POSSIBILITIES(171)=ERR_VE !Will be filled with DIFFVTM array
SU_POSSIBILITIES(172)=ERR_VE !Will be filled with DIFFVT array
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!SU_POSSIBILITIES(85)=REL_SRF_ICE_CON

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!...6: LAM (76-96) : New dynamic filling
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
IF(IJK==1)WRITE(97,'(A)')TRIM(FORMAT_OUTPUT)
J_SU=1
DO I_POSSIB=1,NSUDYN
 IF(OUTPUT_VARIABLES(I_POSSIB))THEN
  IF(VARIABLE_NAME(I_POSSIB)=='SR_SOI')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_SR_SOI(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=SR_SOI(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='TEM_SOI')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_TEM_SOI(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=TEM_SOI(K_NUM_SOI_LAY)-DTK
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='CAPPAETA')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_CAPPAETA(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=CAPPAETA(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='CAPPAETAM')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_CAPPAETAM(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=CAPPAETAM(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='THEDIFM')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_THEDIFM(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=THEDIFM(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
```

```
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='FRP_OCC')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_FRP_OCC(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=FRP_OCC(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='DIFFSUMM')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_DIFFSUMM(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=DIFFSUMM(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='DIFFSUM')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_DIFFSUM(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=DIFFSUM(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='DIFFVT')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_DIFFVT(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=DIFFVT(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='DIFFVTM')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_DIFFVTM(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=DIFFVTM(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='EVATRA')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_EVATRA(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=EVATRA(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='MOIS_POT')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_MOIS_POT(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=MOIS_POT(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='SOILFLUX')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
```

```
    IF(LAYERS_OUT_SOILFLUX(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=SOILFLUX(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='ROC')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_ROC(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=ROC(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='ROCIVOL')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_ROCIVOL(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=ROCIVOL(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='ETA_W')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_ETA_W(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=ETA_W(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE IF(VARIABLE_NAME(I_POSSIB)=='ETA_I')THEN
   DO K_NUM_SOI_LAY=1,NUM_SOI_LAY
    IF(LAYERS_OUT_ETA_I(K_NUM_SOI_LAY))THEN
     SU6(J_SU)=ETA_I(K_NUM_SOI_LAY)
     IF(IJK==1) WRITE(97,'(I4,1X,A,"(",I2.2,")")') J_SU,&
      TRIM(VARIABLE_NAME(I_POSSIB)),K_NUM_SOI_LAY
     J_SU=J_SU+1
    END IF
   END DO
  ELSE
   SU6(J_SU)=SU_POSSIBILITIES(I_POSSIB)
   IF(IJK==1) WRITE(97,'(I4,1X,A)') J_SU,TRIM(VARIABLE_NAME(I_POSSIB))
    J_SU=J_SU+1
   END IF

  END IF
END DO


IF(IJK==1)CLOSE(97)

!************************
!* output data writing *
!************************
IF (NCOD > 0) THEN ! formatted
 IF (ABS(NCOD) >= 1 .AND. ABS(NCOD) <= 2) THEN ! standard
  WRITE(91,9001) IANNEW, IMENEW, IGINEW, IORANEW, IMINEW,(SU1(J),J=1,NSU1)
  WRITE(92,9001) IANNEW, IMENEW, IGINEW, IORANEW, IMINEW,(SU2(J),J=1,NSU2)
  WRITE(93,9001) IANNEW, IMENEW, IGINEW, IORANEW, IMINEW,(SU3(J),J=1,NSU3)
  WRITE(94,9001) IANNEW, IMENEW, IGINEW, IORANEW, IMINEW,(SU4(J),J=1,NSU4)
  WRITE(95,9001) IANNEW, IMENEW, IGINEW, IORANEW, IMINEW,(SU5(J),J=1,NSU5)
 ENDIF
 IF (ABS(NCOD) >= 2 .AND. ABS(NCOD) <= 3) THEN ! LAM
```

```
  WRITE(96,FMT=TRIM(FORMAT_OUTPUT))IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,&
   (SU6(J),J=1,NSU6)
 ENDIF
ELSE ! unformatted
 IF (ABS(NCOD) >= 1 .AND. ABS(NCOD) <= 2) THEN ! standard
  WRITE(91,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU1(J),J=1,NSU1)
  WRITE(92,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU2(J),J=1,NSU2)
  WRITE(93,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU3(J),J=1,NSU3)
  WRITE(94,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU4(J),J=1,NSU4)
  WRITE(95,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU5(J),J=1,NSU5)
 ENDIF
 IF (ABS(NCOD) >= 2 .AND. ABS(NCOD) <= 3) THEN ! LAM
  WRITE(96,REC=JCOUNT)IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,(SU6(J),J=1,NSU6)
 ENDIF
ENDIF !NCOD

RETURN
      FORMAT(I4.4,1x,4(I2.2,1x),50(1x,E16.7))
END SUBROUTINE DYNAMIC_MEMORYOUT

!***********************************************************************************
SUBROUTINE FLUXES(LSNOW,LVEG,NUM_SOI_LAY,PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,VEL_U_STEP,
VEL_V_STEP,&
 CAPPAMIXF,CAPPAMIXG,RP_LAYER,AIR_DENS,DEP_SOI,EVA_A,EVA_F,EVA_F_D,EVA_F_W,EVA_G,EVA_G_D,
 EVA_G_W,&
 EVA_SN,EVA_SN_F,EVA_SN_G,EVATRA,LHF_A,LHF_F,LHF_F_D,LHF_F_W,LHF_G,LHF_G_D,LHF_G_W,LHF_SN,
 LHF_SN_F,&
 LHF_SN_G,SHF_A,SHF_F,SHF_G, SHF_SN,HEI_SN,SHF_SN_F,SHF_SN_G,CONDUCT_D_SN,CONDUCT_SRF_SN,
 PREC_F,&
 PREC_G,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,Q_A_F,BAL_FLU_F,BAL_FLU_G,QSF,QSG,QSG_RH,CONF_SN_SRF
 ,&
 CONF_SN_F,CONF_SN_G,QSSN,NR_F,NR_G,NR_SN,CONDUCT_AH,CONDUCT_AM,CONDUCT_AV,CONDUCT_B,CONDUCT_D
 ,&
 COND_F_DRY,COND_F_WET,COND_G_DRY,COND_G_WET,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,&
 COND_SRF_VAP,TEMP_A_F,TEMP_ROOT,USTAR,UPWP,VPWP,H_OBS_ALL,Q_RAIN_F,Q_RAIN_G,JU_DAY,IORA,&
 IMINUTI,D_NR_F_D_TF,D_BAL_FLU_F_D_TF,BAL_FLU_SN,BAL_FLU_TOT)
!***********************************************************************************
! This routine calculates all latent, sensible, momentum and conductive fluxes
!***********************************************************************************
! Calls: POTENTIAL
! - Authors: C. Cassardo, J.J. Ji, E. Carena, P. Ramieri (13-Jul-1994)
! - Last revision: C. Cassardo (15-May-2014)
!***********************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL, INTENT(IN) :: PRES_A_STEP
REAL, INTENT(IN) :: Q_SP_A_STEP !current humidity at H_OBS_ALL level
REAL, INTENT(IN) :: TEMP_A_STEP !current air temperature at H_OBS_ALL level
REAL, INTENT(IN) :: VEL_U_STEP      !current x-axis wind component
REAL, INTENT(IN) :: VEL_V_STEP      !current y axis wind component
REAL, INTENT(IN) :: CAPPAMIXF
REAL, INTENT(IN) :: CAPPAMIXG
REAL,EXTERNAL :: LAT_HEA_EVA        !latent heat of vaporization/condensation of water
REAL,INTENT(IN) :: D_NR_F_D_TF
REAL :: D_EVA_F_D_TF
REAL :: D_EVA_F_W_D_TF
REAL :: D_EVA_F_D_D_TF
REAL :: D_LHF_F_D_TF
REAL,INTENT(OUT) :: BAL_FLU_SN
REAL :: D_SHF_F_D_TF
REAL,INTENT(OUT) :: D_BAL_FLU_F_D_TF
REAL :: D_QSF_D_TF
REAL :: D_Q_RAIN_F_D_TF
REAL :: D_CONF_SN_F_D_TF
REAL :: D_SPEC_UM
```

```
REAL :: D_THETAF_D_TF
REAL, INTENT(IN), DIMENSION(NUM_SOI_LAY) :: RP_LAYER  !percentage of roots
REAL, INTENT(IN) :: AIR_DENS    !density of dry air
REAL, INTENT(IN), DIMENSION(NUM_SOI_LAY) :: DEP_SOI
REAL, INTENT(OUT) :: EVA_A  !total water vapor flux
REAL, INTENT(OUT) :: EVA_F  !Canopy water vapor flux
REAL, INTENT(OUT) :: EVA_F_W    !wet canopy water vapor flux (>0=evap  <0=condens)
REAL, INTENT(OUT) :: EVA_G  !bare soil water vapor flux (>0=evap  <0=condens)
REAL :: EVA_G_D
REAL, INTENT(OUT) :: EVA_G_W
REAL, INTENT(OUT) :: EVA_SN !snow water flux
REAL :: EVA_SN_F                !snow canopy water flux
REAL :: EVA_SN_G                !snow bare soil water flux
REAL, INTENT(OUT), DIMENSION(NUM_SOI_LAY) :: EVATRA !transp water vapor flux
REAL :: EVA_F_D             !total transpiration water vapor flux
REAL, INTENT(OUT) :: LHF_A  !latent heat flux
REAL, INTENT(OUT) :: LHF_F  !canopy latent heat flux
REAL, INTENT(OUT) :: LHF_G  !bare soil latent heat flux
REAL :: FLUXMAX =1000.
REAL :: FLUXMIN = -1000.
REAL, INTENT(OUT) :: LHF_SN !snow latent heat flux
REAL, INTENT(OUT) :: LHF_SN_F   !snow latent heat flux
REAL, INTENT(OUT) :: LHF_SN_G   !snow latent heat flux
REAL, INTENT(OUT) :: LHF_F_D    !total latent heat transpiration flux
REAL, INTENT(OUT) :: LHF_F_W    !total latent heat transpiration flux
REAL, INTENT(OUT) :: LHF_G_D    !total latent heat transpiration flux
REAL, INTENT(OUT) :: LHF_G_W    !total latent heat transpiration flux
REAL, INTENT(OUT) :: SHF_A  !sensible heat flux
REAL, INTENT(OUT) :: SHF_F,SHF_G,SHF_SN !snow sensible heat flux
REAL, INTENT(IN) :: HEI_SN
REAL, INTENT(OUT) :: SHF_SN_F,SHF_SN_G
INTEGER :: I
INTEGER,INTENT(IN) :: IMINUTI
INTEGER,INTENT(IN) :: IORA
LOGICAL, INTENT (IN) :: LSNOW
INTEGER, INTENT(IN) :: LVEG !vegetation type code (see BATS)
INTEGER,INTENT(IN) :: JU_DAY
REAL, INTENT(IN) :: PREC_F  !precipitation intercepted by canopy
REAL, INTENT(IN) :: PREC_G  !precipitation reaching the soil
REAL :: PIPPO               !non useful var
REAL, INTENT(IN) :: P_TEMP_F_STEP       !current canopy temperature
REAL,DIMENSION(NUM_SOI_LAY), INTENT(IN) :: P_TEM_SOI
REAL, INTENT(IN) :: P_TEMP_SN   !snow temperature (previous step)
REAL, INTENT(IN) :: Q_A_F       !surf spec hum (avg aq, qcanopy and qsurf 8.32b G94)
REAL, INTENT(OUT) :: BAL_FLU_F  !heat flux for the canopy
REAL, INTENT(OUT) :: BAL_FLU_G  !heat flux entering into the soil
REAL, INTENT(OUT) :: BAL_FLU_TOT    ! effective imbalance between radiative and turbulent
heat fluxes [W m-2]
REAL, INTENT(OUT) :: Q_RAIN_F               !heat flux coming from rainfall over vegetation
REAL :: Q_RAIN_G                !heat flux coming from rainfall over bare soil
REAL, INTENT(IN) :: QSF     !canopy air specificy humidity, assumed saturated
REAL, INTENT(IN) :: QSG     !saturated bare soil specific humidity
REAL, INTENT(IN) :: QSG_RH
REAL, INTENT(OUT) :: CONF_SN_SRF
REAL, INTENT(OUT) :: CONF_SN_F
REAL, INTENT(OUT) :: CONF_SN_G
REAL, INTENT(IN) :: QSSN    !snow specific humidity, assumed saturated
REAL, INTENT(IN) :: NR_F        !canopy net radiation
REAL, INTENT(IN) :: NR_G        !bare soil net radiation
REAL, INTENT(IN) :: NR_SN   !snow net radiation
REAL, INTENT(IN) :: CONDUCT_AH
REAL, INTENT(IN) :: CONDUCT_AM      !air conductivity for momentum
REAL, INTENT(IN) :: CONDUCT_AV
REAL, INTENT(IN) :: CONDUCT_B   !BL conduct btw leaves and foliage air space for heat
REAL, INTENT(IN) :: CONDUCT_D   !conductivity between bare soil and canopy air space
REAL, INTENT(IN) :: CONDUCT_D_SN    !conductivity btw canopy and canopy air space for snow
REAL, INTENT(IN) :: COND_F_DRY  !dry canopy conduct for vapor
```

```
REAL, INTENT(IN) :: COND_F_WET  !wet canopy conduct for vapor
REAL, INTENT(IN) :: COND_G_DRY  !dry bare soil conduct for vapor
REAL, INTENT(IN) :: COND_G_WET  !wet bare soil conduct for vapor
REAL, INTENT(IN) :: VEG_COV_STEP    !vegetation cover
REAL, INTENT(IN) :: COVER_SN_F  !snow cover over vegetation
REAL, INTENT(IN) :: COVER_SN_G  !snow cover over bare soil
REAL, INTENT(IN) :: COVER_SN
REAL, INTENT(IN) :: COND_SRF_VAP    !snow cond btw bare soil and canopy air space for vapor
REAL, INTENT(IN) :: CONDUCT_SRF_SN  !snow conductivity btw bare soil and canopy air space
REAL, INTENT(IN) :: TEMP_A_F    !surf temp (avg at, TF and tsurf eq 8.32a G94)
REAL :: THETA               !generic potential temperature
REAL :: THETA_AT            !TEMP_A_STEP correspondent potential temperature
REAL :: THETA_TEMP_A_F      !TEMP_A_F correspondent potential temperature
REAL, INTENT(IN) :: TEMP_ROOT   !initial soil temperature
REAL :: UPWP                !vertical longitudinal momentum flux
REAL, INTENT(OUT) :: USTAR  !surface friction velocity
REAL :: VPWP                !vertical latitudinal momentum flux
REAL, INTENT(IN) :: H_OBS_ALL   !screen level height
REAL :: ZOVERL              !Monin-Obukhov nondimensional height

EVA_F_D=0.
D_EVA_F_D_D_TF=0.
D_QSF_D_TF=D_SPEC_UM(P_TEMP_F_STEP,PRES_A_STEP)

! Evaporation fluxes

IF (LVEG /= 14. .AND. LVEG /= 15. .AND. LVEG /= 12. .AND. LVEG /= 20. .AND. LVEG /= 27. &
 .AND. LVEG /= 28.) THEN ! over land

!================== VEGETATION

!canopy and ground below canopy portions (par. 8.4.2 of G94)
!fluxes are calculated with same parametrization

 IF (QSF >= Q_A_F) THEN   !moisture gradient <0 => evapotranspiration

!claudio
  !IF (TEMP_ROOT >= DTK) THEN !TF > 0^C => transpiration from dry canopy
   EVA_F_D = AIR_DENS * COND_F_DRY * (QSF - Q_A_F) * VEG_COV_STEP *(1. - COVER_SN_F) !eq.8.36
   G94
   D_EVA_F_D_D_TF=AIR_DENS*COND_F_DRY*(D_QSF_D_TF)*VEG_COV_STEP*(1.-COVER_SN_F)
  !END IF

!anyway there is evaporation: associated with wet canopy component
  EVA_F_W = AIR_DENS * COND_F_WET * (QSF - Q_A_F) * VEG_COV_STEP * (1. - COVER_SN_F)
  !eq.8.35 G94
  D_EVA_F_W_D_TF=AIR_DENS*COND_F_WET*D_QSF_D_TF*VEG_COV_STEP*(1.-COVER_SN_F)

 ELSE   !moisture grad >0 => condensation -> acting only RESIST_B

!no need to separate the two conditions: use only EVA_F_W for both EVA_F_W and EVA_F_D
  EVA_F_W = AIR_DENS * CONDUCT_B/2. * (QSF - Q_A_F) * VEG_COV_STEP * (1. - COVER_SN_F)
  !eq.8.35 G94
  D_EVA_F_W_D_TF=AIR_DENS*CONDUCT_B/2.*D_QSF_D_TF*VEG_COV_STEP*(1.-COVER_SN_F)

 END IF

!Partition of transpiration in root zone.
 DO I=1,NUM_SOI_LAY
  EVATRA(I) = RP_LAYER(I) * EVA_F_D
 END DO
!total vegetation evaporation/condensation flux [kg m-2 s-1]
 EVA_F = EVA_F_D + EVA_F_W
 D_EVA_F_D_TF = D_EVA_F_D_D_TF + D_EVA_F_W_D_TF

!================== SOIL
```

```
!bare soil portion of the ground
!par. 5.3.3 G94, also valid for bare soil (last lines pag 133)
!NB: if I will use parag. 5.3.4 I did not introduce RESIST_SRF and I have put QSG_RH
!instead of qsg

!partitioning of bare soil water vapor fluxes for the dry and wet soil portions
 IF (QSG_RH >= Q_A_F) THEN  ! humidity gradient <0 => evaporation

!eq.5.40 G94 and (15) pag 187 GandPC
  EVA_G_D = AIR_DENS * COND_G_DRY * (QSG_RH - Q_A_F) * (1.-VEG_COV_STEP)*(1.-COVER_SN_G)
  EVA_G_W = AIR_DENS * COND_G_WET * (QSG_RH - Q_A_F) * (1.-VEG_COV_STEP)*(1.-COVER_SN_G)
  EVA_G = EVA_G_D + EVA_G_W

 ELSE ! moisture gradient >0 => condensation, by intuition

  EVA_G = AIR_DENS * CONDUCT_D * (QSG_RH - Q_A_F) * (1.-VEG_COV_STEP)*(1.-COVER_SN_G)

 END IF

!================== SNOW

 EVA_SN_F = 0.
 EVA_SN_G = 0.
 EVA_SN = 0.

 IF (LSNOW) THEN

  EVA_SN_F =   AIR_DENS * CONDUCT_SRF_SN * (QSSN - Q_A_F) * VEG_COV_STEP *COVER_SN_F
  EVA_SN_G =   AIR_DENS * CONDUCT_SRF_SN * (QSSN - Q_A_F) * (1.-VEG_COV_STEP)*COVER_SN_G
  EVA_SN = EVA_SN_F + EVA_SN_G

 ENDIF

!================== TOTAL
 EVA_A = EVA_F + EVA_G + EVA_SN

ELSE
!ice (LVEG=12), water (14,15),dense settlement (20),asphalt (27,28)
!all components relative to vegetation are null as VEG_COV_STEP=0

 EVA_F_D=0.
 EVA_F_W=0.
 EVA_SN_F=0.
 EVA_F=0.
 D_EVA_F_D_TF = 0.

!================== SOIL

 IF (QSG_RH >= Q_SP_A_STEP) THEN  ! humidity gradient <0 => evaporation

  EVA_G_D = AIR_DENS * COND_G_DRY * (QSG_RH - Q_SP_A_STEP) * (1.- COVER_SN_G)
  EVA_G_W = AIR_DENS * COND_G_WET * (QSG_RH - Q_SP_A_STEP) * (1.- COVER_SN_G)
  EVA_G = EVA_G_D + EVA_G_W

 ELSE ! moisture gradient >0 => condensation

  EVA_G = AIR_DENS * CONDUCT_D * (QSG_RH - Q_SP_A_STEP) * (1.-COVER_SN_G)

 END IF

!================== SNOW

 EVA_SN_G=0.
 IF (LVEG==20 .OR. LVEG==27 .OR. LVEG==28) THEN !dense settlement, asphalt
  EVA_SN_G =   AIR_DENS * CONDUCT_SRF_SN * (QSSN - Q_A_F) * COVER_SN_G
 ENDIF
 EVA_SN=EVA_SN_G
```

```
!================== TOTAL

 EVA_A = EVA_F + EVA_G + EVA_SN

END IF

! Latent heat fluxes

LHF_A = EVA_A * LAT_HEA_EVA(TEMP_A_STEP)
LHF_F = EVA_F * LAT_HEA_EVA(P_TEMP_F_STEP)
D_LHF_F_D_TF = D_EVA_F_D_TF * LAT_HEA_EVA(P_TEMP_F_STEP)
LHF_G = EVA_G * LAT_HEA_EVA(P_TEM_SOI(1))
LHF_G_D = EVA_G_D * LAT_HEA_EVA(P_TEM_SOI(1))
LHF_G_W = EVA_G_W * LAT_HEA_EVA(P_TEM_SOI(1))
LHF_F_D = EVA_F_D * LAT_HEA_EVA(P_TEMP_F_STEP)
LHF_F_W = EVA_F_W * LAT_HEA_EVA(P_TEMP_F_STEP)
LHF_SN = 0.
LHF_SN_F = 0.
LHF_SN_G = 0.
IF (P_TEMP_SN /= ERR_VE) THEN
 LHF_SN = EVA_SN * LAT_HEA_EVA(P_TEMP_SN)
 LHF_SN_F = EVA_SN_F * LAT_HEA_EVA(P_TEMP_SN)
 LHF_SN_G = EVA_SN_G * LAT_HEA_EVA(P_TEMP_SN)
ENDIF

! Sensible heat fluxes

CALL POTENTIAL(PRES_A_STEP,Q_A_F,TEMP_A_F,THETA_TEMP_A_F,PIPPO)

IF (LVEG /= 12. .AND. LVEG /= 14. .AND. LVEG /= 15 .AND. LVEG/=20 .AND. LVEG/=27 .AND. &
 LVEG/=28) THEN ! over land

!================== VEGETATION

 CALL POTENTIAL(PRES_A_STEP,QSF,P_TEMP_F_STEP,THETA,PIPPO)
 D_THETAF_D_TF=THETA/P_TEMP_F_STEP
 SHF_F = AIR_DENS * CPAIR * CONDUCT_B * (THETA-THETA_TEMP_A_F)*VEG_COV_STEP*(1.-COVER_SN_F)
 D_SHF_F_D_TF=AIR_DENS*CPAIR*CONDUCT_B*D_THETAF_D_TF*VEG_COV_STEP*(1.-COVER_SN_F)
!eq.8.28 G94 weightened

!================== SOIL

 CALL POTENTIAL(PRES_A_STEP,QSG_RH,P_TEM_SOI(1),THETA,PIPPO)
 SHF_G = AIR_DENS*CPAIR*CONDUCT_D*(THETA-THETA_TEMP_A_F)*(1.-VEG_COV_STEP)*(1.-COVER_SN_G)
!eq.8.29 G94 weightened

!================== SNOW

 SHF_SN_F=0.
 SHF_SN_G=0.
 SHF_SN=0.
 IF (LSNOW) THEN
  CALL POTENTIAL(PRES_A_STEP,QSSN,P_TEMP_SN,THETA,PIPPO)
  SHF_SN_F= AIR_DENS * CPAIR * CONDUCT_D_SN * (THETA - THETA_TEMP_A_F)*VEG_COV_STEP*COVER_SN_F
  SHF_SN_G= AIR_DENS * CPAIR * CONDUCT_D_SN * (THETA - THETA_TEMP_A_F)*(1.-VEG_COV_STEP)*
  COVER_SN_G
  SHF_SN = SHF_SN_F + SHF_SN_G
 ENDIF

!================== TOTAL

 SHF_A = SHF_F + SHF_G + SHF_SN

ELSE
!ice (LVEG=12), water (14,15),dense settlement (20),asphalt (27,28)
!all components relative to vegetation are null as VEG_COV_STEP=0
```

```
 CALL POTENTIAL(PRES_A_STEP,QSG_RH,P_TEM_SOI(1),THETA,PIPPO)
 CALL POTENTIAL(PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,THETA_AT,PIPPO)

 SHF_F = 0.
 D_SHF_F_D_TF=0.

!================== SOIL

 SHF_G = AIR_DENS * CPAIR * CONDUCT_D * (THETA - THETA_AT) * (1.- COVER_SN_G)

!================== SNOW

 SHF_SN_F=0.
 SHF_SN_G=0.
 SHF_SN=0.

 IF (LSNOW) THEN

  IF (LVEG==20 .OR. LVEG==27 .OR. LVEG==28) THEN !dense settlement, asphalt
   CALL POTENTIAL(PRES_A_STEP,QSSN,P_TEMP_SN,THETA,PIPPO)
   SHF_SN_G= AIR_DENS * CPAIR * CONDUCT_D_SN * (THETA - THETA_AT) * COVER_SN_G
  END IF

 ENDIF

 SHF_SN=SHF_SN_G

!================== TOTAL

 SHF_A = SHF_F + SHF_G + SHF_SN

ENDIF

! Momentum fluxes

UPWP = - VEL_U_STEP * CONDUCT_AM  !eq 1.3 G94  (sign?)  (Usuolo=Vsuolo=0) Brutsaert 1982 check
VPWP = - VEL_V_STEP * CONDUCT_AM  !  "                  Arya 1988 check
USTAR=SQRT(MAX(0.01, SQRT(UPWP * UPWP + VPWP * VPWP) ))
! 0.01m/s = Cassardo threshold  eq.1.7 G94 (uncertainity on sign)

! calculation of Monin/Obuckov nondimensional height
ZOVERL = H_OBS_ALL*VON_KARMAN*GRAVITY*(SHF_A/AIR_DENS/CPAIR)/TEMP_A_STEP/(USTAR**3)

IF (ZOVERL < 0.) THEN
 ZOVERL = ZOVERL * SQRT(SQRT(1.-15.*ZOVERL))
ELSE
 ZOVERL = ZOVERL / (1.+4.7*ZOVERL)
ENDIF
ZOVERL=MAX(-5000.,MIN(5000.,ZOVERL)) ! limited to 5000

!...calculation of W* (see RAMS)
!PSIN=SQRT((1.-2.86*ZOVERL)/(1.+ZOVERL*(-5.39+ZOVERL*6.998)))   !why???

! heat flux coming from rainfall over vegetation

Q_RAIN_F = CSW*PREC_F*DENS_WAT*(TEMP_A_STEP-P_TEMP_F_STEP) ! [W/m2]
D_Q_RAIN_F_D_TF = -CSW*PREC_F*DENS_WAT
Q_RAIN_G = CSW*PREC_G*DENS_WAT*(TEMP_A_STEP-P_TEM_SOI(1)) ! [W/m2] 5.47a Dingman. Multiplied
for water specific heat = 4186

! soil/snow conductive heat flux

CONF_SN_F = 0.
D_CONF_SN_F_D_TF=0.
CONF_SN_G = 0.
CONF_SN_SRF = 0.
```

```
IF (LSNOW) THEN

! Calculates the vertical gradient of temp between avg point of snow and veg
!layers, and btw snow and soil layers
 CONF_SN_F=-2.*CAPPAMIXF*(P_TEMP_F_STEP-P_TEMP_SN)/(HEI_SN+0.001)*COVER_SN_F*VEG_COV_STEP
 D_CONF_SN_F_D_TF=-2.*CAPPAMIXF/(HEI_SN+0.001)*COVER_SN_F*VEG_COV_STEP
 CONF_SN_G=-2.*CAPPAMIXG*(P_TEM_SOI(1)-P_TEMP_SN)/(DEP_SOI(1)+HEI_SN)*COVER_SN_G*(1.-
 VEG_COV_STEP) !G94 eq.5.6
 CONF_SN_SRF=CONF_SN_F+CONF_SN_G ! total conductive heat flux exchanged btw snow and soil/veg

ENDIF

! effective imbalance of radiative and turbulent fluxes for vegetation [Wm-2]
BAL_FLU_F = NR_F - SHF_F - LHF_F + CONF_SN_F + Q_RAIN_F    !eq.5.5 G94 modified
D_BAL_FLU_F_D_TF=D_NR_F_D_TF-D_SHF_F_D_TF-D_LHF_F_D_TF+D_CONF_SN_F_D_TF+D_Q_RAIN_F_D_TF

! effective imbalance of radiative and turbulent fluxes for bare soil [Wm-2]
BAL_FLU_G = NR_G - SHF_G - LHF_G + CONF_SN_G + Q_RAIN_G    !eq.5.1 G94

! effective imbalance of radiative and turbulent fluxes for snow [Wm-2]
BAL_FLU_SN = 0.
IF (LSNOW) BAL_FLU_SN = NR_SN-CONF_SN_SRF-SHF_SN-LHF_SN

! effective imbalance of radiative and turbulent fluxes [Wm-2]
BAL_FLU_TOT = BAL_FLU_F + BAL_FLU_G + BAL_FLU_SN

!...rules of thumb to avoid extreme fluxes

!...above veg and bare soil

IF (ABS(LHF_A) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_A TOO LARGE:',&
  LHF_A,' -> ',FLUXMAX
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_A TOO LARGE:',&
  LHF_A,' -> ',FLUXMAX
 LHF_A=MIN(FLUXMAX,MAX(FLUXMIN,LHF_A))
 EVA_A=LHF_A/LAT_HEA_EVA(TEMP_A_STEP)
END IF

IF (ABS(SHF_A) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_A TOO LARGE:',&
  SHF_A,' -> ',FLUXMAX
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_A TOO LARGE:',&
  SHF_A,' -> ',FLUXMAX
 SHF_A=MIN(FLUXMAX,MAX(FLUXMIN,SHF_A))
END IF

!...vegetation

IF (ABS(LHF_F) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_F TOO LARGE:',&
  LHF_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_F TOO LARGE:',&
  LHF_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
 LHF_F=MIN(FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F),MAX(FLUXMIN*VEG_COV_STEP*(1.-COVER_SN_F),&
  LHF_F))
 EVA_F =LHF_F/LAT_HEA_EVA(P_TEMP_F_STEP)
END IF

IF (ABS(SHF_F) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_F TOO LARGE:',&
  SHF_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_F TOO LARGE:',&
  SHF_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
 SHF_F=MIN(FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F),MAX(FLUXMIN*VEG_COV_STEP*(1.-COVER_SN_F),&
  SHF_F))
END IF
```

```
IF (ABS(BAL_FLU_F) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - BAL_FLU_F TOO LARGE:',&
  BAL_FLU_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - BAL_FLU_F TOO LARGE:',&
  BAL_FLU_F,' -> ',FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F)
  BAL_FLU_F=MIN(FLUXMAX*VEG_COV_STEP*(1.-COVER_SN_F),MAX(FLUXMIN*VEG_COV_STEP*(1.-COVER_SN_F),
  &
   BAL_FLU_F))
END IF

!...bare soil

IF (ABS(LHF_G) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_G TOO LARGE:',&
  LHF_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_G TOO LARGE:',&
  LHF_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 LHF_G=MIN(FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_G),MAX(FLUXMIN*VEG_COV_STEP*&
  (1.-COVER_SN_F),LHF_G))
 EVA_G=LHF_G/LAT_HEA_EVA(P_TEM_SOI(1))
END IF

IF (ABS(SHF_G) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_G TOO LARGE:',&
  SHF_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_G TOO LARGE:',&
  SHF_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 SHF_G=MIN(FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_G),MAX(FLUXMIN*VEG_COV_STEP*&
  (1.-COVER_SN_F),SHF_G))
END IF

IF (ABS(BAL_FLU_G) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - BAL_FLU_G TOO LARGE:',&
  BAL_FLU_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - BAL_FLU_G TOO LARGE:',&
  BAL_FLU_G,' -> ',FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_F)
 BAL_FLU_G=MIN(FLUXMAX*(1.-VEG_COV_STEP)*(1.-COVER_SN_G),MAX(FLUXMIN*VEG_COV_STEP*&
  (1.-COVER_SN_F),BAL_FLU_G))
END IF

!...snow

IF (ABS(LHF_SN) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_SN TOO LARGE:',&
  LHF_SN,' -> ',FLUXMAX*COVER_SN
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - LHF_SN TOO LARGE:',&
  LHF_SN,' -> ',FLUXMAX*COVER_SN
 LHF_SN=MIN(FLUXMAX*COVER_SN,MAX(FLUXMIN*COVER_SN,LHF_SN))
 EVA_SN=LHF_SN/LAT_HEA_EVA(P_TEMP_SN)
END IF

IF (ABS(SHF_SN) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_SN TOO LARGE:',&
  SHF_SN,' -> ',FLUXMAX*COVER_SN
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - SHF_SN TOO LARGE:',&
  SHF_SN,' -> ',FLUXMAX*COVER_SN
 SHF_SN=MIN(FLUXMAX*COVER_SN,MAX(FLUXMIN*COVER_SN,SHF_SN))
END IF

IF (ABS(CONF_SN_SRF) > FLUXMAX) THEN
 WRITE(*,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - CONF_SN_SRF TOO LARGE:',&
  CONF_SN_SRF,' -> ',FLUXMAX*COVER_SN
 WRITE(99,'(I4,2I3,A,F7.0,A,F7.0)')JU_DAY,IORA,IMINUTI,' - CONF_SN_SRF TOO LARGE:',&
  CONF_SN_SRF,' -> ',FLUXMAX*COVER_SN
 CONF_SN_SRF=MIN(FLUXMAX*COVER_SN,MAX(FLUXMIN*COVER_SN,CONF_SN_SRF))
END IF
```

```
RETURN
END SUBROUTINE FLUXES

!*****************************************************************************
SUBROUTINE FREEZE(ICOD,FILEFRZ,IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,&
 SRF_WAT_CON,HW_SN,HM_SN,NUM_SOI_LAY,P_LEA_WAT_CON,P_SRF_WAT_CON,P_HM_SN,P_HW_SN,P_TEMP_F_STEP
 ,P_TEMP_SN,&
 REL_LEA_WAT_CON,REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,SUM_EVAP_TMP,SUM_RUNO,
 SUM_RUNO_TMP,SUM_DRAI,&
 SUM_DRAI_TMP,SUM_SOI_HEA_STO,TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,TEMP_ROOT,Q_A_F,
 Z0_M,ZER_DIS_LEV,&
 ILOGFUN,LSNOW,SR_SOI,P_TEM_SOI,TEM_SOI,ETA_W,P_ETA_W,ETA_I,P_ETA_I,PWI)
!*****************************************************************************
! Routine to store/read all UTOPIA data in a particular file (FILEFRZ).
! The operations are done according to the ICOD value:
! ICOD value    operation                    file dir.
!   1   read frz data (if one) at the beginning     input
!   2   write frz data at end of the simulation     output
!   3   write tmp frz data at intermediate times    output
!*****************************************************************************
! Calls: none
! - Author: C.C assardo (27-May-1994)
! - Last revision : C. Cassardo (15-May-2014)
!*****************************************************************************
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(INOUT) :: ALB_SN
REAL,INTENT(INOUT) :: DENS_SN
REAL,INTENT(INOUT) :: LEA_WAT_CON
REAL,INTENT(INOUT) :: SRF_WAT_CON
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_W
CHARACTER(len=*),INTENT(IN) :: FILEFRZ
REAL,INTENT(INOUT) :: HW_SN
REAL,INTENT(INOUT) :: HM_SN
INTEGER :: I
INTEGER,INTENT(INOUT) :: IAN
INTEGER,INTENT(IN) :: ICOD
INTEGER,INTENT(INOUT) :: IGI
LOGICAL,INTENT(INOUT) :: ILOGFUN
INTEGER,INTENT(INOUT) :: IME
INTEGER,INTENT(INOUT) :: IMI
INTEGER :: INDERR0
INTEGER :: INDERR1
INTEGER :: INDERR2
INTEGER,INTENT(INOUT) :: IORA
LOGICAL,INTENT(INOUT) :: LSNOW
REAL,INTENT(INOUT) :: P_LEA_WAT_CON
REAL,INTENT(INOUT) :: P_SRF_WAT_CON
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_W
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_I
REAL,INTENT(INOUT) :: P_HM_SN
REAL,INTENT(INOUT) :: P_HW_SN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: SR_SOI
REAL,INTENT(INOUT) :: P_TEMP_F_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_TEM_SOI
REAL,INTENT(INOUT) :: P_TEMP_SN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: PWI
REAL,INTENT(INOUT) :: REL_LEA_WAT_CON
REAL,INTENT(INOUT) :: REL_SRF_WAT_CON
REAL,INTENT(INOUT) :: SUM_PREC
REAL,INTENT(INOUT) :: SUM_PREC_TMP
REAL,INTENT(INOUT) :: SUM_EVAP
REAL,INTENT(INOUT) :: SUM_EVAP_TMP
REAL,INTENT(INOUT) :: SUM_RUNO
```

```
REAL,INTENT(INOUT) :: SUM_RUNO_TMP
REAL,INTENT(INOUT) :: SUM_DRAI
REAL,INTENT(INOUT) :: SUM_DRAI_TMP
REAL,INTENT(INOUT) :: SUM_SOI_HEA_STO
REAL,INTENT(INOUT) :: TEMP_F_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: TEM_SOI
REAL,INTENT(INOUT) :: TEMP_SN
REAL,INTENT(INOUT) :: SSR_FIE_CAP
REAL,INTENT(INOUT) :: SSR_WI
REAL,INTENT(INOUT) :: TEMP_A_F
REAL,INTENT(INOUT) :: TEMP_ROOT
REAL,INTENT(INOUT) :: Q_A_F
REAL,INTENT(INOUT) :: Z0_M
REAL,INTENT(INOUT) :: ZER_DIS_LEV

! open in/out files

IF (ICOD == 1) THEN

 OPEN(UNIT=50,FILE=FILEFRZ,STATUS='OLD',FORM='FORMATTED',IOSTAT=INDERR0)
 IF ( INDERR0 /= 0) THEN
  WRITE(99,*) 'WARNING: Error open INP frz file: ',FILEFRZ
  RETURN
 ENDIF

ELSE IF (ICOD == 2) THEN

 OPEN(UNIT=50,FILE=FILEFRZ,STATUS='UNKNOWN',FORM='FORMATTED',IOSTAT=INDERR1)
 IF ( INDERR1 /= 0) THEN
  WRITE(99,*) 'WARNING: Error open OUT frz file: ',FILEFRZ
  RETURN
 ENDIF

ELSE IF (ICOD == 3) THEN

 OPEN(UNIT=50,FILE=FILEFRZ,STATUS='UNKNOWN',FORM='FORMATTED',IOSTAT=INDERR2)
 IF ( INDERR2 /= 0) THEN
  WRITE(99,*) 'WARNING: Error open OUT TMP frz file: ',FILEFRZ
  RETURN
 ENDIF

ENDIF

IF (ICOD == 1) THEN
! 1 read frz data (if one) at the beginning

 READ(50,*)IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,ILOGFUN,LSNOW,P_LEA_WAT_CON,&
 P_SRF_WAT_CON,P_HM_SN,P_HW_SN,(SR_SOI(I),I=1,NUM_SOI_LAY),(P_TEM_SOI(I),I=1,NUM_SOI_LAY),&
 P_TEMP_F_STEP,P_TEMP_SN,REL_LEA_WAT_CON,REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,
 SUM_EVAP_TMP,SUM_RUNO,&
 SUM_RUNO_TMP,SUM_DRAI,SUM_DRAI_TMP,SUM_SOI_HEA_STO,(TEM_SOI(I),I=1,NUM_SOI_LAY),&
 TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,TEMP_ROOT,Q_A_F,Z0_M,ZER_DIS_LEV,&
 (ETA_W(I),I=1,NUM_SOI_LAY),(P_ETA_W(I),I=1,NUM_SOI_LAY),&
 (ETA_I(I),I=1,NUM_SOI_LAY),(P_ETA_I(I),I=1,NUM_SOI_LAY),&
 (PWI(I),I=1,NUM_SOI_LAY)

!...refreezing of all variables
 DO I=1,NUM_SOI_LAY
  TEM_SOI(I)=P_TEM_SOI(I)
 END DO
 LEA_WAT_CON=P_LEA_WAT_CON
 SRF_WAT_CON=P_SRF_WAT_CON
 HM_SN=P_HM_SN
 HW_SN=P_HW_SN
 TEMP_F_STEP=P_TEMP_F_STEP
 TEMP_SN=P_TEMP_SN
```

```
ELSE IF (ICOD == 2 .OR. ICOD == 3) THEN
! 2 write frz data at end of the simulation
! 3 write tmp frz data at intermediate times

 WRITE(50,*)IAN,IME,IGI,IORA,IMI,ALB_SN,DENS_SN,LEA_WAT_CON,ILOGFUN,LSNOW,P_LEA_WAT_CON,&
  P_SRF_WAT_CON,P_HM_SN,P_HW_SN,(SR_SOI(I),I=1,NUM_SOI_LAY),(P_TEM_SOI(I),I=1,NUM_SOI_LAY),&
  P_TEMP_F_STEP,P_TEMP_SN,REL_LEA_WAT_CON,REL_SRF_WAT_CON,SUM_PREC,SUM_PREC_TMP,SUM_EVAP,
  SUM_EVAP_TMP,SUM_RUNO,&
  SUM_RUNO_TMP,SUM_DRAI,SUM_DRAI_TMP,SUM_SOI_HEA_STO,(TEM_SOI(I),I=1,NUM_SOI_LAY),&
  TEMP_F_STEP,TEMP_SN,SSR_FIE_CAP,SSR_WI,TEMP_A_F,TEMP_ROOT,Q_A_F,Z0_M,ZER_DIS_LEV,&
  (ETA_W(I),I=1,NUM_SOI_LAY),(P_ETA_W(I),I=1,NUM_SOI_LAY),&
  (ETA_I(I),I=1,NUM_SOI_LAY),(P_ETA_I(I),I=1,NUM_SOI_LAY),&
  (PWI(I),I=1,NUM_SOI_LAY)

ENDIF
CLOSE (UNIT=50)

RETURN
END SUBROUTINE FREEZE

!*************************************************************************
SUBROUTINE GRAD(JU_DAY,IORA,IMI,ALON,ALAT,CLEG,PRES_A_STEP,R,CL_HIG_STEP,CL_LOW_STEP,RGM,
L360DAYS_CALENDAR)
!*************************************************************************
! Calculates the solar radiation at a specific site. The value is obtained using a
! theoretical astronomical formula for the direct component value and some empirical
! formulations for the diffuse one. The calculations are made both for clear and
! totally overcast sky. These two quantities are weighted with the cloudiness index
! CL_HIG_STEP and CL_LOW_STEP by considering separately the portion of sky covered by low
clouds,
! the portion of sky covered by high clouds, and the portion of clear sky.
! The formulae used are taken from Page (1986).
!*************************************************************************
! Calls: SOL_ANG
! - Authors: C. Cassardo, F. Mutinelli (15-Sep-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL, DIMENSION(6), PARAMETER ::  AA =&
 (/1.294,2.4417E-2,-3.973E-4,3.8034E-6,-2.2145E-8,5.8832E-11/)
REAL, INTENT(IN) :: ALAT        ! Latitude in degrees (-90 -> 90)
REAL, PARAMETER :: ALFAH = 0.5
REAL :: ALFAL
REAL, INTENT(IN) :: ALON        ! Longitude in degrees (-180 -> 180)
REAL, PARAMETER :: APR0 = 1013.25
REAL, INTENT(IN) :: PRES_A_STEP     ! Atmospheric pressure (hPa)
REAL :: ATI
REAL :: AX
REAL :: CAPPAD
REAL, INTENT(IN) :: CLEG        ! Code legal hour (1 -> legal hour, 0 -> no)
REAL, INTENT(IN) :: CL_HIG_STEP         ! High cloud cover (0->1)
REAL, INTENT(IN) :: CL_LOW_STEP         ! Low cloud cover (0->1)
REAL, EXTERNAL :: COSD
REAL :: DELTAR
REAL :: EMME
REAL :: ERD
REAL :: G
REAL :: GEIP
INTEGER :: I
INTEGER, INTENT(IN) :: IMI      ! Minutes (0 -> 60)
INTEGER, INTENT(IN) :: IORA     ! Hour (0 -> 24)
LOGICAL, INTENT(IN) :: L360DAYS_CALENDAR
INTEGER, INTENT(IN) :: JU_DAY       ! Julian day (1->365, 1=1 Jan)
REAL :: QAM
REAL, INTENT(IN) :: R           ! Term accounting local turbidity of sky (-.1/.1)
```

```
REAL :: RD_CLEAR
REAL :: RD_CLOUD
REAL :: RDM
REAL, INTENT(OUT) :: RGM        ! Global shortwave radiation (Wm-2)
REAL, PARAMETER  :: RI0 = 1367. ! Solar constant [W/m2]
REAL :: RI0J
REAL :: RIABS
REAL :: RIC
REAL :: RIM
REAL :: RIMH
REAL :: RIML
REAL :: RIMO
REAL, EXTERNAL :: SIND
REAL :: SING
REAL :: SUNSHINE
REAL :: TL
REAL :: TLG

TLG=0.

! calculation of solar elevation (sunrise --> 0 deg, zenit --> 90 deg)
CALL SOL_ANG(JU_DAY,IORA,IMI,ALON,ALAT,CLEG,G,L360DAYS_CALENDAR)

IF (G < 0.) THEN ! we are on nigth so solar radiation is null
 RGM=0.
 RETURN
END IF

SING=SIND(G)

! determination of coefficients for direct radiation
AX=0.
DO I=1,6
 AX=AX+AA(I)*G**(I-1) ! Page eq. 2.26
END DO

! determination of direct radiation
ATI=0.69051+0.00193*ALAT+R          ! atmospheric transp. index, Page tab a2.1
TL=22.76+0.0536*ALAT-27.78*ATI      ! air mass 2 turb. factor Page eq A2.6
IF (G <= 10.) THEN                  ! relative optical air mass Page
 EMME=PRES_A_STEP/APR0/(SING+0.15*((G+3.885)**(-1.253))) ! Page eq 2.17
ELSE
 EMME=PRES_A_STEP/APR0/SING             ! Page eq 2.18
END IF

IF(L360DAYS_CALENDAR)THEN
 GEIP=REAL(JU_DAY)
ELSE
 GEIP=360.* REAL(JU_DAY)/365.25     ! day angle, Page eq A1.2
END IF

CAPPAD=1.+0.03344*COSD(GEIP-2.80)   ! corr. for ellipt. orb. Page eq. 2.16
DELTAR=1./(0.9*EMME+9.4) !Rayleigh opt. thick. per opt. air mass, Page eq. 2.22

! Linke turbidity factor, Page eq. 2.23 and 2.24
IF (TL >= 2.5) TLG=TL-(0.85-2.25*SING+1.11*SING*SING)
IF (TL < 2.5) TLG=TL-(0.85-2.25*SING+1.11*SING*SING)*(TL-1)/1.5

RIC=CAPPAD*RI0*EXP(-DELTAR*EMME*TLG)    ! dir rad clear sky Page eq.2.14

! the following 3 formulations are taken from Mutinelli (1998)
RIMH=CL_HIG_STEP*EXP(-ALFAH*CL_HIG_STEP)*RIC        ! high clouds fraction
ALFAL=MIN(1.,2.*CL_LOW_STEP)
RIML=CL_LOW_STEP*EXP(-ALFAL*CL_LOW_STEP)*RIC        ! lowmed clouds fraction
RIMO=(1.-CL_HIG_STEP-CL_LOW_STEP)*RIC               ! clear sky fraction
RIM=RIMH+RIML+RIMO                  ! part. cloudy direct radiation
```

```
! evaluation of sunshine hours (in fraction) from cloudiness (in fraction) averaging values
in
! table II (pag. 1531) of Matuszko (2012)
SUNSHINE = 0.5 * ( (0.9509-0.223*CL_HIG_STEP-0.38*CL_HIG_STEP*CL_HIG_STEP) + (0.9221-0.381*
CL_LOW_STEP-0.38*CL_LOW_STEP*CL_LOW_STEP) )
SUNSHINE = MAX (0., MIN ( 1., SUNSHINE ) )
! determination of diffuse radiation
ERD=((0.444*SUNSHINE-2.98)*SUNSHINE+2.54)*SUNSHINE+1.0  ! polynomial correction factor, Page
eq. 5.13
QAM=AX*(0.506-0.010788*TLG)          ! atm. transm. coeff. by emme, Page eq.2.26
RI0J=RI0*CAPPAD                      ! extraterr. normal irradiance, Page eq. a1.11
RIABS=RI0J*(1.-QAM)                  ! absorbed irradiance, Page eq. 2.27
RD_CLEAR=0.5*(RI0J-RIC-RIABS)*SING   ! dif rad clear sky, Page eq. 2.25
RD_CLOUD=CAPPAD*(2.61+182.6*SING)    ! dif rad only cloudy sky, Page eq. 4.1

RDM=( (1.-CL_HIG_STEP-CL_LOW_STEP)*RD_CLEAR + (CL_HIG_STEP+CL_LOW_STEP)*RD_CLOUD ) * ERD !
diffuse radiation for generically cloud sky, Page eq. 5.12

RGM=RIM*SING+RDM                     ! total radiation

RETURN
END SUBROUTINE GRAD

!***********************************************************************
SUBROUTINE HAZE(ILOGFUN,RH_STEP,VEL_MOD_STEP,RAD_S_D,HAZE_IND)
!***********************************************************************
!  Calculation of a coefficient accounting for haze formation
!***********************************************************************
! Calls: none
! - Authors: C. Cassardo, E. Carena, F. Mutinelli (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!***********************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT (IN) :: RAD_S_D !  Global solar radiation
REAL :: ALFA
REAL,INTENT(IN) :: VEL_MOD_STEP !  Current wind speed
REAL, EXTERNAL :: COSD
REAL :: FA1
REAL :: FA2
REAL :: FA3
REAL,INTENT(OUT) :: HAZE_IND     !  Haze coefficient (no haze=0 <-> 1=haze)
LOGICAL,INTENT(IN) :: ILOGFUN   !  Logical variable: if false then FUN=0
REAL,INTENT(IN) :: RH_STEP  !  Relative humidity

HAZE_IND=0.
IF (.NOT.ILOGFUN) RETURN

! dependence on relative humidity (Mutinelli, 1998)
FA1=0.
IF (RH_STEP >= RHMIN) FA1=SQRT((RH_STEP-RHMIN)/(RHMAX-RHMIN))
IF (FA1 > 1) FA1=1. !  fa1 must be <= 1

! dependence on wind velocity (Mutinelli, 1998)
ALFA = 180.*(VEL_MOD_STEP - VMIN)/(VMAX - VMIN)
ALFA = MAX(0.,MIN(180.,ALFA)) ! 0 < ALFA < 180
FA2 = (1.+COSD(ALFA))/2.
FA2 = MAX(0.,MIN(1.,FA2))      !0 < FA2 < 1

! dependence on solar radiation (Ramieri, 1997)
ALFA = 180.*(RAD_S_D - AGRMAX)/AGRMAX
ALFA = MAX(-180.,MIN(0.,ALFA)) ! -180 < ALFA < 0
FA3 = (1.-COSD(ALFA))/2.
FA3 = MAX(0.,MIN(1.,FA3))

HAZE_IND=FA1*FA2*FA3 ! Cassardo et al. (1995)
```

```
HAZE_IND=MAX(0.,MIN(1.,HAZE_IND))

RETURN
END SUBROUTINE HAZE

!*****************************************************************************
SUBROUTINE ICEDIFF(DIFFWATER,NUM_SOI_LAY)
!*****************************************************************************
! Calculates the sum of the molecular and eddy diffusion coefficients [m2/s]
! TH_CO_W is the thermal conductivity of water (W m-1 K-1)
! ROCI_W is the heat capacity of water (J m-3 K-1)
! TH_CO_I is the thermal conductivity of ice (W m-1 K-1)
! ROCI_I is the heat capacity of ice (J m-3 K-1)
!*****************************************************************************
! Calls: none
! - Author: M. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
INTEGER :: I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFWATER

DO I=1, NUM_SOI_LAY-1
! here it is better to use thermal conductivity of ice
 DIFFWATER(I)=TH_CO_I/ROCI_I
END DO

RETURN
END SUBROUTINE ICEDIFF

!*****************************************************************************
SUBROUTINE INIT(ALB_TOT,NUM_SOI_LAY,ALAT,ALON,ESSE,SUM_DRAI,SUM_RUNO,SUM_PREC,SUM_EVAP,&
 SUM_SOI_HEA_STO,DELTAZ, DTOT,SSR_FIE_CAP,SSR_WI,TEMP_F_STEP,TEMP_ROOT,Q_A_F,LEA_WAT_CON,
 SRF_WAT_CON,&
 SRF_ICE_CON,ZER_DIS_LEV,PREC_OLD,PRS_AIR_OLD,GLORAD_OLD,VEL_U_OLD,VEL_V_OLD,TEM_AIR_OLD,
 RH_AIR_OLD,CL_TOT_OLD,CL_LOW_OLD,Q_AIR_OLD,PREC_CUR,&
 PRS_AIR_CUR,GLORAD_CUR,VEL_U_CUR,VEL_V_CUR,TEM_AIR_CUR,RH_AIR_CUR,CL_TOT_CUR,CL_LOW_CUR,
 Q_AIR_CUR,PREC_NEW,PRS_AIR_NEW,GLORAD_NEW,VEL_U_NEW,&
 VEL_V_NEW,TEM_AIR_NEW,RH_AIR_NEW,CL_TOT_NEW,CL_LOW_NEW,Q_AIR_NEW,LSNOW,P_HM_SN,P_HW_SN,
 TEMP_SN,GAMMA,&
 ROOT_DEP,VEG_HEI_STEP,HM_SN,DEP_SOI,MID_LAY_DEP,FRP_OCC,TEM_SOI,P_TEM_SOI,SR_SOI,SSR_FIE_CAPC
 ,ETA_WI,&
 ETA_S,TEM_SOI_I,SR_SOI_I,USTAR,Z0_SN,Z0_M,ETA_W,P_ETA_W,ETA_I,HW_SN,DENS_SN,ERR_ECO,&
 Z0_M_ECO,Z0_H_ECO,C_DREN,PWI,P_ETA_I,CL_HIG_STEP,EVA_G,EVA_G_W,EVA_F_W,TIM_E,VEG_HEI_OLD,
 VEG_HEI_CUR,&
 VEG_HEI_NEW,LAI_OLD,LAI_CUR,LAI_NEW,VEG_COV_OLD,VEG_COV_CUR,VEG_COV_NEW,CO2_CON_STEP,RESIST_B
 ,CUMTOT)
!*****************************************************************************
! Initializes all the variables of the model
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (27-Jun-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: ALAT
REAL,INTENT(INOUT) :: ALB_TOT
REAL,INTENT(IN) :: ALON
REAL,INTENT(OUT) :: PRS_AIR_NEW
REAL,INTENT(OUT) :: PRS_AIR_CUR
REAL,INTENT(OUT) :: PRS_AIR_OLD
```

```
REAL,INTENT(IN) :: GAMMA
REAL,INTENT(INOUT) :: C_DREN
REAL,INTENT(OUT) :: CL_LOW_NEW
REAL,INTENT(OUT) :: CL_LOW_CUR
REAL,INTENT(OUT) :: CL_LOW_OLD
REAL,INTENT(OUT) :: CL_TOT_NEW
REAL,INTENT(OUT) :: CL_TOT_CUR
REAL,INTENT(OUT) :: CL_TOT_OLD
REAL,INTENT(INOUT) :: CL_HIG_STEP
REAL,INTENT(INOUT) :: CO2_CON_STEP
REAL,EXTERNAL :: COSD
REAL,INTENT(OUT) :: CUMTOT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: FRP_OCC
REAL,INTENT(OUT) :: DELTAZ
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: DEP_SOI
REAL :: DEP_SOITOT
REAL,INTENT(OUT) :: DENS_SN
REAL,INTENT(OUT) :: ZER_DIS_LEV
REAL,INTENT(IN) :: ROOT_DEP
REAL,INTENT(OUT) :: DTOT
REAL,INTENT(OUT) :: EVA_G
REAL,INTENT(OUT) :: EVA_G_W
REAL,INTENT(OUT) :: EVA_F_W
REAL,INTENT(OUT) :: LEA_WAT_CON
REAL,INTENT(OUT) :: SRF_WAT_CON
REAL,INTENT(OUT) :: SRF_ICE_CON
REAL,INTENT(IN) :: ERR_ECO
REAL,INTENT(IN) :: ESSE
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: ETA_W
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_WI
REAL,INTENT(OUT) :: GLORAD_NEW
REAL,INTENT(OUT) :: GLORAD_CUR
REAL,INTENT(OUT) :: GLORAD_OLD
REAL,INTENT(IN) :: VEG_HEI_STEP
REAL,INTENT(INOUT) :: HM_SN
REAL,INTENT(OUT) :: VEG_HEI_NEW
REAL,INTENT(OUT) :: VEG_HEI_CUR
REAL,INTENT(OUT) :: VEG_HEI_OLD
REAL,INTENT(OUT) :: HW_SN
INTEGER :: I
INTEGER :: J
REAL,INTENT(OUT) :: LAI_NEW
REAL,INTENT(OUT) :: LAI_CUR
REAL,INTENT(OUT) :: LAI_OLD
LOGICAL,INTENT(OUT) :: LSNOW
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: P_ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: P_ETA_W
REAL,INTENT(OUT) :: P_HM_SN
REAL,INTENT(OUT) :: P_HW_SN
REAL,INTENT(OUT) :: PREC_NEW
REAL,INTENT(OUT) :: PREC_CUR
REAL,INTENT(OUT) :: PREC_OLD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: SR_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: P_TEM_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: PWI
REAL,INTENT(OUT) :: Q_A_F
REAL,INTENT(OUT) :: SSR_FIE_CAP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SSR_FIE_CAPC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SR_SOI_I
REAL,INTENT(OUT) :: Q_AIR_NEW
REAL,INTENT(OUT) :: Q_AIR_CUR
REAL,INTENT(OUT) :: Q_AIR_OLD
REAL,INTENT(OUT) :: SSR_WI
REAL,INTENT(OUT) :: RESIST_B
REAL,INTENT(OUT) :: RH_AIR_NEW
```

```
REAL,INTENT(OUT) :: RH_AIR_CUR
REAL,INTENT(OUT) :: RH_AIR_OLD
REAL,INTENT(OUT) :: VEG_COV_NEW
REAL,INTENT(OUT) :: VEG_COV_CUR
REAL,INTENT(OUT) :: VEG_COV_OLD
REAL,EXTERNAL :: SIND
REAL,INTENT(OUT) :: SUM_PREC
REAL,INTENT(OUT) :: SUM_EVAP
REAL,INTENT(OUT) :: SUM_SOI_HEA_STO
REAL,INTENT(OUT) :: SUM_RUNO
REAL,INTENT(OUT) :: SUM_DRAI
REAL,INTENT(OUT) :: TEM_AIR_NEW
REAL,INTENT(OUT) :: TEM_AIR_CUR
REAL,INTENT(OUT) :: TEM_AIR_OLD
REAL,INTENT(OUT) :: TEMP_F_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: TEM_SOI_I
REAL(KIND=DBL_PREC),INTENT(OUT) :: TIM_E
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: TEM_SOI
REAL,INTENT(OUT) :: TEMP_ROOT
REAL,INTENT(OUT) :: TEMP_SN
REAL,INTENT(OUT) :: VEL_U_NEW
REAL,INTENT(OUT) :: VEL_U_CUR
REAL,INTENT(OUT) :: VEL_U_OLD
REAL,INTENT(OUT) :: USTAR
REAL,INTENT(OUT) :: VEL_V_NEW
REAL,INTENT(OUT) :: VEL_V_CUR
REAL,INTENT(OUT) :: VEL_V_OLD
REAL,INTENT(INOUT) :: Z0_H_ECO
REAL,INTENT(OUT) :: Z0_M
REAL,INTENT(INOUT) :: Z0_M_ECO
REAL,INTENT(OUT) :: Z0_SN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: MID_LAY_DEP

! total albedo
ALB_TOT = 0.3

! CO2 concentration [ppm = umol/mol]
CO2_CON_STEP=ERR_VE

! correction of depths for sloping surfaces
!DO I=1,NUM_SOI_LAY
! DEP_SOI(I)=DEP_SOI(I)/COSD(ESSE) !???
!ENDDO

! initializations of cumulative variables
SUM_DRAI= 0.   ! surface runoff
SUM_RUNO= 0.   ! undergroung drainage
SUM_PREC=   0.   ! precipitation
SUM_EVAP=   0.   ! evaporation
SUM_SOI_HEA_STO= 0.   ! heat flux
CUMTOT=  0.   ! vegetation heat index

! calculation of height of the middle of the i-th layer (MID_LAY_DEP)
DO I=1,NUM_SOI_LAY
 MID_LAY_DEP(I)=0.
 DO J=1,I-1
  MID_LAY_DEP(I)=MID_LAY_DEP(I)+DEP_SOI(J)
 ENDDO
 MID_LAY_DEP(I)=MID_LAY_DEP(I)+DEP_SOI(I)/2. !Height of middle of layer
ENDDO

! total layer depth
DEP_SOITOT=0.
DO I=1,NUM_SOI_LAY
 DEP_SOITOT=DEP_SOITOT+DEP_SOI(I)
ENDDO
```

```
! In the case in which root depth is larger than total soil depth: even if it
! is a wrong assumption, we make hypothesis that all transpiration occurs from
! the total soil depth and not from total root depth as the portion of roots
! below total soil depth are in unknown conditions so previous equations are
! kept by using DELTAZ instead of ROOT_DEP

DELTAZ=MIN(DEP_SOITOT,ROOT_DEP)

! calculation of coefficients accounting root distribution (roots are
! considered uniformly distributed in the layer 0-DELTAZ)

DO I=1,NUM_SOI_LAY
 FRP_OCC(I) = 0.
 DTOT = 0.
 DO J=1,I
  DTOT = DTOT+DEP_SOI(J) ! total soil depth
 END DO
 IF (DTOT <= DELTAZ) THEN
  FRP_OCC(I) = DEP_SOI(I)/DELTAZ ! percentage of root soil depth (linear)
 ELSE
  FRP_OCC(I) = MAX(0.,(DEP_SOI(I)-(DTOT-DELTAZ))/DELTAZ) ! root fraction in i-th layer
 END IF ! FRP_OCC=relative weigth of soil levels in EVATRA
END DO


! calculation of wilting point and field capacity mean values in root layer
SSR_FIE_CAP = 0. ! field capacity (mean in root zone)
SSR_WI = 0. ! wilting point (mean in root zone)
DO I=1,NUM_SOI_LAY
 SSR_FIE_CAP = SSR_FIE_CAP+FRP_OCC(I)*SSR_FIE_CAPC(I)
 SSR_WI = SSR_WI+FRP_OCC(I)*ETA_WI(I)/ETA_S(I)
 TEM_SOI(I) = TEM_SOI_I(I) !inizialization of actual and previos time step temp.and moisture
 with initial values
 P_TEM_SOI(I) = TEM_SOI_I(I)
 SR_SOI(I) = SR_SOI_I(I)
 ETA_W(I) = SR_SOI_I(I)*ETA_S(I)
 P_ETA_W(I) = ETA_W(I)
 ETA_I(I) = 0.      ! soil ice initialised to ZERO
 P_ETA_I(I) = 0.
 PWI(I) = 0. ! soil freezing initialization
ENDDO

! initialization of canopy variables
TEMP_F_STEP = ERR_VE ! it is put =ERR_VE and then changed in UTOPIA at the first time
TEMP_ROOT = TEM_SOI_I(1)                              !
Q_A_F = 0.01        ! kg/kg
LEA_WAT_CON = 0. ! initialized to ZERO m of water
Z0_M = 0.05
Z0_M_ECO = ERR_ECO ! roughness length for momentum, initialized to error
Z0_H_ECO = ERR_ECO ! roughness length for heat, initialized to error
ZER_DIS_LEV = 0.1*VEG_HEI_STEP  ! initial value
SRF_WAT_CON=0.
SRF_ICE_CON=0.

! initializations of other variables
CL_HIG_STEP=0. ! high level cloudiness
USTAR=0.01 ! friction velocity

! initialization of input data running variables: OLD values
PREC_OLD    = ERR_VE
PRS_AIR_OLD  = ERR_VE
GLORAD_OLD   = ERR_VE
VEL_U_OLD    = ERR_VE
VEL_V_OLD    = ERR_VE
TEM_AIR_OLD   = ERR_VE
RH_AIR_OLD = ERR_VE
CL_TOT_OLD   = ERR_VE
```

```
CL_LOW_OLD  = ERR_VE
Q_AIR_OLD   = ERR_VE
VEG_HEI_OLD = ERR_VE
LAI_OLD = ERR_VE
VEG_COV_OLD= ERR_VE

! current (NM) values
PREC_CUR     = ERR_VE
PRS_AIR_CUR   = ERR_VE
GLORAD_CUR    = ERR_VE
VEL_U_CUR     = ERR_VE
VEL_V_CUR     = ERR_VE
TEM_AIR_CUR    = ERR_VE
RH_AIR_CUR  = ERR_VE
CL_TOT_CUR    = ERR_VE
CL_LOW_CUR   = ERR_VE
Q_AIR_CUR    = ERR_VE
VEG_HEI_CUR = ERR_VE
LAI_CUR = ERR_VE
VEG_COV_CUR= ERR_VE

! NEW values
PREC_NEW    = ERR_VE
PRS_AIR_NEW  = ERR_VE
GLORAD_NEW   = ERR_VE
VEL_U_NEW    = ERR_VE
VEL_V_NEW    = ERR_VE
TEM_AIR_NEW   = ERR_VE
RH_AIR_NEW = ERR_VE
CL_TOT_NEW   = ERR_VE
CL_LOW_NEW  = ERR_VE
Q_AIR_NEW   = ERR_VE
VEG_HEI_NEW = ERR_VE
LAI_NEW = ERR_VE
VEG_COV_NEW= ERR_VE

! initialization of snow variables
Z0_SN=0.001
LSNOW   = .FALSE.
IF (HM_SN > 0.) LSNOW=.TRUE.
P_HM_SN = HM_SN
P_HW_SN = 0.
HW_SN = 0.
TEMP_SN = DTK
DENS_SN=ERR_VE
IF (P_HM_SN>0.) DENS_SN=100.

! drainage
IF (C_DREN == -1.) C_DREN=MIN(1.,SIND(ESSE))   ! semi-permeable soil

EVA_G=0.
EVA_G_W=0.
EVA_F_W=0.
TIM_E=0.

! resistance in laminar layer within vegetation (needed by photosynthesis routine)
RESIST_B=100.

RETURN
END SUBROUTINE INIT

!***************************************************************************
SUBROUTINE ISTHERESNOW(PREC_STEP,PRES_A_STEP,TEMP_A_STEP,AIR_DENS,TIMESTEP_SEC,EVA_F_W,
LEA_WAT_CON,&
 LEA_WAT_CON_MAX,HM_SN,LSNOW,LSNOW1,LSNOW2,LSNOW3,P_LEA_WAT_CON,PREC_F,PREC_G,PREC_ON_SNOW,&
 P_HM_SN,PRECSN,SUM_PREC_SN,P_TEMP_F_STEP,RH_STEP,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,
 TWET)
```

```
!*****************************************************************************
! Checks if there is snow and calculates logical variables
!*****************************************************************************
! Calls: WATER_MASS, WETBULB_TEMP
! - Author: C. Cassardo (27-Jun-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: PREC_STEP
REAL,INTENT(IN) :: PRES_A_STEP
REAL,INTENT(IN) :: TEMP_A_STEP
REAL,INTENT(IN) :: AIR_DENS
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_F_W
REAL,INTENT(INOUT) :: LEA_WAT_CON
REAL,INTENT(IN) :: LEA_WAT_CON_MAX
REAL,INTENT(INOUT) :: HM_SN
LOGICAL,INTENT(INOUT) :: LSNOW
LOGICAL,INTENT(OUT) :: LSNOW1
LOGICAL,INTENT(OUT) :: LSNOW2
LOGICAL,INTENT(OUT) :: LSNOW3
REAL,INTENT(IN) :: P_LEA_WAT_CON
REAL,INTENT(OUT) :: PREC_F
REAL,INTENT(INOUT) :: PREC_G
REAL,INTENT(OUT) :: PREC_ON_SNOW
REAL,INTENT(IN) :: P_HM_SN
REAL,INTENT(OUT) :: PRECSN
REAL,INTENT(INOUT) :: SUM_PREC_SN
REAL,INTENT(IN) :: P_TEMP_F_STEP
REAL,INTENT(IN) :: RH_STEP
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN_F
REAL,INTENT(IN) :: COVER_SN_G
REAL,INTENT(IN) :: COVER_SN
REAL,INTENT(OUT) :: TWET
REAL :: WETBULB_TEMP

! check if there is old snow
LSNOW1=(P_HM_SN>0. .OR.(P_HM_SN==0 .AND. HM_SN>0.)) !old snow

! check if new snow is precipitating: snow will occur if:
!   1) there is precipitation
!   2) TWET<1.^C - thus snow can occur even for T>0 C if TWET<1.^C
TWET=WETBULB_TEMP(RH_STEP,TEMP_A_STEP,PRES_A_STEP,AIR_DENS)
LSNOW2=(PREC_STEP>0..AND.TWET<(DTK+1.))

! water redistribution between canopy, rain, snow and DENS_WAT
CALL WATER_MASS(PREC_STEP,TIMESTEP_SEC,EVA_F_W,LEA_WAT_CON,LEA_WAT_CON_MAX,LSNOW2,LSNOW3,&
 P_LEA_WAT_CON,PREC_F,PREC_G,PREC_ON_SNOW,PRECSN,SUM_PREC_SN,P_TEMP_F_STEP,VEG_COV_STEP,
 COVER_SN_F,&
 COVER_SN_G,COVER_SN)

! at the end, a logical variable resuming snow presence is calculated
! There is snow (SNOW=.TRUE.) if:
!    1-there is old snow, or
!    2-it's snowing now, or
!    3-there is frost on soil/canopy
LSNOW=(LSNOW1.OR.LSNOW2.OR.LSNOW3)

RETURN
END SUBROUTINE ISTHERESNOW

!*****************************************************************************
SUBROUTINE READ_INP(ALBEDO,ESSE,GAMMA,NDATI,QUOTA,ALON,ALAT,CLEG,LITUFA2,&
 JU_DAY,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,I1,I2,I3,I4,I5,&
```

```
 IER,TIM_E,GR_THEO,KDATA,LLONGWFLAG,L360DAYS_CALENDAR,JCOUNT,ACCESSO,VEGFLAG,&
 IC_IN,I_INT,I_SLO)
!******************************************************************************
! Read and store current values of input variables in the following order in
! the vector DOUT:
!
! y     1   Air temperature                (^C)        270     330
!       2   Atmospheric pressure           (hPa)       600     1060
! y*    3   Specific humidity              (kg/kg)     1.3e-5  630
! y**   4   Total cloudiness               (Fract)     0       1
! y     5   X component wind velocity U    (m/s)       -50     50
! y     6   Y component wind velocity V    (m/s)       -50     50
! y     7   Precipitation rate (rain+snow) (mm/h)      0       72
! y**   8   Low cloudiness                 (Fract)     0       1
! y**   9   Sw downward radiation          (W/m2)      0       1367
! y*    10  Relative humidity              (%)         0       100
! y**   11  Long wave downward radiation   (W/m2)      0       600
! y***  12  vegetation height             (m)         0.1     30
! y***  13  LAI                            (-)         1       20
! y***  14  vegetation cover fraction      (-)         0       1
! y***  15  CO2                            (ppm)       100     1000
!
! (y)   The y near datum means that the variable is necessary.
! (*)   Data of specific humidity and relative humidity are complementary.
! (**)  Data of cloudiness and global radiation are complementary.
! (***) Non mandatory data
!
! UTOPIA expects to have NDATI columns for any record, the first one for the time
! (n. of minutes from the starting time) and the following columns for all data.
! Then, the vector IC must contain the column number (inclusive of first column) of
! the M input data in the order listed above. If any data is missing, a negative
! number should be given to IC As example, if the input data contains:
! time, U wind, total cloud., low cloud., V wind, pressure, relative humidity,
! precipitation, temp., the following values should be given: M/8/,
! IC/9,6,-1,3,2,5,8,4,-1,7/
!******************************************************************************
! Calls : GRAD, RH_UMID, SPECUM
! - Author: C. Cassardo (11-Aug-1999)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

!INTEGER,INTENT(IN) :: M
CHARACTER(len=*), INTENT(IN) :: ACCESSO
REAL, INTENT(IN) :: ALAT
REAL, INTENT(IN) :: ALBEDO
REAL, INTENT(IN) :: ALON
REAL, INTENT(IN) :: ESSE
REAL, INTENT(IN) :: GAMMA
REAL, INTENT(IN) :: CLEG
INTEGER, DIMENSION(5) :: DATAOUT
INTEGER, INTENT(IN) :: JCOUNT
INTEGER, PARAMETER :: NDMAX=40
INTEGER, DIMENSION(MDAT_IN),INTENT(IN) :: IC_IN
REAL, DIMENSION(MDAT_IN),INTENT(IN) :: I_INT
REAL, DIMENSION(MDAT_IN),INTENT(IN) :: I_SLO
REAL,DIMENSION(MDAT_IN) ::  DIN
REAL,DIMENSION(MDAT_IN) :: DMAX = (/330.,1060.,630.,1.,50.,50.,72.,1.,&
                        .,100.,600.,30.,20.,1.,1000./)
REAL,DIMENSION(MDAT_IN) :: DMIN = (/200.,600.,1.3E-5,0.,-50.,-50.,0.,&
                    .,0.,0.,0.,0.1,1.,0.,100./)
REAL,DIMENSION(MDAT_IN) ::  DOUT
REAL :: GR_THEO0
REAL,INTENT(OUT) :: GR_THEO
INTEGER :: I
INTEGER,INTENT(OUT) :: I1
```

```
INTEGER,INTENT(OUT) :: I2
INTEGER,INTENT(OUT) :: I3
INTEGER,INTENT(OUT) :: I4
INTEGER,INTENT(OUT) :: I5
INTEGER,DIMENSION(12) :: IDAYZER =(/0,31,59,90,120,151,181,212,242,272,303,333/)
INTEGER, INTENT(OUT) :: IER
INTEGER :: J
INTEGER, INTENT(IN) :: KDATA
LOGICAL,INTENT(IN) :: L360DAYS_CALENDAR
LOGICAL, INTENT(OUT) :: LLONGWFLAG  !true if the long wave rad. is avaiable
INTEGER, INTENT(IN) :: NDATI
INTEGER, INTENT(OUT) :: JU_DAY
REAL :: PIPPO
REAL :: QSAT
REAL, INTENT(IN) :: QUOTA
REAL :: RATIO
REAL :: RGM
REAL :: RH_UMID
REAL, INTENT(IN) :: LITUFA2
REAL(KIND=DBL_PREC), INTENT(INOUT) :: TIM_E
REAL, INTENT(OUT) :: X1
REAL, INTENT(OUT) :: X2
REAL, INTENT(OUT) :: X3
REAL, INTENT(OUT) :: X4
REAL, INTENT(OUT) :: X5
REAL, INTENT(OUT) :: X6
REAL, INTENT(OUT) :: X7
REAL, INTENT(OUT) :: X8
REAL, INTENT(OUT) :: X9
REAL, INTENT(OUT) :: X10
REAL, INTENT(OUT) :: X11
REAL, INTENT(OUT) :: X12
REAL, INTENT(OUT) :: X13
REAL, INTENT(OUT) :: X14
REAL, INTENT(OUT) :: X15

LOGICAL, INTENT(OUT) :: VEGFLAG

DIN(:)=ERR_VE
DOUT(:)=ERR_VE
! check number of input data
IER=0
IF (NDATI > NDMAX) THEN
 IER = 1
 PRINT *,'NDATI>NDMAX => Please change'
 RETURN
END IF

! data reading
IF(ACCESSO=='SEQUENTIAL')THEN
 READ(71,*,IOSTAT=IER) (DATAOUT(J),J=1,5),(DIN(J),J=1,NDATI)
ELSE IF(ACCESSO=='DIRECT')THEN
 READ(71,REC=JCOUNT,IOSTAT=IER)(DATAOUT(J),J=1,5),(DIN(J),J=1,NDATI)
ELSE
 IER = -999
 RETURN
END IF

IF(IER/=0)THEN
 IER=-999
 RETURN
END IF

! determination of number of minutes since beginning of observations
TIM_E=TIM_E+60./REAL(KDATA)

! calculates date
```

```
IF(L360DAYS_CALENDAR)THEN
 JU_DAY=30*(DATAOUT(2)-1)+DATAOUT(3)
ELSE
 JU_DAY=IDAYZER(DATAOUT(2))+DATAOUT(3)
END IF

! extract input data and store it in DOUT in the correct order
! after that, linearly correct input data according with regression line in input (see
routine MEMORYIN)
DO I=1,MDAT_IN
 IF (IC_IN(I) > 0) THEN
  DOUT(I)=DIN(IC_IN(I))
  DOUT(I)=I_INT(I)+I_SLO(I)*DOUT(I)
 ENDIF
END DO

! first check: the data 1,5,6,7,(3or10),((4and8)or9) must be not error
IF (DOUT(1) == ERR_VE.OR.DOUT(5) == ERR_VE.OR.DOUT(6) == ERR_VE.OR.&
 DOUT(7) == ERR_VE.OR.(DOUT(3) == ERR_VE.AND.DOUT(10) == ERR_VE).OR.&
 (DOUT(9) == ERR_VE.AND.(DOUT(4) == ERR_VE.OR.DOUT(8) == ERR_VE))) THEN
 WRITE(99,9001) (DATAOUT(J),J=1,5),(DOUT(J),J=1,MDAT_IN)
 IER = 3
 PRINT *,'INPUT_ERROR'
 DO I=1,MDAT_IN
  WRITE(99,'("Variable n. ",I2," = ",G15.5)')I,DOUT(I)
 ENDDO
 RETURN
END IF

! check on pressure data (if error it uses hydrostatic approximation)
IF (DOUT(2)<= 0) DOUT(2)=1013.25-QUOTA*DENS_DRY_AIR*GRAVITY/100.

! check on temperature data (if in C translate in K)
IF (DOUT(1) < 100.) DOUT(1)=DOUT(1)+DTK

! check on humidity
IF (IC_IN(3) > 0) THEN !it is specific humidity

 IF (DOUT(3) /= ERR_VE) DOUT(3)=DOUT(3)/1000. ! conversion from g/kg to kg/kg

! basic check: if specific humidity is higher than saturated value, the saturated value is
assumed and relative humidity is set to 100 %
 CALL SPECUM(100.,DOUT(1),DOUT(2),QSAT,PIPPO,PIPPO)
 IF (DOUT(3) > QSAT) THEN
  WRITE(99,9003) (DATAOUT(J),J=1,5),DOUT(3),QSAT
  DOUT(3)=QSAT
  DOUT(10)=100.
 ELSE

! evaluation of relative humidity starting from observed specific humidity
  DOUT(10)=RH_UMID(DOUT(3),DOUT(1),DOUT(2))
 END IF
END IF

IF (IC_IN(10) > 0) THEN !it is relative humidity --> evaluation of spec. hum. from rel. hum.
 CALL SPECUM(DOUT(10),DOUT(1),DOUT(2),DOUT(3),PIPPO,PIPPO)
END IF

! precipitation data: the conversion from mm/h to m/s (divide by 3.6E6 as mm-->m: 10^3, h-->s
3.6 10^3) is
! not done here but directly when the numerical value of PREC_STEP is used. This because if
we divide the datum
! by 10^6 it may become too small and exceed the precision of REAL*4

IF(DOUT(11)==ERR_VE)THEN
 LLONGWFLAG=.FALSE.
ELSE
```

```
 LLONGWFLAG=.TRUE.
END IF

IF(DOUT(12)==ERR_VE)THEN !se non sono presenti i dati H,LAI,SIGMA
 VEGFLAG=.FALSE.
ELSE
 VEGFLAG=.TRUE.
END IF

! some rules of thumb to check input data
! each datum should be in the range DIN <= DOUT <= DMAX
DO I=1,MDAT_IN
 IF (DOUT(I) /= ERR_VE) THEN
  IF (DOUT(I) > DMAX(I) .OR. DOUT(I) < DMIN(I)) THEN
   WRITE(99,9000) I,DOUT(I),DMIN(I),DMAX(I),(DATAOUT(J),J=1,5)
   DOUT(I)=MAX(DMIN(I),MIN(DMAX(I),DOUT(I)))
  END IF
 END IF
END DO

! cloudiness set to 1 if error in case of precipitation
IF (DOUT(4) == ERR_VE .AND. DOUT(7)>0.) THEN
 DOUT(4)=1.
ENDIF

! check low cloudiness: must be not larger than total one
IF (DOUT(8) /= ERR_VE) THEN
 IF (DOUT(8) > DOUT(4)) THEN
  DOUT(8)=MAX(DMIN(8),MIN(DOUT(8),DOUT(4)))
  WRITE(99,9002) (DATAOUT(J),J=1,5)
 END IF
END IF

! correct low cloudiness if error and high cloudiness not error: low=high/2
IF (DOUT(8) == ERR_VE) THEN
 IF (DOUT(4) /= ERR_VE)THEN
  DOUT(8)=DOUT(4)/2.
  DOUT(8)=MAX(DMIN(8),MIN(DOUT(8),DOUT(4)))
 END IF
END IF

! calculation of solar radiation if no observations
IF (DOUT(9) == ERR_VE) THEN
 IF (DOUT(4) == ERR_VE.OR.DOUT(8) == ERR_VE) THEN
  WRITE(99,*)'Basic radiation & cloud. data missing: impossible to run UTOPIA'
  PRINT *,'Basic radiation & cloud. data missing: impossible to run UTOPIA'
  IER = 4
  RETURN
 END IF
 CALL GRAD(JU_DAY,DATAOUT(4),DATAOUT(5),ALON,ALAT,CLEG,DOUT(2),LITUFA2,&
  (DOUT(4)-DOUT(8)),DOUT(8),RGM,L360DAYS_CALENDAR)
! solar radiation on tilted surface calculated only if surface is tilted
 IF (ESSE/=0.) THEN
  IF (RGM>0.) THEN
   CALL RAD_TILT(ALBEDO,CLEG,JU_DAY,ESSE,GAMMA,DATAOUT(4),DATAOUT(5),ALAT,ALON,&
    RGM,DOUT(9))
! in case solar radiation on horizontal surface is null (i.e. before sunrise
! or after sunset) it is null also on tilted surface
  ELSE
   DOUT(9)=0.
  ENDIF
 ELSE
  DOUT(9)=RGM
 ENDIF
 DOUT(9)=MAX(0.,DOUT(9))
END IF
```

```
! calculation of theoretical clear sky solar downward radiation
CALL GRAD(JU_DAY,DATAOUT(4),DATAOUT(5),ALON,ALAT,CLEG,DOUT(2),LITUFA2,0.,0.,GR_THEO0,&
 L360DAYS_CALENDAR)
! solar radiation on tilted surface calculated only if surface is tilted
IF (ESSE/=0.) THEN
 IF (GR_THEO0>0.) THEN
  CALL RAD_TILT(ALBEDO,CLEG,JU_DAY,ESSE,GAMMA,DATAOUT(4),DATAOUT(5),ALAT,ALON,&
   GR_THEO0,GR_THEO)
 ELSE ! in case solar radiation on horizontal surface is null (i.e. before sunrise or after
 sunset) it is null also on tilted surface
  GR_THEO=0.
 ENDIF
ELSE
 GR_THEO=GR_THEO0
ENDIF
GR_THEO=MAX(0.,GR_THEO)

! calculation of cloudiness from solar radiation if no observations: the variable
! GR_THEO is evaluated without cloudiness and the ratio between GR_THEO and observed
! is assumed as total cloudiness (limited by 0 and 1)
! During night cloudiness=0 (no other possibilities)
! Low cloudiness assumed as 0.5 times total one.
!???In realtà si potrebbe calcolare dalla longwave (vedi appunti Corea)
IF(.NOT.LLONGWFLAG)THEN
 IF (DOUT(4) == ERR_VE.AND.DOUT(8) == ERR_VE) THEN
  IF (DOUT(9) == ERR_VE) THEN
   WRITE(99,*)'Basic data of rad. and cloud. are missing: impossible to run UTOPIA'
   IER = 4
   PRINT *, 'Basic data of rad. and cloud. are missing: impossible to run UTOPIA'
   RETURN
  END IF
  DOUT(4)=0.
  IF (GR_THEO > 0.) THEN
   RATIO=1.-DOUT(9)/GR_THEO
   DOUT(4)=MAX(0.,MIN(1.,RATIO))
   DOUT(4)=MAX(DMIN(4),MIN(DMAX(4),DOUT(4)))
  END IF
  DOUT(8)=DOUT(4)/2.
 END IF
END IF

! storage of data read in single variables
X1 =DOUT(1)
X2 =DOUT(2)
X3 =DOUT(3)
X4 =DOUT(4)
X5 =DOUT(5)
X6 =DOUT(6)
X7 =DOUT(7)
X8 =DOUT(8)
X9 =DOUT(9)
X10=DOUT(10)
X11=DOUT(11)
X12=DOUT(12)
X13=DOUT(13)
X14=DOUT(14)
X15=DOUT(15)

I1 =DATAOUT(1)
I2 =DATAOUT(2)
I3 =DATAOUT(3)
I4 =DATAOUT(4)
I5 =DATAOUT(5)

RETURN
      FORMAT('Corrected input n. ',I2.2,' = ',F18.2,'out of range ',F8.2,'-',F8.2,&
  ' at time ',I4.4,4I2.2)
```

```
      FORMAT('Some input data at time ',I4.4,4I2.2,' are wrong:',/,50F10.2)
       FORMAT('At time ',i4.4,4I2.2,' low cloudiness inconsistent ','with total: corrected')
      FORMAT('At time ',i4.4,4I2.2,' specific humidity= ',E12.4,' too large :',&
      ' corrected with QSAT=',E12.4)
END SUBROUTINE READ_INP

!******************************************************************************
SUBROUTINE LONGW(Q_SP_A_STEP,TEMP_A_STEP,CL_TOT_STEP,D_NR_L_F_D_D_TF,D_NR_L_F_U_D_TF,EPSF,EPSG
,&
 HAZE_IND,LLONGWFLAG,NSOIL,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,RAD_L_G_D,RAD_L_G_U,RAD_L_D,
 NR_L_F_D,RAD_L_SN_F_D,RAD_L_SN_F_U,&
 NR_L_F_U,RAD_L_G_TOT_D,RAD_L_G_TOT_U,RAD_L_SN_G_D,RAD_L_SN_G_U,RAD_L_SN_U,NR_L_SN_D,NR_L_SN_U
 ,RAD_L_U,RAD_L_F_D,RAD_L_F_U,RAD_L_FG_D,RAD_L_FG_U,VEG_COV_STEP,&
 COVER_SN_F,COVER_SN_G)
!******************************************************************************
! Routine to evaluate the balance terms for longwave radiation
! (all expressed in W m-2) following Mutinelli (1998) chapter 4.
!******************************************************************************
! Calls: none
! - Authors: C. Cassardo, F. Mutinelli (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NSOIL
REAL,INTENT(IN) :: Q_SP_A_STEP
REAL,INTENT(IN) :: TEMP_A_STEP
REAL,INTENT(IN) :: CL_TOT_STEP
REAL,INTENT(OUT) :: D_NR_L_F_D_D_TF
REAL,INTENT(OUT) :: D_NR_L_F_U_D_TF
REAL :: D_RAD_L_F_U_D_TF
REAL :: D_RAD_L_FG_D_D_TF
REAL :: D_RAD_L_FG_U_D_TF
REAL,INTENT(IN) :: EPSF
REAL,INTENT(IN) :: EPSG
REAL,INTENT(IN) ::  HAZE_IND
LOGICAL,INTENT(IN) :: LLONGWFLAG
!INTEGER,INTENT(IN) :: NSOIL
REAL,INTENT(IN) :: P_TEMP_F_STEP
REAL,DIMENSION(NSOIL),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: P_TEMP_SN
REAL,INTENT(OUT) :: RAD_L_G_D
REAL,INTENT(OUT) :: RAD_L_G_U
REAL,INTENT(INOUT) :: RAD_L_D
REAL,INTENT(OUT) :: NR_L_F_D
REAL,INTENT(OUT) :: RAD_L_SN_F_D
REAL,INTENT(OUT) :: RAD_L_SN_F_U
REAL,INTENT(OUT) :: NR_L_F_U
REAL,INTENT(OUT) :: RAD_L_G_TOT_D
REAL,INTENT(OUT) :: RAD_L_G_TOT_U
REAL,INTENT(OUT) :: RAD_L_SN_G_D
REAL,INTENT(OUT) :: RAD_L_SN_G_U
REAL,INTENT(OUT) :: RAD_L_SN_U
REAL,INTENT(OUT) :: NR_L_SN_D
REAL,INTENT(OUT) :: NR_L_SN_U
REAL,INTENT(OUT) :: RAD_L_U
REAL,INTENT(OUT) :: RAD_L_F_D
REAL,INTENT(OUT) :: RAD_L_F_U
REAL,INTENT(OUT) :: RAD_L_FG_D
REAL,INTENT(OUT) :: RAD_L_FG_U
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN_F
REAL,INTENT(IN) :: COVER_SN_G

! total downward emitted from atmosphere
IF(.NOT.LLONGWFLAG)THEN
```

```
 RAD_L_D=(1.+0.22*CL_TOT_STEP*CL_TOT_STEP+0.22*(1.-CL_TOT_STEP*CL_TOT_STEP)*HAZE_IND)*(0.67*&
  (1670*Q_SP_A_STEP)**0.08)*BOZ*(TEMP_A_STEP**4)
END IF

! total upward emitted from snow
RAD_L_SN_U=EPSSN*BOZ*P_TEMP_SN**4 + (1.-EPSSN)*RAD_L_D
! downward above vegetation without snow
RAD_L_F_D=RAD_L_D*VEG_COV_STEP*(1.-COVER_SN_F)
! upward above vegetation without snow
RAD_L_F_U=(EPSF*BOZ*P_TEMP_F_STEP**4)*VEG_COV_STEP*(1.-COVER_SN_F)+(1.-EPSF)*RAD_L_F_D
! downward above bare soil not covered by vegetation without snow
RAD_L_G_D=RAD_L_D*(1.-VEG_COV_STEP)*(1.-COVER_SN_G)
! upward above bare soil not covered by vegetation without snow
RAD_L_G_U=(EPSG*BOZ*P_TEM_SOI(1)**4)*(1.-VEG_COV_STEP)*(1.-COVER_SN_G)+(1.-EPSG)*RAD_L_G_D
! downward from vegetation to bare soil covered by vegetation without snow
RAD_L_FG_D=(EPSF*BOZ*P_TEMP_F_STEP**4+(1.-EPSF)*EPSG*BOZ*P_TEM_SOI(1)**4)/(EPSF+EPSG-EPSF*EPSG
)*VEG_COV_STEP
! upward to vegetation from bare soil covered by vegetation without snow
RAD_L_FG_U=(EPSG*BOZ*P_TEM_SOI(1)**4+(1.-EPSG)*EPSF*BOZ*P_TEMP_F_STEP**4)/(EPSF+EPSG-EPSF*EPSG
)*VEG_COV_STEP
! downward to bare soil without snow
RAD_L_G_TOT_D=RAD_L_G_D+RAD_L_FG_D
! upward from bare soil without snow
RAD_L_G_TOT_U=RAD_L_G_U+RAD_L_FG_U
! net downward radiation on vegetation without snow
NR_L_F_D=RAD_L_F_D-RAD_L_FG_D
! net upward radiation from vegetation without snow
NR_L_F_U=RAD_L_F_U-RAD_L_FG_U
! downward on vegetation covered by snow
RAD_L_SN_F_D=VEG_COV_STEP*COVER_SN_F*RAD_L_D
! upward from vegetation covered by snow
RAD_L_SN_F_U=VEG_COV_STEP*COVER_SN_F*RAD_L_SN_U
! downward on bare soil covered by snow
RAD_L_SN_G_D=(1.-VEG_COV_STEP)*COVER_SN_G*RAD_L_D
! upward from bare soil covered by snow
RAD_L_SN_G_U=(1.-VEG_COV_STEP)*COVER_SN_G*RAD_L_SN_U
! total downward on snow
NR_L_SN_D=RAD_L_SN_F_D+RAD_L_SN_G_D
! total upward from snow
NR_L_SN_U=RAD_L_SN_F_U+RAD_L_SN_G_U
! total emitted from surface (snow+vegetation+bare soil)
RAD_L_U=RAD_L_F_U+RAD_L_G_U+NR_L_SN_U

! derivatives of values with respect to TEMP_F_STEP (only those non null)
D_RAD_L_F_U_D_TF=4.*(EPSF*BOZ*P_TEMP_F_STEP**3)*VEG_COV_STEP*(1.-COVER_SN_F)
D_RAD_L_FG_D_D_TF=4.*(EPSF*BOZ*P_TEMP_F_STEP**3)/(EPSF+EPSG-EPSF*EPSG)*VEG_COV_STEP
D_RAD_L_FG_U_D_TF=(4.*(1.-EPSG)*EPSF*BOZ*P_TEMP_F_STEP**3)/(EPSF+EPSG-EPSF*EPSG)*VEG_COV_STEP
D_NR_L_F_D_D_TF=-D_RAD_L_FG_D_D_TF
D_NR_L_F_U_D_TF=D_RAD_L_F_U_D_TF-D_RAD_L_FG_U_D_TF

RETURN
END SUBROUTINE LONGW

!*****************************************************************************
REAL FUNCTION LOUIS (CDN, Z0, RIC_NUM, ICOD)
!*****************************************************************************
! Calculates the modification F of the neutral drag coefficient CDN due to the
! stability, according to the quantity (ICOD=1 -> momentum, =2 -> heat, =3 -> water
! vapor) and gives the non-neutral drag coefficient CD.
! Formulations are taken from G94 (1994), paragraph 8.5.2 pag. 243-4.
!*****************************************************************************
! Calls: none
! - Authors: C. Cassardo, P. Ramieri (01-Dec-1997)
! - Last revision : C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE
```

```
REAL :: B
REAL :: BS
REAL, INTENT(IN) :: CDN     !drag or bulk neutral transfer coefficient
REAL :: F
INTEGER, INTENT(IN) :: ICOD !icod=1 -> momentum, =2 -> heat, =3 -> water vapor
REAL, INTENT(IN) :: RIC_NUM  !Richardson number
REAL, INTENT(IN) :: Z0      !ratio between height and roughness length

BS=0.
IF (ICOD == 1) THEN ! momentum
 BS = - 0.34/SQRT(MAX(CDN,EPSIL))+13.7
ELSE IF (ICOD == 2 .OR. ICOD == 3) THEN ! heat and water vapor
 BS= - 0.18/SQRT(MAX(CDN,EPSIL))+6.3
ELSE
 WRITE (*,*) "ICOD not allowed in LOUIS"
 STOP
ENDIF

B = BS * CDN * 9.4 * SQRT(Z0)

IF (RIC_NUM < 0.) THEN
 F= 1 - 9.4 * RIC_NUM/(1 + B * SQRT(-RIC_NUM))      ! eq. 8.44 G94
ELSE IF  (RIC_NUM > 0.) THEN
 F= 1 / (1 + 4.7 * RIC_NUM)**2.          ! eq. 8.45 G94
ELSE
 F=1
END IF

LOUIS = MAX(EPSIL,CDN*F)    !see eqn. 3.42 e 8.41 G94 and intuition

RETURN
END FUNCTION LOUIS

!*******************************************************************************
SUBROUTINE UTOPIA_DRIVER(IMI, INP_SOIL, IORA, JU_DAY, LVEG, MITER, NUM_SOI_LAY, PTW, &
 AIR_DENS, ALAT, ALB_FH, ALON, &
 C_DREN, CL_LOW_CUR, CL_LOW_NEW, CL_TOT_CUR, CL_TOT_NEW, CLEG, CO2_CON_CUR, CO2_CON_NEW, &
 D0_VEGPAR, DELT, DELTAZ, EPS, ERR_ECO, ESSE, &
 GAMMA, GLORAD_NEW, GLORAD_CUR, &
 HAJ, HAV, HDJ, HDV, JOPT, &
 LAI_CUR, LAI_NEW, LONRAD_CUR, LONRAD_NEW, MRS, &
 PREC_NEW, PREC_CUR, PRS_AIR_CUR, PRS_AIR_NEW, PVMAX25, &
 Q_AIR_CUR, Q_AIR_NEW, &
 RGL_NOILHAN, RESIST_F_MIN, RH_AIR_CUR, RH_AIR_NEW, &
 SSR_FIE_CAP, SSR_WI, &
 TEM_AIR_NEW, TEM_AIR_CUR, TIMESTEP_SEC, TOPTJ, TOPTV, TORTUOS, &
 VEG_COV_CUR, VEG_COV_NEW, VEG_HEI_CUR, VEG_HEI_NEW, VEL_U_CUR, VEL_U_NEW, VEL_V_CUR,
 VEL_V_NEW, &
 VOPT, Z0_H_ECO, Z0_M_ECO, &
 ALB_SD, ALB_SN, ALB_SNT, CO2_CON_STEP, CUMTOT, &
 DENS_SN, EPSF, EPSG, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, &
 EVA_SN_F, EVA_SN_G, H_OBS_ALL, H_OBS_WIND, HEI_SN, HM_SN, HW_SN, &
 LAI_STEP, LAI_STEP0, LEA_WAT_CON, &
 P_HM_SN, P_HW_SN, P_LEA_WAT_CON, P_SRF_ICE_CON, P_SRF_WAT_CON, P_TEMP_SN, &
 SHF_SN_F, SHF_SN_G, SRF_ICE_CON, SRF_WAT_CON, &
 SUM_DRAI, SUM_EVAP, SUM_PREC, SUM_RUNO, SUM_SOI_HEA_STO, &
 TEMP_F_STEP, TEMP_SN, USTAR, &
 VEG_COV_STEP, VEG_COV_STEP0, VEG_HEI_STEP, VEG_HEI_STEP0, &
 Z0_H, Z0_M, Z0_SN, Z0_V, ZER_DIS_LEV, &
 ALB_TOT, BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT, &
 CAPPAMIXF, CAPPAMIXG, &
 CL_HIG_STEP, CL_LOW_STEP, CL_TOT_STEP, COND_F_DRY, COND_F_WET, COND_G_DRY, COND_G_WET, &
 CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F, &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, &
 DRAG_H, DRAG_M, DRAG_V, GRO_CAR_ASS_RAT, HAZE_IND, &
 LHF_A, LHF_F, LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, &
```

```
 NET_CAR_ASS_RAT, NR_A, NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, &
 P_TEMP_F_STEP, PREC_F, PREC_G, PRECSN, PREC_STEP, PRES_A_STEP, &
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, &
 RAD_L_D, RAD_L_F_D, RAD_L_F_U, RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D,
 RAD_L_G_TOT_U, &
 RAD_L_G_U, RAD_L_SN_F_D, RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, &
 RAD_S_D, RAD_S_F_D, RAD_S_F_U, RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U,
 &
 RAD_S_SN_G_D, RAD_S_SN_G_U, RAD_S_SN_U, RAD_S_U, &
 REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, &
 RESIST_AH, RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, &
 RESIST_LEAF, RESIST_SRF, RESIST_SRF_SN, RH_STEP, &
 SHF_A, SHF_F, SHF_G, SHF_SN, SO_SRAT_RO_DE, &
 SOL_ALFA, SUM_DRAI_TMP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, SUM_PREC_TMP, SUM_PREC_SN, SUM_RUNO_TMP
 , &
 TEMP_A_F, TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_ROOT, UPWP, &
 VEL_MOD_CUR, VEL_MOD_NEW, VEL_MOD_STEP, VEL_U_STEP, VEL_V_STEP, VPWP, &
 ILOGALB, ILOGFUN, L360DAYS_CALENDAR, LLONGWFLAG, LMEDLYN, LPHOTO, &
 LTCAN_NEW, VEGSHR, &
 LSNOW, LVAP, &
 LSNOW2, &
 NSO, &
 BSOIL, CAPPAETA_S, DEP_SOI, ETA_S, ETA_WI, FRP_OCC,&
 MID_LAY_DEP, MOIS_POT_SAT, ROC, SSR_FIE_CAPC, &
 ETA_I, ETA_W, P_ETA_I, P_ETA_W, P_TEM_SOI,&
 TEM_SOI,&
 CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, DIFFVT, &
 DIFFVTM, EVATRA, MOIS_POT, ROCIVOL, SOILFLUX, SR_SOI, THEDIFM)
!******************************************************************************
! The main routine of the program
!******************************************************************************
! Calls: DEWPOINT, SURFPROC
! - Authors: C. Cassardo, J.J. Ji (13-Jul-1994)
! - Last revision: C. Cassardo (5-Oct-2016)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: IMI, INP_SOIL, IORA, JU_DAY, LVEG, MITER, NUM_SOI_LAY, PTW

REAL,INTENT(IN) :: AIR_DENS, ALAT, ALB_FH, ALON, &
 C_DREN, CL_LOW_CUR, CL_LOW_NEW, CL_TOT_CUR, CL_TOT_NEW, CLEG, CO2_CON_CUR, CO2_CON_NEW, &
 D0_VEGPAR, DELT, DELTAZ, EPS, ERR_ECO, ESSE, &
 GAMMA, GLORAD_NEW, GLORAD_CUR, &
 HAJ, HAV, HDJ, HDV, JOPT, &
 LAI_CUR, LAI_NEW, LONRAD_CUR, LONRAD_NEW, MRS, &
 PREC_NEW, PREC_CUR, PRS_AIR_CUR, PRS_AIR_NEW, PVMAX25, &
 Q_AIR_CUR, Q_AIR_NEW, &
 RGL_NOILHAN, RESIST_F_MIN, RH_AIR_CUR, RH_AIR_NEW, &
 SSR_FIE_CAP, SSR_WI, &
 TEM_AIR_NEW, TEM_AIR_CUR, TIMESTEP_SEC, TOPTJ, TOPTV, TORTUOS, &
 VEG_COV_CUR, VEG_COV_NEW, VEG_HEI_CUR, VEG_HEI_NEW, VEL_U_CUR, VEL_U_NEW, VEL_V_CUR,
 VEL_V_NEW, &
 VOPT, Z0_H_ECO, Z0_M_ECO

REAL,INTENT(INOUT) :: ALB_SD, ALB_SN, ALB_SNT, CO2_CON_STEP, CUMTOT, &
 DENS_SN, EPSF, EPSG, EVA_A, EVA_F, EVA_F_D, EVA_F_W, EVA_G, EVA_G_D, EVA_G_W, EVA_SN, &
 EVA_SN_F, EVA_SN_G, H_OBS_ALL, H_OBS_WIND, HEI_SN, HM_SN, HW_SN, &
 LAI_STEP, LAI_STEP0, LEA_WAT_CON, &
 P_HM_SN, P_HW_SN, P_LEA_WAT_CON, P_SRF_ICE_CON, P_SRF_WAT_CON, P_TEMP_SN, &
 SHF_SN_F, SHF_SN_G, SRF_ICE_CON, SRF_WAT_CON, &
 SUM_DRAI, SUM_EVAP, SUM_PREC, SUM_RUNO, SUM_SOI_HEA_STO, &
 TEMP_F_STEP, TEMP_SN, USTAR, &
 VEG_COV_STEP, VEG_COV_STEP0, VEG_HEI_STEP, VEG_HEI_STEP0, &
 Z0_H, Z0_M, Z0_SN, Z0_V, ZER_DIS_LEV

REAL,INTENT(OUT) :: ALB_TOT, &
```

```
 BAL_FLU_F, BAL_FLU_G, BAL_FLU_SN, BAL_FLU_TOT, &
 CAPPAMIXF, CAPPAMIXG, &
 CL_HIG_STEP, CL_LOW_STEP, CL_TOT_STEP, COND_F_DRY, COND_F_WET, COND_G_DRY, COND_G_WET, &
 CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F, &
 CONF_SN_F, CONF_SN_G, CONF_SN_SRF, COVER_SN, COVER_SN_F, COVER_SN_G, &
 DRAG_H, DRAG_M, DRAG_V, &
 GRO_CAR_ASS_RAT, HAZE_IND, &
 LHF_A, LHF_F, LHF_F_D, LHF_F_W, LHF_G, LHF_G_D, LHF_G_W, LHF_SN, LHF_SN_F, LHF_SN_G, &
 NET_CAR_ASS_RAT, NR_A, NR_F, NR_G, NR_L_F_D, NR_L_F_U, NR_L_SN_D, NR_L_SN_U, NR_SN, &
 P_TEMP_F_STEP, PREC_F, PREC_G, PRECSN, PREC_STEP, PRES_A_STEP, &
 Q_A_F, Q_RAIN_F, Q_RAIN_G, Q_SP_A_STEP, QSF, QSG, QSG_RH, &
 RAD_L_D, RAD_L_F_D, RAD_L_F_U, RAD_L_FG_D, RAD_L_FG_U, RAD_L_G_D, RAD_L_G_TOT_D,
 RAD_L_G_TOT_U, &
 RAD_L_G_U, RAD_L_SN_F_D, RAD_L_SN_F_U, RAD_L_SN_G_D, RAD_L_SN_G_U, RAD_L_SN_U, RAD_L_U, &
 RAD_S_D, RAD_S_F_D, RAD_S_F_U, RAD_S_G_D, RAD_S_G_U, RAD_S_SN_D, RAD_S_SN_F_D, RAD_S_SN_F_U,
 &
 RAD_S_SN_G_D, RAD_S_SN_G_U, RAD_S_SN_U, RAD_S_U, &
 REL_LEA_WAT_CON, REL_SRF_ICE_CON, REL_SRF_WAT_CON, RES_RAT, &
 RESIST_AH, RESIST_AM, RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_PHOTO, &
 RESIST_LEAF, RESIST_SRF, RESIST_SRF_SN, RH_STEP, &
 SHF_A, SHF_F, SHF_G, SHF_SN, SO_SRAT_RO_DE, &
 SOL_ALFA, SUM_DRAI_TMP, SUM_EVAP_TMP, SUM_MIN_LW_TMP, SUM_PREC_TMP, SUM_PREC_SN, SUM_RUNO_TMP
 , &
 TEMP_A_F, TEMP_A_STEP, TEMP_DEW_A_STEP, TEMP_ROOT, UPWP, &
 VEL_MOD_CUR, VEL_MOD_NEW, VEL_MOD_STEP, VEL_U_STEP, VEL_V_STEP, VPWP

LOGICAL,INTENT(IN) :: ILOGALB, ILOGFUN, L360DAYS_CALENDAR, LLONGWFLAG, LMEDLYN, LPHOTO, &
 LTCAN_NEW, VEGSHR

LOGICAL,INTENT(INOUT) :: LSNOW, LVAP

LOGICAL,INTENT(OUT) :: LSNOW2

INTEGER,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: NSO

REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: BSOIL, CAPPAETA_S, DEP_SOI, ETA_S, ETA_WI, FRP_OCC,&
 MID_LAY_DEP, MOIS_POT_SAT, ROC, SSR_FIE_CAPC

REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_I, ETA_W, P_ETA_I, P_ETA_W, SR_SOI, P_TEM_SOI
,&
 TEM_SOI

REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: CAPPAETA, CAPPAETAM, DIFFSUM, DIFFSUMM, DIFFVT, &
 DIFFVTM, EVATRA, MOIS_POT, ROCIVOL, SOILFLUX, THEDIFM

 ! dummy variables
INTEGER :: I, IMINUTI, J, JJ

REAL :: DEWPOINT, EPSF1, LEA_WAT_CON_MAX, RADINF, RADSUP, SRF_WAT_CON_MAX, V_ROOT_TOT,
VEG_HEA_CAP,&
 Z01, Z02

REAL,DIMENSION(NUM_SOI_LAY) :: RP_LAYER, RP_TOTAL

! initializations for cumulative variables
SUM_RUNO_TMP = 0.
SUM_DRAI_TMP = 0.
SUM_PREC_TMP   = 0.
SUM_EVAP_TMP   = 0.
SUM_MIN_LW_TMP  = 0.
SUM_PREC_SN = 0.

! main loop for calculation of variables
DO J=1,MITER

! root distribution: the volume occuped by root is assimilated to a cone of
! height ROOT_DEP and of basis ray ROOT_DEP (the basis area is not important because
```

```
! the ratio between layers is used). The percentage of soil layers occuped by
! roots is also calculated as RP_TOTAL(i) - all volumes are in PI/3 units since
! the volume of a cone is VOL = PI r^2 h /3: being r=h, VOL = PI/3 r^3

 V_ROOT_TOT=DELTAZ**3 ! total volume used by roots [m3]
 DO I=1,NUM_SOI_LAY
  RADINF=0.
  RADSUP=0.
  DO JJ=1,I
   IF (JJ /= I) RADSUP=RADSUP+DEP_SOI(JJ)
   RADINF=RADINF+DEP_SOI(JJ)
  END DO
  RADINF=MAX(0., (DELTAZ-RADINF) ) ! ray/height of largest cone
  RADSUP=MAX(0., (DELTAZ-RADSUP) ) ! ray/height of smallest cone

! volume fraction occupied by roots in i-th layer vs total root volume
  RP_LAYER(I)=(RADSUP**3-RADINF**3)/V_ROOT_TOT

! percentage of roots volume in i-th layer
  RP_TOTAL(I)=ROOT_FRAC_COST*VEG_COV_STEP*(RADSUP**3-RADINF**3)/&
   (3.*DELTAZ*DELTAZ*DEP_SOI(I))
 END DO

! roots percentage is used for EVATRA sharing between layers
! vegetation cannot evapotranspirate if a specific layer is below wilting point
 RADSUP = 0.
 DO I=1,NUM_SOI_LAY
  IF (ETA_W(I) < ETA_WI(I)) RP_LAYER(I) = 0.
  RADSUP = RADSUP + RP_LAYER(I)
 ENDDO

! renormalization to RADSUP (<=1) if /= 0
! RADSUP is sum of percents: in normal conditions RADSUP=1.; if <1 it means that
! some layer is below wilting point; if =0 all layers are below wilting point
 DO I=1,NUM_SOI_LAY
  IF (RADSUP > EPSIL) THEN
   RP_LAYER(I) = RP_LAYER(I) / RADSUP
  ELSE
   RP_LAYER(I) = 0. ! all layers below wilting point: no evapotranspiration
  ENDIF
 ENDDO

! assign to current variable output value from the previous time step simulation
 DO I=1,NUM_SOI_LAY
  P_TEM_SOI(I)=TEM_SOI(I)      ! update Temp and Humidity in the soil layers
  P_ETA_W(I)=ETA_W(I)
 ENDDO

 P_LEA_WAT_CON = LEA_WAT_CON
 P_SRF_WAT_CON = SRF_WAT_CON
 P_SRF_ICE_CON = SRF_ICE_CON

 IF (LSNOW) THEN
  P_HM_SN=HM_SN
  P_HW_SN=HW_SN
  P_TEMP_SN=TEMP_SN
 ELSE
  P_TEMP_SN=ERR_VE
  P_HM_SN=0.
  P_HW_SN=0.
 ENDIF

! root zone averaged quantities
 SO_SRAT_RO_DE=0.
 TEMP_ROOT=0.
 DO I=1,NUM_SOI_LAY
! mean soil temperature and moisture in root zone considers as weight the relative
```

```
! weigth of root in each layer
!??? non mi è chiara la differenza con RP_TOTAL - rivedere
  SO_SRAT_RO_DE=SO_SRAT_RO_DE+FRP_OCC(I)*SR_SOI(I)
  TEMP_ROOT=TEMP_ROOT+FRP_OCC(I)*P_TEM_SOI(I)
 ENDDO

 ! initializations
 ! maximum water on leafs
 LEA_WAT_CON_MAX=0.0002*VEG_COV_STEP*LAI_STEP ! (G94 pag. 237)

 ! maximum water on bare soil
 IF(NSO(1)==15 .OR. NSO(1)==16)THEN
  SRF_WAT_CON_MAX=0.00048 ! Grimmond and Oke 1991: urban pavement
 ELSE
  !Bare soil
  SRF_WAT_CON_MAX=0.00020 ! any REF?
 END IF

! soil emissivity
 IF(EPSG==ERR_VE) EPSG=0.90+MIN(1.,0.18*ETA_S(1)*SR_SOI(1)) ! Braud et al. (1995)
!???impostare valida sempre ed eliminarla dal .par

! vegetation emissivity correction using Ecoclimap
 IF (EPS /= ERR_ECO) THEN

! ecoclimap emissivity is the total emissivity, including vegetation and base
! soil - to derive vegetation emissivity, we invert the formula
!    EPS = VEG_COV_STEP * EPSF + (1.-VEG_COV_STEP) * EPSG using for EPSG the above
! calculated value being VEG_COV_STEP at denominator, the inversion is calculated only
! when VEG_COV_STEP>EPSIL - viceversa we put EPSF=EPSG as the EPSF value is useless
  IF (VEG_COV_STEP > EPSIL) THEN
   EPSF1 = ( EPS - (1. - VEG_COV_STEP) * EPSG ) / VEG_COV_STEP
  ELSE
   EPSF1 = EPSG
  ENDIF

! check that EPSF1 is in the range 0.9-1.0
  IF (EPSF1 < 0.9 .OR. EPSF1 > 1.0) EPSF1 = EPSF
  EPSF = EPSF1
 ENDIF

! VEG_HEA_CAP is in J/K; it is equivalent to water heat capacity(=4186000J/m3K) for a
! vertical depth of max 1 mm; in G94 (pag 237) is considered equivalent to
! the heat capacity of 0.1-1 mm of water; mean value of 0.1-1 mm is 0.55 mm;
! then we choose   VEG_HEA_CAP=4186000*(0.55E-3*LAI_STEP)= 2302.3*LAI_STEP (*1 m2) = J/K;
! the adding factor VEG_COV_STEP accounts for the canopy presence.
! The final term accounts for the water due to dew

! VEG_HEA_CAP=2302.3*LAI_STEP*VEG_COV_STEP + 4.186E6*P_LEA_WAT_CON ! J/K
!claudio eliminated the dependency from VEG_COV_STEP
 VEG_HEA_CAP=2302.3*LAI_STEP + 4.186E6*P_LEA_WAT_CON ! J/K

! linear interpolation in time of observed data during the TIMESTEP_SEC time step
! * primary data (directly observed)
 TEMP_A_STEP    = TEM_AIR_CUR + REAL(J) * (TEM_AIR_NEW-TEM_AIR_CUR) / REAL(MITER)
 PRES_A_STEP    = PRS_AIR_CUR + REAL(J) * (PRS_AIR_NEW-PRS_AIR_CUR) / REAL(MITER)
 Q_SP_A_STEP    = Q_AIR_CUR    + REAL(J) * (Q_AIR_NEW-Q_AIR_CUR)     / REAL(MITER)
 CL_TOT_STEP    = CL_TOT_CUR  + REAL(J) * (CL_TOT_NEW-CL_TOT_CUR)   / REAL(MITER)
 VEL_U_STEP     = VEL_U_CUR   + REAL(J) * (VEL_U_NEW-VEL_U_CUR)     / REAL(MITER)
 VEL_V_STEP     = VEL_V_CUR   + REAL(J) * (VEL_V_NEW-VEL_V_CUR)     / REAL(MITER)
 VEL_MOD_CUR    = SQRT(VEL_U_CUR*VEL_U_CUR+VEL_V_CUR*VEL_V_CUR)
 VEL_MOD_NEW    = SQRT(VEL_U_NEW*VEL_U_NEW+VEL_V_NEW*VEL_V_NEW)
 VEL_MOD_STEP   = VEL_MOD_CUR + REAL(J) * (VEL_MOD_NEW-VEL_MOD_CUR) / REAL(MITER)
 PREC_STEP      = PREC_CUR    + REAL(J) * (PREC_NEW-PREC_CUR)       / REAL(MITER)
 CL_LOW_STEP    = CL_LOW_CUR  + REAL(J) * (CL_LOW_NEW-CL_LOW_CUR)   / REAL(MITER)
 RAD_S_D    = GLORAD_CUR  + REAL(J) * (GLORAD_NEW-GLORAD_CUR)   / REAL(MITER)
 RH_STEP        = RH_AIR_CUR  + REAL(J) * (RH_AIR_NEW-RH_AIR_CUR)   / REAL(MITER)
```

```
 IF(LLONGWFLAG)THEN
  RAD_L_D   = LONRAD_CUR  + REAL(J) * (LONRAD_NEW-LONRAD_CUR)   / REAL(MITER)
 END IF
 IF(VEG_HEI_NEW/=ERR_VE)THEN
  VEG_HEI_STEP0 = VEG_HEI_CUR + REAL(J) * (VEG_HEI_NEW-VEG_HEI_CUR) / REAL(MITER)
 END IF
 IF(LAI_NEW/=ERR_VE)THEN
  LAI_STEP0     = LAI_CUR     + REAL(J) * (LAI_NEW-LAI_CUR)         / REAL(MITER)
 END IF
 IF(VEG_COV_NEW/=ERR_VE)THEN
  VEG_COV_STEP0 = VEG_COV_CUR + REAL(J) * (VEG_COV_NEW-VEG_COV_CUR) / REAL(MITER)
 END IF
 IF(CO2_CON_NEW/=ERR_VE)THEN
  CO2_CON_STEP  = CO2_CON_CUR + REAL(J) * (CO2_CON_NEW-CO2_CON_CUR) / REAL(MITER)
 END IF

! change of vegetation temperature (it acts only for the first step after which
! VEG_COV_STEP has been equal to zero)
! the factor 0.01 to avoid zero flux
 IF (TEMP_F_STEP == ERR_VE) TEMP_F_STEP = TEMP_A_STEP+0.01
 P_TEMP_F_STEP = TEMP_F_STEP

! correction to take into account different T and wind measurement quote
! these instructions will report wind speed from H_OBS_WIND to H_OBS_ALL reference level
!??? occhio che però in molti casi poi si tiene conto di H_OBS_WIND... mi pare!!!
 IF (H_OBS_WIND/=H_OBS_ALL) THEN
  Z01=(H_OBS_WIND-ZER_DIS_LEV)/Z0_M
  Z02=(H_OBS_ALL-ZER_DIS_LEV)/Z0_M
  VEL_U_STEP=VEL_U_STEP*ALOG(Z02)/ALOG(Z01)
  VEL_V_STEP=VEL_V_STEP*ALOG(Z02)/ALOG(Z01)
  VEL_MOD_STEP=VEL_MOD_STEP*ALOG(Z02)/ALOG(Z01)
 ENDIF

! secondary data
 VEL_MOD_STEP = MAX(0.5, VEL_MOD_STEP)  ! minimum wind thresold
 CL_HIG_STEP = CL_TOT_STEP-CL_LOW_STEP ! High level cloudiness
 TEMP_DEW_A_STEP = DEWPOINT(Q_SP_A_STEP,PRES_A_STEP) ! dew point [K]
 IMINUTI = IMI+INT(REAL(J)*DELT/REAL(MITER)) ! current minutes

! check on vegetation input (to avoid computational problems)
 IF (LAI_STEP0 == 0. .AND. VEG_COV_STEP0 /= 0.) LAI_STEP0 = 0.1
 IF (LAI_STEP0 /= 0. .AND. VEG_COV_STEP0 == 0.) VEG_COV_STEP0 = 0.1
 IF (LAI_STEP == 0. .AND. VEG_COV_STEP /= 0.) LAI_STEP = 0.1
 IF (LAI_STEP /= 0. .AND. VEG_COV_STEP == 0.) VEG_COV_STEP = 0.1

! run core of the model
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
 CALL SURFPROC(ALB_FH,ALAT,ALB_SD,ALB_SN,ALB_SNT,ALB_TOT,ALON,PREC_STEP,PRES_A_STEP,&
  Q_SP_A_STEP,ESSE,TEMP_A_STEP,VEL_U_STEP,VEL_V_STEP,VEL_MOD_STEP,GAMMA,C_DREN,CAPPAETA,
  CAPPAETAM,&
  CAPPAMIXF,CAPPAMIXG,DRAG_H,DRAG_M,DRAG_V,VEG_HEA_CAP,CLEG,CL_TOT_STEP,CO2_CON_STEP,D0_VEGPAR
  ,&
  AIR_DENS,DENS_SN,DIFFSUM,DIFFSUMM,DIFFVT,DIFFVTM,ZER_DIS_LEV,&
  TIMESTEP_SEC,EVA_G,EVA_G_W,EVA_F_W,LEA_WAT_CON,SRF_WAT_CON,SRF_ICE_CON,&
  LEA_WAT_CON_MAX,SRF_WAT_CON_MAX,EPSF,EPSG,EVATRA,LHF_A,LHF_F,LHF_G,MOIS_POT,&
  LHF_SN,LHF_F_D,VEG_HEI_STEP,SHF_A,SHF_F,SHF_G,SHF_SN,SHF_SN_F,SHF_SN_G,HAZE_IND,HM_SN,HEI_SN
  ,HW_SN,&
  ILOGALB,ILOGFUN,LLONGWFLAG,LVAP,LVEG,NUM_SOI_LAY,LSNOW,LSNOW2,LTCAN_NEW,IMINUTI,&
  INP_SOIL,IORA,L360DAYS_CALENDAR,LAI_STEP,JU_DAY,PRECSN,SUM_PREC_SN,P_HM_SN,&
  P_HW_SN,P_LEA_WAT_CON,P_SRF_WAT_CON,P_SRF_ICE_CON,PREC_G,P_TEMP_F_STEP,P_TEMP_SN,SSR_FIE_CAP
  ,&
  SO_SRAT_RO_DE,SSR_WI,Q_A_F,BAL_FLU_F,BAL_FLU_G,QSF,QSG,CONF_SN_SRF,RESIST_AH,RESIST_AM,&
  RESIST_AV,RESIST_B,RESIST_D,RESIST_F,RESIST_LEAF,RGL_NOILHAN,RH_STEP,RESIST_F_MIN,&
  ROCIVOL,RP_TOTAL,REL_LEA_WAT_CON,REL_SRF_WAT_CON,REL_SRF_ICE_CON,RAD_L_D,NR_L_F_D,NR_L_F_U,
  RAD_L_G_TOT_D,&
  RAD_L_G_TOT_U,RAD_L_U,RAD_L_FG_D,RAD_L_FG_U,NR_A,NR_F,NR_G,NR_SN,RAD_S_D,RAD_S_F_D,RAD_S_F_U
  ,RAD_S_G_D,&
```

```
  RAD_S_G_U,RAD_S_U,SOL_ALFA,COVER_SN,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,SUM_DRAI,SUM_RUNO,&
  SUM_PREC,SUM_EVAP,SUM_SOI_HEA_STO,SUM_DRAI_TMP,SUM_RUNO_TMP,SUM_MIN_LW_TMP,SUM_PREC_TMP,&
  SUM_EVAP_TMP,TORTUOS,TEMP_A_F,TEMP_F_STEP,TEMP_ROOT,TEMP_SN,H_OBS_ALL,Z0_M,Z0_H,Z0_V,BSOIL,&
  CAPPAETA_S,ETA_S,THEDIFM,RP_LAYER,DEP_SOI,MOIS_POT_SAT,SR_SOI,P_TEM_SOI,TEM_SOI,ROC,&
  MID_LAY_DEP,USTAR,Z0_SN,ETA_W,P_ETA_W,ETA_I,SSR_FIE_CAPC,ERR_ECO,Z0_M_ECO,Z0_H_ECO,P_ETA_I,&
  H_OBS_WIND,NSO,GRO_CAR_ASS_RAT,NET_CAR_ASS_RAT,HAJ,HAV,HDJ,HDV,JOPT,LMEDLYN,MRS,PTW,
  RESIST_F_PHOTO,RES_RAT,TOPTJ,&
  TOPTV,VOPT,PVMAX25,LPHOTO,BAL_FLU_SN,BAL_FLU_TOT,&
  RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_SN_G_D,RAD_S_SN_G_U,RAD_S_SN_D,RAD_S_SN_U,RAD_L_F_D,
  RAD_L_F_U,RAD_L_G_D,RAD_L_G_U,&
  RAD_L_SN_F_U,RAD_L_SN_G_D,RAD_L_SN_G_U,RAD_L_SN_U,NR_L_SN_D,NR_L_SN_U,LHF_F_W,LHF_G_D,
  LHF_G_W,LHF_SN_F,LHF_SN_G,&
  EVA_A,EVA_F,EVA_F_D,EVA_G_D,EVA_SN,EVA_SN_F,EVA_SN_G,UPWP,VPWP,CONF_SN_F,CONF_SN_G,Q_RAIN_F,
  Q_RAIN_G,RESIST_D_SN,RESIST_SRF,RESIST_SRF_SN,&
  CONDUCT_AH,CONDUCT_AM,CONDUCT_AV,CONDUCT_B,CONDUCT_D,CONDUCT_D_SN,COND_F_DRY,CONDUCT_F,
  COND_F_WET,COND_G_DRY,&
  COND_G_WET,PREC_F,QSG_RH,RAD_L_SN_F_D)

! evaluation of soil underground fluxes at the interfaces
 DO I=1,NUM_SOI_LAY-1
  SOILFLUX(I)=-2.0*THEDIFM(I)*(TEM_SOI(I+1)-TEM_SOI(I))/(DEP_SOI(I)+DEP_SOI(I+1))
 ENDDO

ENDDO !(loop on j)

! modification of some vegetation parameters considering the vegetation shrink
! during winter season. NB: only vegetation for which VEGSHR is true will be modified
 CALL MANAGEVEG(CUMTOT,DTK,MITER*TIMESTEP_SEC,HEI_SN,VEG_HEI_STEP,VEG_HEI_STEP0,LAI_STEP,
 LAI_STEP0,&
  VEG_COV_STEP,VEG_COV_STEP0,TEMP_A_STEP,P_TEMP_F_STEP,TEMP_ROOT,VEGSHR)

RETURN
END SUBROUTINE UTOPIA_DRIVER

SUBROUTINE MEMORYIN_CONFIG(DIRPA,IC_IN,I_INT,I_SLO,NDATI,INDERR)
!------------------------------------------------------------------------------
! Reads the configuration file for input data in memoryin.config, stores in
! memory the required input variables.
!------------------------------------------------------------------------------
! Calls: none
! - Authors: C. Cassardo (23-Aug-2015)
! - Last revision: C. Cassardo (23-Aug-2015)
!------------------------------------------------------------------------------
USE CONSTANTS
IMPLICIT NONE
INTEGER,DIMENSION(MDAT_IN),INTENT(OUT) :: IC_IN
INTEGER :: I
INTEGER,INTENT(INOUT) :: NDATI
INTEGER,INTENT(OUT) :: INDERR

REAL,DIMENSION(MDAT_IN),INTENT(OUT) :: I_INT,I_SLO
CHARACTER(len=*),INTENT(IN) :: DIRPA
CHARACTER(len=18) :: VARIABLE_NAME


! Opens the configuration file to be read from DIRPA default directory
OPEN(UNIT=1,FILE=TRIM(DIRPA)//'memoryin.config',STATUS='OLD',ACTION='READ',&
 IOSTAT=INDERR)
IF(INDERR/=0)THEN
 PRINT *,'ERROR : cannot open the file'
 PRINT *,TRIM(DIRPA)//'memoryin.config'
 PRINT *,'for the configuration of input data.'
 RETURN
END IF

! Reads the first 5 heading lines
READ(1,*)
```

```
READ(1,*)
READ(1,*)
READ(1,*)
READ(1,*)

DO I=1,MDAT_IN
 READ(1,'(A18,I2,2F12.2)',IOSTAT=INDERR)VARIABLE_NAME,IC_IN(I),I_INT(I),I_SLO(I)
 IF(INDERR/=0)THEN
  PRINT *,'Unexpected ERROR at line ',I+1,'. Check the file'
  PRINT *,TRIM(DIRPA)//'memoryin.config'
  CLOSE(UNIT=1)
  RETURN
 END IF
ENDDO

!evaluate how many input data are present in input file
NDATI=0
DO I=1,MDAT_IN
 !IF (IC_IN(I)>0) NDATI=NDATI+1
 IF (IC_IN(I)>0 .AND. IC_IN(I)>NDATI) NDATI=IC_IN(I)
ENDDO
PRINT *,'Founded',NDATI,' input data'

RETURN
END

!************************************************************************************
SUBROUTINE MEMORYOUT_CONFIG(DIRPA,FORMAT_OUTPUT,LAYERS_OUT_SR_SOI,&
 LAYERS_OUT_TEM_SOI,LAYERS_OUT_CAPPAETA,LAYERS_OUT_CAPPAETAM,&
 LAYERS_OUT_THEDIFM,LAYERS_OUT_FRP_OCC,LAYERS_OUT_DIFFSUMM,&
 LAYERS_OUT_DIFFSUM,LAYERS_OUT_DIFFVT,LAYERS_OUT_DIFFVTM,&
 LAYERS_OUT_EVATRA,LAYERS_OUT_MOIS_POT,LAYERS_OUT_SOILFLUX,LAYERS_OUT_ROC,LAYERS_OUT_ROCIVOL,&
 LAYERS_OUT_ETA_W,LAYERS_OUT_ETA_I,OUTPUT_VARIABLES,NUM_SOI_LAY,&
 VARIABLE_NAME,INDERR)
!************************************************************************************
! Reads the configuration file for routine dynamic_memoryout, stores in memory the
! informations about required output variables and output format.
!************************************************************************************
! Calls: none
! - Authors: M. Galli, R. Bonanno (23-Apr-2009)
! - Last revision: C. Cassardo (15-May-2014)
!************************************************************************************
USE CONSTANTS
IMPLICIT NONE

CHARACTER(len=1) :: CHAR_SEPARATOR
CHARACTER(len=1) :: CHAR_SEPARATOR_OLD
CHARACTER(len=*),INTENT(IN) :: DIRPA
LOGICAL :: FLAG_EXIT
CHARACTER(len=*),INTENT(OUT) :: FORMAT_OUTPUT
CHARACTER(len=100) :: FORMAT_OUTPUT_SINGLEVAR
INTEGER :: I
INTEGER :: ILAYER
INTEGER :: IND_MAX
INTEGER :: IND_MIN
INTEGER :: J
INTEGER :: K
INTEGER,INTENT(OUT) :: INDERR
LOGICAL,DIMENSION(NUM_SOI_LAY) :: LAYERS_OUT_GENERIC
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_SR_SOI
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_TEM_SOI
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_CAPPAETA
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_CAPPAETAM
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_THEDIFM
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_FRP_OCC
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_DIFFSUM
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_DIFFSUMM
```

```
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_DIFFVT
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_DIFFVTM
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_EVATRA
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_MOIS_POT
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_SOILFLUX
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_ROC
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_ROCIVOL
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_ETA_W
LOGICAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: LAYERS_OUT_ETA_I
INTEGER :: LAY_MAX
INTEGER :: LAY_MIN
INTEGER,INTENT(IN) :: NUM_SOI_LAY
INTEGER :: NUM_OUTPUT_VECTOR
LOGICAL,DIMENSION(NSUDYN),INTENT(OUT) :: OUTPUT_VARIABLES
INTEGER :: POSITION_COMMA
INTEGER :: POSITION_HYPHEN
CHARACTER(len=100) :: SELECT_OUTPUT
CHARACTER(len=100),DIMENSION(NSUDYN),INTENT(OUT) :: VARIABLE_NAME

! Opens the configuration file to be read from DIRPA default directory
OPEN(UNIT=1,FILE=TRIM(DIRPA)//'memoryout.config',STATUS='OLD',ACTION='READ',IOSTAT=INDERR)
IF(INDERR/=0)THEN
 PRINT *,'ERROR : cannot open the file'
 PRINT *,TRIM(DIRPA)//'memoryout.config'
 PRINT *,'for the configuration of subroutine DYNAMIC_MEMORYOUT.'
 RETURN
END IF

! Reads the first heading line
READ(1,*)

! Initialization of NSU6
NSU6=0

DO I=1,NSUDYN
 READ(1,*,IOSTAT=INDERR)VARIABLE_NAME(I),SELECT_OUTPUT,FORMAT_OUTPUT_SINGLEVAR
 IF(INDERR/=0)THEN
  PRINT *,'Unexpected ERROR at line ',I+1,'. Check the file'
  PRINT *,TRIM(DIRPA)//'memoryout.config'
  CLOSE(UNIT=1)
  RETURN
 END IF

!If here, UTOPIA variable is an array
 IF (VARIABLE_NAME(I)=='SR_SOI'.OR.VARIABLE_NAME(I)=='TEM_SOI'.OR.&
  VARIABLE_NAME(I)=='CAPPAETA'.OR.VARIABLE_NAME(I)=='CAPPAETAM'.OR.&
  VARIABLE_NAME(I)=='THEDIFM'.OR.VARIABLE_NAME(I)=='DIFFSUMM'.OR.&
  VARIABLE_NAME(I)=='DIFFSUM'.OR.VARIABLE_NAME(I)=='DIFFVTM'.OR.&
  VARIABLE_NAME(I)=='DIFFVT'.OR.VARIABLE_NAME(I)=='EVATRA'.OR.&
  VARIABLE_NAME(I)=='MOIS_POT'.OR.VARIABLE_NAME(I)=='SOILFLUX'.OR.&
  VARIABLE_NAME(I)=='FRP_OCC' .OR.VARIABLE_NAME(I)=='ROC' .OR.&
  VARIABLE_NAME(I)=='ROCIVOL'.OR.VARIABLE_NAME(I)=='ETA_W' .OR.&
  VARIABLE_NAME(I)=='ETA_I' )THEN

  IF(SELECT_OUTPUT=='Y'.OR.SELECT_OUTPUT=='y'.OR.&
   SELECT_OUTPUT=='T'.OR.SELECT_OUTPUT=='t')THEN
   OUTPUT_VARIABLES(I)=.TRUE.
   LAYERS_OUT_GENERIC=.TRUE.
  ELSE IF(SELECT_OUTPUT=='N'.OR.SELECT_OUTPUT=='n'.OR.&
   SELECT_OUTPUT=='F'.OR.SELECT_OUTPUT=='f')THEN
   OUTPUT_VARIABLES(I)=.FALSE.
   LAYERS_OUT_GENERIC=.FALSE.
  ELSE

! Initialization of the vector that describes the required layers in output
   LAYERS_OUT_GENERIC=.FALSE.
! Initialization of the pointers for an interval of layers
```

```
   LAY_MIN=0
   LAY_MAX=0
   CHAR_SEPARATOR=''
   CHAR_SEPARATOR_OLD=''

! Parsing the characters in the SELECT_OUTPUT string
   DO J=1,LEN_TRIM(SELECT_OUTPUT)

!???a cosa serve questo IF?
    IF(J==1)THEN
     IND_MIN=1
    ELSE
!IND_MIN=IND_MAX+2
     IND_MIN=1
    END IF

    POSITION_COMMA=INDEX(SELECT_OUTPUT(IND_MIN:),'&')
    POSITION_HYPHEN=INDEX(SELECT_OUTPUT(IND_MIN:),'-')

    IF(POSITION_COMMA>0.AND.POSITION_HYPHEN>0)THEN
     IND_MAX=MIN(POSITION_COMMA,POSITION_HYPHEN)-1
    ELSE IF(POSITION_COMMA==0.AND.POSITION_HYPHEN>0)THEN
     IND_MAX=POSITION_HYPHEN-1
    ELSE IF(POSITION_COMMA>0.AND.POSITION_HYPHEN==0)THEN
     IND_MAX=POSITION_COMMA-1
    ELSE
     IND_MAX=-1
    END IF

    IF(IND_MAX==-1)THEN
     READ(SELECT_OUTPUT(IND_MIN:),*,IOSTAT=INDERR)ILAYER
     FLAG_EXIT=.TRUE.
    ELSE
     READ(SELECT_OUTPUT(IND_MIN:IND_MAX),*,IOSTAT=INDERR)ILAYER
     FLAG_EXIT=.FALSE.
     CHAR_SEPARATOR=SELECT_OUTPUT(IND_MAX+1:IND_MAX+1)
    END IF

    IF(INDERR/=0)THEN
     PRINT *,'Unexpected character in layers reading at line ',I
     RETURN
    ELSE IF(ILAYER>NUM_SOI_LAY)THEN
     PRINT *,'ERROR : invalid requirement for output.'
     PRINT *,'Too high number of required layers: ',ILAYER,&
      ' > NUM_SOI_LAY = ',NUM_SOI_LAY
     PRINT *,'variable = ',VARIABLE_NAME(I),'   line = ',I
     INDERR=-17
     CLOSE(UNIT=1)
     RETURN
    END IF

! Determination of pointers to minimum and maximum soil layers
!selected for output in an interval
    IF(.NOT.FLAG_EXIT)THEN
     IF(SELECT_OUTPUT(IND_MAX+1:IND_MAX+1)=='-')THEN
      LAY_MIN=ILAYER
     ELSE IF(CHAR_SEPARATOR_OLD=='-')THEN
      LAY_MAX=ILAYER
     END IF
    ELSE
     IF(CHAR_SEPARATOR_OLD=='-')THEN
      LAY_MAX=ILAYER
     END IF
    END IF

! Setting to true all the layers within the interval, once it is set.
! Reinitialization of the pointers after the layers interval is finished
```

```
    IF(LAY_MAX==ILAYER)THEN
     DO K=LAY_MIN,LAY_MAX
      LAYERS_OUT_GENERIC(K)=.TRUE.
     END DO
     LAY_MIN=0
     LAY_MAX=0
    END IF

! Setting to true the elements of LAYERS_OUT_GENERIC given near the commas
! Maybe this can be a little redundant, but it should work
    IF(IND_MIN>1)THEN
     IF(SELECT_OUTPUT(IND_MAX+1:IND_MAX+1)=='&'.OR.&
      SELECT_OUTPUT(IND_MIN-1:IND_MIN-1)=='&'.OR.FLAG_EXIT)THEN
      LAYERS_OUT_GENERIC(ILAYER)=.TRUE.
     END IF
    ELSE IF(IND_MIN==1)THEN
     IF(FLAG_EXIT)THEN
      LAYERS_OUT_GENERIC(ILAYER)=.TRUE.
     END IF
     IF(IND_MAX>0)THEN
      IF(SELECT_OUTPUT(IND_MAX+1:IND_MAX+1)=='&')THEN
       LAYERS_OUT_GENERIC(ILAYER)=.TRUE.
      END IF
     END IF
    END IF

    IF(FLAG_EXIT)THEN
     EXIT
    ELSE
!Shift the reading string
     SELECT_OUTPUT=SELECT_OUTPUT(IND_MAX+2:)
     CHAR_SEPARATOR_OLD=CHAR_SEPARATOR
    END IF

   END DO

  END IF

! Count the number of elements required in output in the vector
  NUM_OUTPUT_VECTOR=0
  DO K=1,NUM_SOI_LAY
   IF(LAYERS_OUT_GENERIC(K))THEN
    NUM_OUTPUT_VECTOR=NUM_OUTPUT_VECTOR+1
   END IF
  END DO

! Compute the number of elements of vector SU6 of subroutine memoryout
  NSU6=NSU6+NUM_OUTPUT_VECTOR

! Assignement of the LAYERS_OUT_GENERIC to the correct UTOPIA field
  IF(VARIABLE_NAME(I)=='SR_SOI')THEN
   LAYERS_OUT_SR_SOI=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='TEM_SOI')THEN
   LAYERS_OUT_TEM_SOI=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='CAPPAETA')THEN
   LAYERS_OUT_CAPPAETA=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='CAPPAETAM')THEN
   LAYERS_OUT_CAPPAETAM=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='THEDIFM')THEN
   LAYERS_OUT_THEDIFM=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='FRP_OCC')THEN
   LAYERS_OUT_FRP_OCC=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='DIFFSUMM')THEN
   LAYERS_OUT_DIFFSUMM=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='DIFFSUM')THEN
   LAYERS_OUT_DIFFSUM=LAYERS_OUT_GENERIC
  ELSE IF(VARIABLE_NAME(I)=='DIFFVT')THEN
```

```
    LAYERS_OUT_DIFFVT=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='DIFFVTM')THEN
    LAYERS_OUT_DIFFVTM=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='EVATRA')THEN
    LAYERS_OUT_EVATRA=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='MOIS_POT')THEN
    LAYERS_OUT_MOIS_POT=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='SOILFLUX')THEN
    LAYERS_OUT_SOILFLUX=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='ROC')THEN
    LAYERS_OUT_ROC=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='ROCIVOL')THEN
    LAYERS_OUT_ROCIVOL=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='ETA_W')THEN
    LAYERS_OUT_ETA_W=LAYERS_OUT_GENERIC
   ELSE IF(VARIABLE_NAME(I)=='ETA_I')THEN
    LAYERS_OUT_ETA_I=LAYERS_OUT_GENERIC
   END IF

   IF(NUM_OUTPUT_VECTOR>0)THEN
    OUTPUT_VARIABLES(I)=.TRUE.
   END IF

  ELSE !UTOPIA variable is a scalar
   IF(SELECT_OUTPUT=='Y'.OR.SELECT_OUTPUT=='y'.OR.&
    SELECT_OUTPUT=='T'.OR.SELECT_OUTPUT=='t')THEN
    OUTPUT_VARIABLES(I)=.TRUE.
! Compute the number of elements of vector SU6 of subroutine memoryout
    NSU6=NSU6+1
   ELSE IF(SELECT_OUTPUT=='N'.OR.SELECT_OUTPUT=='n'.OR.&
    SELECT_OUTPUT=='F'.OR.SELECT_OUTPUT=='f')THEN
    OUTPUT_VARIABLES(I)=.FALSE.
   ELSE
    INDERR=17
    PRINT *,'ERROR : invalid output requirement at line ',I+1,' of '
    PRINT *,TRIM(DIRPA)//'memoryout.config'
    PRINT *,'VARIABLE NAME = ',TRIM(VARIABLE_NAME(I))
    CLOSE(UNIT=1)
    RETURN
   END IF

! Useful fariable for then managing the output format
   NUM_OUTPUT_VECTOR=1

  END IF

! Managing the format
  IF(I==1)THEN
   WRITE(FORMAT_OUTPUT(1:),'(A)')'(I4.4,1x,4(I2.2,1x),'    !'(1x,I4.4,4I2.2,'
  END IF

  IF(OUTPUT_VARIABLES(I))THEN
   WRITE(FORMAT_OUTPUT(LEN_TRIM(FORMAT_OUTPUT)+1:),'(I2.2,3A)',IOSTAT=INDERR)&
    NUM_OUTPUT_VECTOR,'(',TRIM(FORMAT_OUTPUT_SINGLEVAR),',x),'
   IF(INDERR/=0)THEN
    PRINT *,'WARNING : Too many output required in output, or too short character'
    PRINT *,'          variable to store the format for the required output.'
    PRINT *,'          Please increase the length of the variable FORMAT_OUTPUT,'
    PRINT *,'          and recompile the program, or select a minor number of output'
    PRINT *,'          variables.'
    INDERR=13
    CLOSE(1)
    RETURN
   END IF
  END IF

  IF(I==NSUDYN)THEN
```

```
  FORMAT_OUTPUT(LEN_TRIM(FORMAT_OUTPUT):LEN_TRIM(FORMAT_OUTPUT)+9)=')'
 END IF

END DO

! Closes the configuration file
CLOSE(UNIT=1)

END SUBROUTINE MEMORYOUT_CONFIG

!***************************************************************************
SUBROUTINE PALMER(PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,AIR_DENS,DELT,DEP_SOI,DIROU,EVAPOTRA,
LHF_A,&
 SHF_G,IANNEW,IGINEW,IMENEW,IMINEW,INN1,INNO,IORANEW,NUM_SOI_LAY,PORO,&
 POT_EVAPO_MM,POT_LOSS,POT_RECHARGE,POT_RUNOFF,SR_SOI,PREF,NR_A,RESIST_SRF,RESIST_D,&
 RECHARGE,RH_STEP,RLOSS,SUM_PREC_TMP,SUM_DRAI_TMP)
!***************************************************************************
! Calculates the Palmer drought severity index using the potential
! evapotranspiration evaluated following the Penman-Monteith equation
! DELTA_PDSI = slope of the saturation vapor pressure temperature relationship
! GAMMA_PDSI = psychrometric constant
!***************************************************************************
! Calls: LAT_HEA_EVA, SPECUM
! - Authors: S. Cavalletto (01-Jan-2003)
! - Last revision: C. Cassardo (15-May-2014)
!***************************************************************************
USE CONSTANTS
IMPLICIT NONE

CHARACTER(len=90),INTENT(IN) :: PREF
CHARACTER(len=80),INTENT(IN) :: DIROU
INTEGER,INTENT(IN) :: NUM_SOI_LAY
INTEGER,INTENT(IN) :: INNO,INN1
REAL,INTENT(IN) :: TEMP_A_STEP,PRES_A_STEP,Q_SP_A_STEP,AIR_DENS,DELT,SHF_G,RH_STEP,NR_A
REAL,INTENT(IN) :: RESIST_SRF,RESIST_D,SUM_PREC_TMP,SUM_DRAI_TMP,LHF_A
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: PORO
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SR_SOI
REAL,INTENT(IN) :: IANNEW,IMENEW,IGINEW,IORANEW,IMINEW
REAL :: GAMMA_PDSI,DELTA_PDSI,EA_PDSI,ES_PDSI,POT_EVAPO,T_PDSI
REAL :: POT_EVAPO_MM,POT_RECHARGE
REAL :: POT_RUNOFF,POT_LOSS,SS,SU,AWC
REAL :: RLOSS,RECHARGE,EVAPOTRA,RMAX_CAPA,REC_LOW,REC_SURF
REAL,EXTERNAL :: LAT_HEA_EVA
REAL :: PIPPO
INTEGER :: I

SS=0.
SU=0.

T_PDSI=TEMP_A_STEP-DTK

DELTA_PDSI = 10.*4098.*(0.6108*EXP((17.27*T_PDSI)/(T_PDSI+237.3)))/&
 ((T_PDSI+237.3)**2)    ! multiply for 10. to convert in hPa/^C from kPa/^C
GAMMA_PDSI = (CPAIR*PRES_A_STEP)/(0.622*LAT_HEA_EVA(TEMP_A_STEP)) ! hPa/^C
CALL SPECUM(RH_STEP,TEMP_A_STEP,PRES_A_STEP,PIPPO,EA_PDSI,ES_PDSI) ! actual & sat vapor press

! Calculate the potential evapotranspiration from the Penman-Monteith formulation
POT_EVAPO = (DELTA_PDSI*(NR_A-SHF_G)+AIR_DENS*CPAIR*&
 ((ES_PDSI-EA_PDSI)/RESIST_D))/(DELTA_PDSI+GAMMA_PDSI*(1+RESIST_SRF/RESIST_D))  ! W/m2

! Convert the Wm-2 in mm in order to follow the PDSI method
POT_EVAPO_MM = 60.*DELT * POT_EVAPO / LAT_HEA_EVA(TEMP_A_STEP) ! mm

! Following the PDSI formulation, calculate the potential run-off
POT_RUNOFF=0.
DO I=1,NUM_SOI_LAY
```

```
 POT_RUNOFF=POT_RUNOFF+SR_SOI(I)*DEP_SOI(I)*PORO(I)
ENDDO

! Calculates the available water content
AWC=0.
DO I=1,NUM_SOI_LAY
 AWC=AWC+( PORO(I)*DEP_SOI(I) ) *1000.   ! mm
ENDDO

! The potential recharge
POT_RECHARGE = AWC - POT_RUNOFF      ! mm

! The potential loss
IF (PORO(1)*DEP_SOI(1)*SR_SOI(1) >= POT_EVAPO_MM) THEN
 POT_LOSS = POT_EVAPO_MM
ELSE
 SS       = PORO(1)*DEP_SOI(1)*SR_SOI(1)*1000.
 SU       = POT_RUNOFF - SS
 POT_LOSS = SS + ((POT_EVAPO_MM - SS) * SU)/AWC        ! mm
END IF

! Method need potential quantities (computed) as well as actual quantities
! Some of them (evapotranspiration, run-off) are directly computed by UTOPIA
! Some other (loss and recharge) need a parameterization: PDSI formulation
RMAX_CAPA = PORO(1)*DEP_SOI(1)*1000.    ! Maximum water storage of the first layer

! Calculate the recharge and the loss then
! 1) there is enough precipitation to meet the demand from pot.evapotransp.,
!    so evapotranspiration occurs athe the potential rate, there is no loss and
!    there is possibility of recharge

IF (SUM_PREC_TMP*1000. >= POT_EVAPO_MM) THEN
! There is enough extra moisture from precipitation to recharge soil layers
 IF ((SUM_PREC_TMP*1000. - POT_EVAPO_MM) > (RMAX_CAPA - SS)) THEN
  REC_SURF = RMAX_CAPA - SS
  IF ((SUM_PREC_TMP*1000.-POT_EVAPO_MM-REC_SURF) < ((AWC-RMAX_CAPA)-SU)) THEN
   REC_LOW = (SUM_PREC_TMP*1000.-POT_EVAPO_MM-REC_SURF)
  ELSE
   REC_LOW = (AWC-RMAX_CAPA)-SU
  ENDIF
  RECHARGE = REC_LOW + REC_SURF                 ! mm
 ELSE
! There is only moisture to recharge the top soil layer
  RECHARGE = SUM_PREC_TMP*1000. - POT_EVAPO_MM      ! mm
 END IF
 RLOSS = 0.
END IF

! This is the evapotranspiration in mm
EVAPOTRA = 60.*DELT*LHF_A / LAT_HEA_EVA(TEMP_A_STEP)        ! mm

! 2) there is not enough precipitation to satisfy pot.evapotransp., so some
!    soil moisture must be lost. Clearly there cannot be any recharge

IF (SUM_PREC_TMP*1000. < POT_EVAPO_MM) THEN
 RECHARGE = 0.
 IF (EVAPOTRA >= POT_EVAPO_MM) EVAPOTRA = POT_EVAPO_MM
! limit the size of evapotranspiration: this is the loss
 RLOSS = EVAPOTRA - SUM_PREC_TMP*1000.              ! mm
 IF (RLOSS >= POT_LOSS) RLOSS = POT_LOSS
END IF

!???write(67,7001) IANNEW,IMENEW,IGINEW,IORANEW,IMINEW,POT_EVAPO_MM,&
!??? POT_RECHARGE,POT_LOSS,POT_RUNOFF,EVAPOTRA,RECHARGE,RLOSS,&
!??? SUM_DRAI_TMP*1000.,SUM_PREC_TMP*1000.
!???sarebbe meglio mettere in output
```

```
!???7001 FORMAT(1x,I4.4,4I2.2,50(1x,F10.3))

RETURN
END SUBROUTINE PALMER
!!inserirla in usoil???

!*******************************************************************************
SUBROUTINE PHOTOSYNTHESIS(GRO_CAR_ASS_RAT,ALB_FH,NET_CAR_ASS_RAT,PRES_A_STEP,CO2,E_CAN,ES_CAN,
HAJ,HAV,&
 HDJ,HDV,JOPT,LAI_STEP,LMEDLYN,LVEG,MRS,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RESIST_B,
 RAD_S_F_D,&
 RESIST_F_PHOTO,RES_RAT,TEMP_F_STEP,TOPTJ,TOPTV,PVMAX25,VOPT)
!*******************************************************************************
! Routine to calculate the carbon fixation due to photosynthesis
!*******************************************************************************
! Calls: CALC_CAR_ASS_RAT, POTENTIAL, RH_UMID,VERSION_PHOTOSYNTHESIS
! - Author: Ines Cerenzia (06-Feb-2012)
! - Last revision: C. Cassardo (15-May-2014)
!*******************************************************************************
USE CONSTANTS
IMPLICIT NONE

! parameters
REAL,PARAMETER :: O2 = 209. ! O2 air concentration [mmol/mol]

! input
REAL, INTENT(IN) :: ALB_FH
REAL, INTENT(IN) :: PRES_A_STEP     ! air pressure [hPa]
REAL, INTENT(IN) :: CO2             ! CO2 concentration [ppm]
REAL, INTENT(IN) :: E_CAN           ! air-canopy vapor pressure [Pa]
REAL, INTENT(IN) :: ES_CAN          ! air-canopy saturated vapor pressure [Pa]
REAL, INTENT(IN) :: HAJ             ! Activation energy of electron chain reactions [J/mol]
REAL, INTENT(IN) :: HAV             ! Activation energy of Rubisco carboxylase [J/mol]
REAL, INTENT(IN) :: HDJ             ! Deactivation energy of elecron chain reaction [J/mol]
REAL, INTENT(IN) :: HDV             ! Deactivation energy of Rubisco carboxylase [J/mol]
REAL, INTENT(IN) :: JOPT            ! Rate of electron transport at the optimum temperature
[umol/m2s]
REAL, INTENT(IN) :: LAI_STEP        ! LAI
LOGICAL, INTENT(IN) ::LMEDLYN       ! If true, it activates Medlyn parameterization
INTEGER, INTENT(IN) :: LVEG         ! Vegetation code
REAL, INTENT(IN) :: MRS
INTEGER,INTENT(IN) :: PTW
REAL, INTENT(IN) :: PVMAX25         ! [umol/m2s]
REAL, INTENT(IN) :: SSR_FIE_CAP
REAL, INTENT(IN) :: SO_SRAT_RO_DE
REAL, INTENT(IN) :: SSR_WI
REAL, INTENT(IN) :: TOPTJ
REAL, INTENT(IN) :: TOPTV
REAL, INTENT(IN) :: VOPT

! input/output
REAL, INTENT(INout) :: RESIST_B     ! boundary layer resistance [s/m]
REAL, INTENT (INout) :: RAD_S_F_D       ! incident solar global radiation [Wm-2]
REAL, INTENT(INout) :: TEMP_F_STEP      ! Canopy temperature [K]

! output
REAL, INTENT(OUT) :: GRO_CAR_ASS_RAT    ! gross rate of carbon assimilation by photosynthesis
[umol/m2s]
REAL, INTENT(OUT) :: NET_CAR_ASS_RAT    ! net rate of carbon assimilation by photosynthesis
[umol/m2s]
REAL, INTENT(OUT) :: RES_RAT            ! mitocondrial respiration [umol/m2s]
REAL, INTENT(OUT) :: RESIST_F_PHOTO     ! stomatal resistance [s/m]

! dummy
REAL :: AV
REAL :: AJ
REAL :: AC
```

```
REAL :: APR                          ! Atmospheric pressure [Pa]
REAL :: CI_A
REAL :: CI_B
REAL :: CI_C
REAL :: CO2_CON_STEP                        ! CO2 concentration [Pa]
REAL :: F_A
REAL :: F_B
REAL :: F_C
REAL :: F_D
REAL :: GAMMA_STAR                   ! Compensation point [Pa]
INTEGER :: I ! Index
REAL :: J                            ! Rate of electron transport [umol/m2s]
REAL :: KC              ! Michaelis Menten coefficient of Rubisco carboxylation for carbon
[Pa]
REAL :: KO              ! Michaelis Menten coefficient of Rubisco carboxylation for oxygen
[kPa]
REAL :: O2CONT                       ! Oxygen air concentration [kPa]
REAL :: PPFD
REAL :: PVMAX                        ! [umol/m2s]
REAL :: RB                           ! boundary layer resistance [m2s/umol]
REAL :: RS                           ! stomatal resistance [m2s/umol]

! Conversions
APR=PRES_A_STEP*1.e2             ! atmospheric pressure, from hPa to Pa
O2CONT=O2*APR/1.e6           ! oxigen concentration, from mmol/mol to kPa
PPFD = 4.5 * 0.4 * RAD_S_F_D * (1.- ALB_FH) ! Absorbed PAR (photon flux density), from W/m2
to umol/m2s

!Conversione unita di misura resistenza da s/m a m2s/umol
RB=0.025*1.e-6*RESIST_B
!la formula corretta sarebbe RS(m2s/umol) = 10^-6 RS(m2s/mol) = 10^-6 (R THETAatm) / Patm
RS(s/m) = 10^-6 0.024617 RS(s/m)
!assumendo Patm=101325 Pa, R=8.314472 Pa m3/(mol K), THETAatm=300 K si ottiene (R THETAatm) /
Patm = 0.024617 m3/mol

! Conversion from gas concentration to gas partial pressure (Dalton's law)
! If CO2 is not included in the input, it is set equal to 393.84ppm
! (better to create a swicth to activate or not the photosynthesis calc.)
IF (CO2 /= ERR_VE) THEN
 CO2_CON_STEP = CO2*APR/1.e6             ! from ppm to Pa
ELSE
 CO2_CON_STEP = 393.84*APR/1.e6      ! from ppm to Pa
END IF

! If specific coefficients for new formulation exist (check on HDJ just choosing one of them)
! and LMEDLYN is true, uses new parameterization
IF (HDJ /= ERR_VE .AND. LMEDLYN .EQV. .FALSE.) THEN
!IF (LVEG==2 .OR. LVEG== 4 .or. lveg==26) THEN
 CALL VERSION_PHOTOSYNTHESIS('OLD',APR,GAMMA_STAR,HAJ,HAV,HDJ,HDV,J,&
  JOPT,KC,KO,O2CONT,PPFD,PTW,TEMP_F_STEP,TOPTJ,TOPTV,PVMAX,PVMAX25,VOPT)
ELSE
 CALL VERSION_PHOTOSYNTHESIS('NEW',APR,GAMMA_STAR,HAJ,HAV,HDJ,HDV,J,&
  JOPT,KC,KO,O2CONT,PPFD,PTW,TEMP_F_STEP,TOPTJ,TOPTV,PVMAX,PVMAX25,VOPT)
END IF

RES_RAT = (0.015*REAL(PTW)+0.021*REAL(1-PTW)) * PVMAX / (1.+exp(1.3*(TEMP_F_STEP-328.15))) !
mitocondrial respiration [umol/m2s]

! Iteration with bisection method to calculate NET_CAR_ASS_RAT, Ci and Rs in function of
CO2_CON_STEP
CI_A=0.1        ! Pa (--> 1 ppm of CO2 - see below)
CI_B=100.       ! Pa (--> 1000 ppm of CO2=10^-3 mol/mol, as 100 Pa = 10^-3 101300 Pa)

   CONTINUE

DO I=1,20
 IF (ABS(CI_A-CI_B) < 1.e-3) EXIT
```

```
 CI_C=(CI_A+CI_B)/2
 CALL CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI_A,CO2_CON_STEP,E_CAN,
 ES_CAN,F_A,GAMMA_STAR,&
  J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)
 CALL CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI_B,CO2_CON_STEP,E_CAN,
 ES_CAN,F_B,GAMMA_STAR,&
  J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)

! this is the case in which bisector method cannot be applied: the function has the same sign
for both points
! in this case, we try to enlarge the segment by decreasing CI_A or increasing CI_B up to
some empirical threshold, hoping that function will change sign
 IF ((F_A*F_B) > 0 ) THEN
  F_D = MIN(ABS(F_A),ABS(F_B))
  IF (F_D == ABS(F_A)) CI_A = CI_A/2.
  IF (F_D == ABS(F_B)) CI_B = CI_B*2.
! in the following case, CI_A will be too small, risking division by zero - in this case, the
function value corresponding to CI_A will be taken
  IF (CI_A < 1.E-3) THEN
   CALL CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI_A,CO2_CON_STEP,E_CAN,
   ES_CAN,F_A,GAMMA_STAR,&
    J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)
   GOTO 99
  ENDIF
! in the following case, CI_B will be too large, risking division by zero - in this case, the
function value corresponding to CI_B will be taken
  IF (CI_B > 1.E4) THEN
   CALL CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI_B,CO2_CON_STEP,E_CAN,
   ES_CAN,F_B,GAMMA_STAR,&
    J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)
   GOTO 99
  ENDIF
! if the two previous cases will not verify, then it retries to check function values for
modified CI_A and CI_B values
  GOTO 10
 ENDIF
! this is the function value in the middle between CI_A and CI_B, in the case in which
function has different signs there
 CALL CALC_CAR_ASS_RAT(GRO_CAR_ASS_RAT,AV,AJ,AC,NET_CAR_ASS_RAT,APR,CI_C,CO2_CON_STEP,E_CAN,
 ES_CAN,F_C,GAMMA_STAR,&
  J,KC,KO,MRS,O2CONT,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RB,RES_RAT,RS,TEMP_F_STEP,PVMAX)
! this is bisection method - see https://en.wikipedia.org/wiki/Bisection_method
 IF ((F_A*F_C) < 0 )THEN
  CI_B=CI_C
 ELSE
  CI_A=CI_C
 END IF
END DO

   CONTINUE
! Conversion of stomatal resistance from [m2s/umol] to [s/m] and from a
! single leaf stomatal resistance to canopy stomatal resistance
RESIST_F_PHOTO = RS*(1.e6/0.025)/LAI_STEP

RETURN
END SUBROUTINE PHOTOSYNTHESIS

!************************************************************************
SUBROUTINE VERSION_PHOTOSYNTHESIS(VERSION,APR,GAMMA_STAR,HAJ,HAV,HDJ,&
 HDV,J,JOPT,KC,KO,O2CONT,PPFD,PTW,TEMP_F_STEP,TOPTJ,TOPTV,PVMAX,PVMAX25,VOPT)
!************************************************************************
! Calculates kinetics Rubisco coefficients and J, to use in the photosynthesis
! computation, using the parameterization suggested by Collatz, 1991 (old version) and
! the parameterization presented in Medlyn, 2002 (new version)
!************************************************************************
! Calls: none
! - Author: Ines Cerenzia (06-Feb-2012)
```

```
! - Last revision: C. Cassardo (15-May-2014)
!*******************************************************************************
USE CONSTANTS
IMPLICIT NONE

! parameter
REAL,PARAMETER :: FI_C3 = 0.69 ! Photosynthesis quantum yield for C3 plants
REAL,PARAMETER :: FI_C4 = 0.04 ! Photosynthesis quantum yield for C4 plants
REAL,PARAMETER :: FIPSII = 0.385 ! Electron chain quantum yield [mol electrons/mol photon]
REAL,PARAMETER :: THETA = 0.7  ! convexity of the transition btw initial slope and hyperbola
plateau used in J equation (new) []

! input
REAL, INTENT(IN) :: APR ! Air pressure [Pa]
REAL, INTENT(IN) :: HAJ ! Activation energy of electron chain reactions [J/mol]
REAL, INTENT(IN) :: HAV ! Activation energy of Rubisco carboxylase [J/mol]
REAL, INTENT(IN) :: HDJ ! Deactivation energy of elecron chain reaction [J/mol]
REAL, INTENT(IN) :: HDV ! Deactivation energy of Rubisco carboxylase [J/mol]
REAL, INTENT(IN) :: JOPT ! Electron transp rate at the opt.temp. [umol/m2s]
REAL, INTENT(IN) :: O2CONT ! O2 concentration [kPa]
REAL, INTENT(IN) :: PPFD ! Absorbed PAR (photon flux density) [umol/m2s]
INTEGER, INTENT(IN) :: PTW ! Photosynthesis pathway (1=c3, 0=c4)
REAL, INTENT(IN) :: TEMP_F_STEP ! Canopy temperature [K]
REAL, INTENT(IN) :: TOPTJ ! Optimum temperature of the elecron transport [K]
REAL, INTENT(IN) :: TOPTV ! Optimum temperature of Rubisco carboxylation [K]
CHARACTER(LEN=3),INTENT(IN) :: VERSION ! Version of parameterization
REAL, INTENT(IN) :: PVMAX25 ! Max rate of Rubisco carbox at 25C [umol/m2s]
REAL, INTENT(IN) :: VOPT ! Max Rubisco carbox rate at opt.temp. [umol/m2s]

! output
REAL, INTENT(OUT) :: GAMMA_STAR ! Compensation point [Pa]
REAL, INTENT(OUT) :: KC ! Michaelis Menten coeff of Rubisco carboxylation for C [Pa]
REAL, INTENT(OUT) :: KO ! Michaelis Menten coeff of Rubisco carboxylation for O [kPa]
REAL, INTENT(OUT) :: J ! Electron transport rate [umol/m2s]
REAL, INTENT(OUT) :: PVMAX ! Maximum rate of Rubisco carboxylation [umol/m2s]

! dummy
REAL :: DISC
REAL :: JMAX
REAL :: Q10 ! Coefficient

IF (VERSION=='OLD') THEN !method by Farquhar(1980)->Sellers(1992)

 Q10=(TEMP_F_STEP-298.15)/10.
 KC=30.*2.1**Q10    !Pa - in truth should be divided by 10^6 (mol/mol) & mult. by 101325 Pa
 KO=30.*1.2**Q10    !kPa - in truth should be divided by 10^3 (mol/mol) & mult. by 101.325 kPa
 GAMMA_STAR=0.5*KC/KO*(O2CONT)*0.21 !Pa - O2CONT in kPa, KC in Pa, KO in kPa --> GAMMA_STAR
 come KC
 !PVMAX = PVMAX25*REAL(PTW)*(2.4**Q10) /
 (1+exp((-2.2e5+710*TEMP_F_STEP)/(RGAS*TEMP_F_STEP)))+PVMAX25*REAL(1.-PTW)  !umol/m2s
 PVMAX = PVMAX25*REAL(PTW)*(2.4**Q10) / (1.+exp((-2.2e5+710.*(TEMP_F_STEP-10.))/(RGAS*&
  (TEMP_F_STEP-10.))))+PVMAX25*REAL(1.-PTW) !umol/m2s

 !!!!!VMAX=35.*REAL(PFT)*(2.4**Q10)/(1+exp((-2.2e5+710.*(TEMP_F_STEP-10.))/(R*(TEMP_F_STEP-10.
 ))))+30.*(1-PFT)  !umol/m2s
 !NBNBNB: nei test era 35 per PTW e 30 per 1-PTW, mentre qui e fisso tra PTW e 1-PTW ma varia
 col tipo di pianta tra 33 e 35
 J = (FI_C3*REAL(PTW)+FI_C4*REAL(1-PTW)) * PPFD !DA CLM  umol/m2s

ELSE IF (VERSION=='NEW') THEN ! method by (Bernacchi,2001)

! multiplication by (APR*1.e-6) converts from umol/mol to Pa
 KC=404.9*EXP(79430.*(TEMP_F_STEP-298.)/(298.*RGAS*TEMP_F_STEP)) * (APR*1.e-6)          ! [Pa]
 KO=278.4*EXP(36380.*(TEMP_F_STEP-298.)/(298.*RGAS*TEMP_F_STEP)) * (APR*1.e-6)          ! [Pa]
 GAMMA_STAR=42.75*EXP(37830.*(TEMP_F_STEP-298.)/(298.*RGAS*TEMP_F_STEP)) * (APR*1.e-6)  ! [Pa]
! compute of Vmax and J with Arrhenius modified function
 PVMAX=VOPT*HDV*EXP(HAV*(TEMP_F_STEP-TOPTV)/(TEMP_F_STEP*RGAS*TOPTV))/&
```

```
  (HDV-HAV*(1.-EXP(HDV*(TEMP_F_STEP-TOPTV)/(TEMP_F_STEP*RGAS*TOPTV))))

 JMAX=JOPT*HDJ*EXP(HAJ*(TEMP_F_STEP-TOPTJ)/(TEMP_F_STEP*RGAS*TOPTJ))/&
  (HDJ-HAJ*(1.-EXP(HDJ*(TEMP_F_STEP-TOPTJ)/(TEMP_F_STEP*RGAS*TOPTJ))))

 DISC=(FIPSII*PPFD+JMAX)**2-4.*FIPSII*PPFD*THETA*JMAX
 J=(FIPSII*PPFD+JMAX - SQRT(DISC))/(2.*THETA)

END IF

END SUBROUTINE VERSION_PHOTOSYNTHESIS

!*****************************************************************************
SUBROUTINE POTENTIAL(P,Q,T,THETA,THETA_V)
!*****************************************************************************
! Calculates potential & virtual potential temperature given a temperature
!*****************************************************************************
! Calls: none
! - Author: M. Romani (21-Apr-2004)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL :: K
REAL, INTENT(IN) :: P              !air pressure (hPa)
REAL, INTENT(IN) :: Q              !specific humidity
REAL, INTENT(IN) :: T              !temperature in Kelvin degree
REAL, INTENT(OUT) :: THETA         !potential temperature
REAL, INTENT(OUT) :: THETA_V       !virtual potential temperature

K = ERRED * (1 - 0.23*Q)/CPAIR
THETA = T * (P0_REF/P)**K          ! eq.3.35 Brutsaert (1982)
THETA_V = ( 1 + 0.61*Q) * THETA    ! eq. 3.37a  Brutsaert (1982)

RETURN
END SUBROUTINE POTENTIAL

!*****************************************************************************
SUBROUTINE RADIATION(ALB_F,ALB_G,ALB_SN,Q_SP_A_STEP,TEMP_A_STEP,CL_TOT_STEP,D_NR_F_D_TF,&
 EPSF,EPSG,HAZE_IND,LLONGWFLAG,NUM_SOI_LAY,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,RAD_L_G_D,
 RAD_L_G_U,RAD_L_D,NR_L_F_D,&
 RAD_L_SN_F_D,RAD_L_SN_F_U,NR_L_F_U,RAD_L_G_TOT_D,RAD_L_G_TOT_U,RAD_L_SN_G_D,RAD_L_SN_G_U,
 RAD_L_SN_U,NR_L_SN_D,NR_L_SN_U,RAD_L_U,RAD_L_F_D,RAD_L_F_U,RAD_L_FG_D,&
 RAD_L_FG_U,NR_A,NR_F,NR_G,NR_SN,RAD_S_D,RAD_S_F_D,RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_F_U,
 RAD_S_G_D,RAD_S_SN_G_D,RAD_S_SN_G_U,RAD_S_G_U,&
 RAD_S_SN_D,RAD_S_SN_U,RAD_S_U,VEG_COV_STEP,COVER_SN_F,COVER_SN_G)
!*****************************************************************************
! Performs the calculation of all radiation term of bare soil, canopy and snow.
!*****************************************************************************
! Calls: LONGW, SHORTW
! - Authors: C. Cassardo, J.J. Ji, E. Carena (16-Jun-1994)
! - Last revision : C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: ALB_F
REAL,INTENT(IN) :: ALB_G
REAL,INTENT(IN) :: ALB_SN
REAL,INTENT(IN) :: Q_SP_A_STEP
REAL,INTENT(IN) :: TEMP_A_STEP
REAL,INTENT(IN) :: CL_TOT_STEP
REAL :: D_NR_L_F_D_D_TF
REAL :: D_NR_L_F_U_D_TF
REAL,INTENT(OUT) :: D_NR_F_D_TF
```

```
REAL,INTENT(IN) :: EPSF
REAL,INTENT(IN) :: EPSG
REAL,INTENT(IN) :: HAZE_IND
LOGICAL,INTENT(IN) :: LLONGWFLAG
REAL,INTENT(IN) :: P_TEMP_F_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL,INTENT(IN) :: P_TEMP_SN
REAL,INTENT(OUT) :: RAD_L_G_D
REAL,INTENT(OUT) :: RAD_L_G_U
REAL,INTENT(INOUT) :: RAD_L_D
REAL,INTENT(OUT) :: NR_L_F_D
REAL,INTENT(OUT) :: RAD_L_SN_F_D
REAL,INTENT(OUT) :: RAD_L_SN_F_U
REAL,INTENT(OUT) :: NR_L_F_U
REAL,INTENT(OUT) :: RAD_L_G_TOT_D
REAL,INTENT(OUT) :: RAD_L_G_TOT_U
REAL,INTENT(OUT) :: RAD_L_SN_G_D
REAL,INTENT(OUT) :: RAD_L_SN_G_U
REAL,INTENT(OUT) :: RAD_L_SN_U
REAL,INTENT(OUT) :: NR_L_SN_D
REAL,INTENT(OUT) :: NR_L_SN_U
REAL,INTENT(OUT) :: RAD_L_U
REAL,INTENT(OUT) :: RAD_L_F_D
REAL,INTENT(OUT) :: RAD_L_F_U
REAL,INTENT(OUT) :: RAD_L_FG_D
REAL,INTENT(OUT) :: RAD_L_FG_U
REAL,INTENT(OUT) :: NR_A
REAL,INTENT(OUT) :: NR_F
REAL,INTENT(OUT) :: NR_G
REAL,INTENT(OUT) :: NR_SN
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(OUT) :: RAD_S_F_D
REAL,INTENT(OUT) :: RAD_S_SN_F_D
REAL,INTENT(OUT) :: RAD_S_SN_F_U
REAL,INTENT(OUT) :: RAD_S_F_U
REAL,INTENT(OUT) :: RAD_S_G_D
REAL,INTENT(OUT) :: RAD_S_SN_G_D
REAL,INTENT(OUT) :: RAD_S_SN_G_U
REAL,INTENT(OUT) :: RAD_S_G_U
REAL,INTENT(OUT) :: RAD_S_SN_D
REAL,INTENT(OUT) :: RAD_S_SN_U
REAL,INTENT(OUT) :: RAD_S_U
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN_F
REAL,INTENT(IN) :: COVER_SN_G

! longwave radiation terms calculation
CALL LONGW(Q_SP_A_STEP,TEMP_A_STEP,CL_TOT_STEP,D_NR_L_F_D_D_TF,D_NR_L_F_U_D_TF,EPSF,EPSG,
HAZE_IND,&
 LLONGWFLAG,NUM_SOI_LAY,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,RAD_L_G_D,RAD_L_G_U,RAD_L_D,NR_L_F_D
 ,RAD_L_SN_F_D,RAD_L_SN_F_U,&
 NR_L_F_U,RAD_L_G_TOT_D,RAD_L_G_TOT_U,RAD_L_SN_G_D,RAD_L_SN_G_U,RAD_L_SN_U,NR_L_SN_D,NR_L_SN_U
 ,RAD_L_U,RAD_L_F_D,RAD_L_F_U,RAD_L_FG_D,RAD_L_FG_U,VEG_COV_STEP,&
 COVER_SN_F,COVER_SN_G)

! shortwave radiation terms calculation
CALL SHORTW(ALB_F,ALB_G,ALB_SN,RAD_S_D,RAD_S_F_D,RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_F_U,RAD_S_G_D
,RAD_S_SN_G_D,&
 RAD_S_SN_G_U,RAD_S_G_U,RAD_S_SN_D,RAD_S_SN_U,RAD_S_U,VEG_COV_STEP,COVER_SN_F,COVER_SN_G)

! net radiation terms calculation, in W/m2
NR_A = RAD_S_D-RAD_S_U+RAD_L_D-RAD_L_U  ! total
NR_F = RAD_S_F_D-RAD_S_F_U+NR_L_F_D-NR_L_F_U        ! total entering in vegetation
NR_G = RAD_S_G_D-RAD_S_G_U+RAD_L_G_TOT_D-RAD_L_G_TOT_U      ! total entering into soil
NR_SN = RAD_S_SN_D-RAD_S_SN_U+NR_L_SN_D-NR_L_SN_U   ! total entering into snow

! derivative of net radiation with respect to TEMP_F_STEP
```

```
D_NR_F_D_TF = D_NR_L_F_D_D_TF-D_NR_L_F_U_D_TF

RETURN
END SUBROUTINE RADIATION

!***********************************************************************
SUBROUTINE RAD_TILT(ALBEDO,CLEG,DOY,ESSE,GAMMA,HHH,IMI,LATI,LONG,RSmhor,RS)
!***********************************************************************
! Calculates the solar radiation at a specific site. The value is obtained using
! the suggestion reported in the paper by Allen et al. (2006), Agric. for. Met.
!***********************************************************************
! - Author: C. Cassardo (15-Mar-2013)
! - Last revision: C. Cassardo (15-May-2014)
!***********************************************************************
IMPLICIT NONE

INTEGER :: II, KK
REAL, EXTERNAL :: ASIND, COSD, SIND
REAL, INTENT(IN) :: ALBEDO      ! Albedo of surface
REAL, INTENT(IN) :: CLEG        ! summer time lag or time shift, in hours
REAL :: COST                    ! cosine of solar angle
REAL :: COSThor                 ! cosine of solar angle on horizontal surface
REAL :: DELTA                   ! solar declination
REAL :: DIST                    ! relative Earth-Sun distance in astronomic units
INTEGER, INTENT(IN) :: DOY      ! Julian day (1 -> 365, 1=1st jan.)
REAL, INTENT(IN) :: ESSE        ! Surface slope angle, always positive
                                ! (=0^ horizontal, >0^ tilted, =90^ vertical)
REAL :: ET                      ! equation of time, hours, Page eq. A1.5.1
REAL :: FB                      ! ratio between rad on tilted and horizontal srf
REAL :: Fi                      ! ratio between diffuse radiation on a tilted surface vs
diffuse radiation on a flat surface - isotropic calculation
REAL :: Fia                     ! ratio between diffuse radiation on a tilted surface vs
diffuse radiation on a flat surface - anisotropic calculation
REAL, INTENT(IN) :: GAMMA       ! Surface aspect angle
                                ! (=0^ S, =-90^ E, =90^ W, =+/-180^ N)
REAL :: GEIP                    ! day angle (deg), Page eq. A1.2
REAL, PARAMETER  :: Gsc = 1367. ! solar constant (W m-2)
INTEGER, INTENT(IN) :: HHH      ! hour
INTEGER, INTENT(IN) :: IMI      ! minutes
REAL :: KBhor                   ! clearness index for direct rad for horiz surf
REAL :: KDhor                   ! clearness index for diffuse rad on hor srf cloudy
REAL, INTENT(IN) :: LATI        ! Latitude in degrees (-90 -> 90, >0 in NH, <0 in SH)
REAL, INTENT(IN) :: LONG        ! Longitude in degrees
                                ! (-180 -> 180, >0 E of Greenwich, <0 W of Greenwich)
REAL :: LONST                   ! mean lon of Earth slice of 15 deg of time zone
REAL :: OMEGA                   ! solar hour angle, deg, Page eq. A1.6
REAL :: RAhor                   ! theoretical solar global radiation on a horiz srf
REAL, INTENT(IN) :: RSmhor      ! measured solar global radiation on horiz srf
REAL, INTENT(OUT) :: RS         ! Global shortwave radiation (Wm-2)
REAL :: T                       ! local apparent time, h, Page eq. A1.5.2
REAL :: TAUswhor                ! actual atmospheric transmissivity (direct+diffuse)
                                ! for horizontal surface = Rsm/Ra
REAL :: TT                      ! local standard time, hours

! Determination of longitude of time reference zones (deg, Page pag 393):
! it is the slice of earth of 15 degrees of time zone (of mean
! longitude ALONST) in which there is the point of coordinates ALON, ALAT
! Attention: point ALON-ALAT could not be in the middle of the slice thus each
! point is assigned to a slice depending on ALONST
LONST=LONG
DO II=1,24
 KK=II-13           ! search interval of ALON from -180? to +180?
 IF (LONG > REAL(KK*15) .AND. LONG <= REAL((KK+1)*15)) THEN
  LONST=REAL((KK+1)*15)      ! at the point of longit ALON we assign the slice
  EXIT                        !with the superior extreme of the 15deg interval
 END IF
END DO
```

```
TT=REAL(HHH)+REAL(IMI)/60.    !local standard time, hours
GEIP=360.*REAL(DOY)/365.25    !day angle (deg), Page eq. A1.2

ET=-0.128*SIND(GEIP-2.8)-0.165*SIND(2*GEIP+19.7)    ! equation of time, hours,
                                                    ! Page eq. A1.5.1

T=TT+(LONG-LONST)/15.+ET-CLEG                      ! local apparent time,
                                                    ! h, Page eq. A1.5.2

DELTA=ASIND(0.3978*SIND(GEIP-80.2+1.92*SIND(GEIP-2.80)))     ! solar declination,
                                                           ! deg, Page eq. A1.3

OMEGA=15.*(T-12)                          ! solar hour angle, deg, Page eq. A1.6

! calculation of the cosine of solar angle on horizontal surface
COSThor = SIND(DELTA)*SIND(LATI)+COSD(DELTA)*COSD(LATI)*COSD(OMEGA)

! calculation of the cosine of solar angle on tilted surface
COST = SIND(DELTA)*SIND(LATI)*COSD(ESSE) &
       - SIND(DELTA)*COSD(LATI)*SIND(ESSE)*COSD(GAMMA) &
       + COSD(DELTA)*COSD(LATI)*COSD(ESSE)*COSD(OMEGA) &
       + COSD(DELTA)*SIND(LATI)*SIND(ESSE)*COSD(GAMMA)*COSD(OMEGA) &
       + COSD(DELTA)*SIND(GAMMA)*SIND(ESSE)*SIND(OMEGA)

! calculation of the relative Earth-Sun distance in astronomic units
! NB: with respect to the Allen et al. (2006) paper, the denominator has been set to
! 365.25 and not 365 for accounting leap years
DIST = 1. + 0.033 * COSD(REAL(GEIP))

! calculation of the ...
Fb = COST / COSThor

!calculation of the theoretical solar global radiation on a horizontal surface
RAhor = GSC * COSThor / (DIST*DIST)

! calculation of the ...
TAUswhor = RSmhor / RAhor

! clearness index for direct beam radiation for horizontal surface
IF (TAUswhor >= 0.42) THEN
 KBhor = 1.56*TAUswhor - 0.55
ELSE IF (TAUswhor <= 0.175) THEN
 KBhor = 0.016*TAUswhor
ELSE
 KBhor = 0.022 - 0.280*TAUswhor + 0.828*TAUswhor*TAUswhor &
         + 0.765*TAUswhor*TAUswhor*TAUswhor
ENDIF

! clearness index for diffuse beam radiation on horizontal surface
KDhor = TAUswhor - KBhor

! ratio between diffuse radiation on a tilted surface vs diffuse radiation on a flat
! surface - isotropic calculation
Fi = 0.75 + 0.25 * COSD(ESSE) - ESSE/360.

! ratio between diffuse radiation on a tilted surface vs diffuse radiation on a flat
! surface - anisotropic calculation
Fia = (1.-KBhor) * (1. + SQRT(KBhor/(KBhor+KDhor))*SIND(ESSE/2.)**3) * Fi &
      + Fb*KBhor

! calculation of solar global rad on tilted srf starting from the measured one RSmhor
RS = RSmhor  * ( FB*KBhor/TAUswhor + Fia*KDhor/TAUswhor + ALBEDO*(1.-Fi) )

RS = RS/COSD(ESSE)

RETURN
```

```
END SUBROUTINE RAD_TILT

!****************************************************************************
SUBROUTINE RESIST(CO2_CON_STEP, COND_F_DRY, COND_F_WET, COND_G_DRY, COND_G_WET, COND_SRF_VAP,&
 CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F,
 CONDUCT_SRF_SN,&
 D0_VEGPAR, DRAG_H, DRAG_M, DRAG_V, LAI_STEP,LVEG, NUM_SOI_LAY, SR_SOI, P_TEM_SOI, PRES_A_STEP
 ,&
 Q_SP_A_STEP, RAD_S_D, REL_LEA_WAT_CON, RGL_NOILHAN, REL_SRF_WAT_CON, RESIST_AH, RESIST_AM,&
 RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_MIN, RESIST_F_PHOTO,
 RESIST_LEAF,&
 RESIST_SRF, RESIST_SRF_SN, SO_SRAT_RO_DE, SSR_FIE_CAP, SSR_WI, TEMP_A_STEP, TEMP_ROOT, UAF,
 USTAR,&
 VEG_COV_STEP, VEG_HEI_STEP, VEL_MOD_STEP, Z0_F_H, Z0_G_H, Z0_SN, ZER_DIS_LEV)
!****************************************************************************
! Evaluates resistances and conductivities
!****************************************************************************
! Calls: SPECUM
! - Author : C. Cassardo, J.J. Ji, P. Ramieri (22-Mar-1994)
! - Last revision : C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL, INTENT(IN) :: CO2_CON_STEP    !CO2 (ppm)
REAL, INTENT(OUT) :: COND_F_DRY !dry canopy conductivity for vapor
REAL, INTENT(OUT) :: COND_F_WET !wet canopy conductivity for vapor
REAL, INTENT(OUT) :: COND_G_DRY !dry bare soil conductivity for vapor
REAL, INTENT(OUT) :: COND_G_WET !wet bare soil conductivity for vapor
REAL, INTENT(OUT) :: COND_SRF_VAP   !Low air-canopy space conductivity for vapor
REAL, INTENT(OUT) :: CONDUCT_AH !air conductivity for heat
REAL, INTENT(OUT) :: CONDUCT_AM !air conductivity for momentum
REAL, INTENT(OUT) :: CONDUCT_AV !air conductivity for vapor
REAL, INTENT(OUT) :: CONDUCT_B  !laminar conductivity of air near the foliage
REAL, INTENT(OUT) :: CONDUCT_D  !Low air-canopy space conductivity for heat and vapor
REAL, INTENT(OUT) :: CONDUCT_D_SN   !conductivity for snow corresponding to CONDUCT_D
REAL, INTENT(OUT) :: CONDUCT_F  !canopy conductivity for vapor
REAL, INTENT(OUT) :: CONDUCT_SRF_SN !conductivity for snow corresponding to COND_SRF_VAP
REAL, INTENT(IN) :: D0_VEGPAR   !dimension of leaves
REAL, INTENT(IN) :: DRAG_H      !heat transfer coefficient
REAL, INTENT(IN) :: DRAG_M      !drag coefficient
REAL, INTENT(IN) :: DRAG_V      !bulk transfer coefficient for water vapor
REAL :: DRYF                !percentage of dry canopy (>0.1)
REAL :: DRYG                !percentage of dry bare soil (>0.1)
REAL :: EFFE
REAL :: FF1
REAL :: FF2
REAL :: FF3
REAL :: FF4
REAL :: FF5
REAL, INTENT(IN) :: LAI_STEP    !leaf area index
INTEGER, INTENT(IN) :: LVEG !vegetation type code (see BATS)
INTEGER, INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SR_SOI    !current relative soil moisture
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI !current relative soil temperature
REAL :: PIPPO               !not useful variable
REAL, INTENT(IN) :: PRES_A_STEP !current pressure at H_OBS_ALL level
REAL, INTENT(IN) :: Q_SP_A_STEP !current humidity at H_OBS_ALL level
REAL, INTENT(IN) :: RAD_S_D !Current global radiation
REAL, INTENT(IN) :: REL_LEA_WAT_CON !percentage of wetness canopy
REAL, INTENT(IN) :: REL_SRF_WAT_CON !percentage of wetness bare soil
REAL, INTENT(OUT) :: RESIST_SRF !bare soil resistance
REAL, INTENT(OUT) :: RESIST_SRF_SN
REAL, INTENT(OUT) :: RESIST_AH
REAL, INTENT(OUT) :: RESIST_AM  !bulk aerod resist for momentum
REAL, INTENT(OUT) :: RESIST_AV
REAL, INTENT(OUT) :: RESIST_B   !laminar resistance of air near the foliage
```

```
REAL, INTENT(OUT) :: RESIST_D   !aerodynamic air resistance (Z0*-VEG_HEI_STEP)
REAL, INTENT(OUT) :: RESIST_D_SN
REAL, INTENT(OUT) :: RESIST_F   !canopy stomatal resistance
REAL, INTENT(IN) :: RESIST_F_MIN    !minimum stomatal resistance
REAL, INTENT(IN) :: RESIST_F_PHOTO  ! stomatal resistance from photosynthetic calculation
[s/m]
REAL, INTENT(OUT) :: RESIST_LEAF    !leaf stomatal resistance
REAL,INTENT(IN) :: RGL_NOILHAN
REAL :: SAT_SPHU_A      !saturated specific humidity at TEMP_A_STEP
REAL, INTENT(IN) :: SO_SRAT_RO_DE   !Average soil moisture in root zone
REAL, INTENT(IN) :: SSR_FIE_CAP     !Relative field capacity
REAL, INTENT(IN) :: SSR_WI      !Relative wilting point
REAL, INTENT(IN) :: TEMP_A_STEP !current air temperature at H_OBS_ALL level
REAL, INTENT(IN) :: TEMP_ROOT   !mean soil temperature in canopy root zone
REAL, INTENT(IN) :: UAF     !top wind velocity within the canopy
REAL, INTENT(IN) :: USTAR
REAL, INTENT(IN) :: VEG_COV_STEP    !vegetation cover
REAL, INTENT(IN) :: VEG_HEI_STEP    !height of canopy
REAL, INTENT(IN) :: VEL_MOD_STEP    !current horizontal wind velocity
REAL :: WETF                !percentage of wet canopy (dry canopy >0.1)
REAL :: WETG    !percentage of wet bare soil (dry  >0.1)
REAL, INTENT(IN) :: Z0_F_H  !aerodynamic roughness lenght for heat for canopy
REAL, INTENT(IN) :: Z0_G_H  !aerodynamic roughness lenght for heat for bare soil
REAL, INTENT(IN) :: Z0_SN   !aerodynamic roughness lenght for snow
REAL, INTENT(IN) :: ZER_DIS_LEV !zero displacement level

! A) aerodynamic resistances
RESIST_AH = 1.0/(DRAG_H * VEL_MOD_STEP)  !eq.3.56 G94
RESIST_AM = 1.0/(DRAG_M * VEL_MOD_STEP)  !eq.3.53 G94
RESIST_AV = 1.0/(DRAG_V * VEL_MOD_STEP)  !eq.3.57 G94

! B) boundary layer resistance over leaves
IF (VEG_COV_STEP /= 0. .AND. LAI_STEP /= 0.) THEN ! Only if there is Vegetation
 RESIST_B = MAX(1.,193.0825*SQRT(D0_VEGPAR/UAF)/LAI_STEP) ! !param.modif from p.63 Bonan (div
 LAI)
ELSE
 RESIST_B = 1./EPSIL      ! =>infinite
END IF

! C) below canopy aerodynamic resistance - from Bonan (1996) pag 62-3, but:
!    1) put VEG_HEI_STEP instead of H_OBS_ALL
!    2) put ustar instead of VEL_MOD_STEP*sqrt(DRAG_M)
!    3) approssimate Z0_G_H=Z0_H and put Z0_G_H

RESIST_D = VEG_HEI_STEP/(A_RESIST*VON_KARMAN*USTAR*(VEG_HEI_STEP-ZER_DIS_LEV))*(EXP(A_RESIST*&
 (1.-Z0_F_H/VEG_HEI_STEP))-EXP(A_RESIST*(1.-(Z0_G_H+ZER_DIS_LEV)/VEG_HEI_STEP)))
!!!RESIST_D = MAX(EPSIL,RESIST_D)
RESIST_D = MAX(15.,RESIST_D)

! D) resistances over snow

RESIST_D_SN= VEG_HEI_STEP/(A_RESIST*VON_KARMAN*USTAR*(VEG_HEI_STEP-ZER_DIS_LEV))*(EXP(A_RESIST
*&
 (1.-Z0_SN/VEG_HEI_STEP))-EXP(A_RESIST*(1.-(Z0_SN+ZER_DIS_LEV)/VEG_HEI_STEP)))
RESIST_D_SN= MAX(EPSIL,RESIST_D_SN)

! E) Soil resistances

IF (LVEG /= 14 .AND. LVEG /= 15 .AND. LVEG /= 12) THEN ! all landuse but ice/water
 RESIST_SRF =EXP(8.206-4.255 * SR_SOI(1)) !Sellers et al. (1992) see pag 187 GandPC
ELSE IF (LVEG.EQ.14.OR.LVEG.EQ.15) THEN ! water
 RESIST_SRF = 0.
ELSE IF (LVEG.EQ.12) THEN ! ice
! formulation of M. Qian (REF=???)
! PICE=2.*ABS(QSURF-Q_SP_A_STEP)/(QSURF+Q_SP_A_STEP)
! RESIST_SRF=10.*(1.+EXP(PICE*4.))
! IF ((TEMP_A_STEP-273.15).GT.-3.0 .AND. RAD_S_D.GT.400.0) RESIST_SRF=100.
```

```
 RESIST_SRF=100.
!???preferirei la formula di Qian
END IF
RESIST_SRF = MAX(EPSIL,RESIST_SRF)

!if surface soil is frozen, this resistance is set to infinite
IF (P_TEM_SOI(1)<DTK) RESIST_SRF=150. ! as for snow

! F) Canopy resistance - version of Dickinson (1984)
! canopy resistance depends on 4 factors (plus CO2)
!initialize in order to give RF=1/EPSIL (infinite) if no vegetation
FF1=1./EPSIL
FF2=1.
FF3=1.
FF4=1.
FF5=1.

IF (VEG_COV_STEP /= 0. .AND. LAI_STEP /= 0.) THEN ! Only if there is Vegetation

 IF (RESIST_F_PHOTO == ERR_VE) THEN

! 1) radiation, follows Dickinson (1984)
  CALL SPECUM(100.,TEMP_A_STEP,PRES_A_STEP,SAT_SPHU_A,PIPPO,PIPPO) !sat spec hum SAT_SPHU_A
  at TEMP_A_STEP
  EFFE=1.1*RAD_S_D/RGL_NOILHAN/LAI_STEP
  FF1=(RESIST_F_MIN/RSMAX+EFFE)/(1.+EFFE)

! 2) soil moisture, follws ISBA version
  FF2=(SO_SRAT_RO_DE-SSR_WI)/(SSR_FIE_CAP-SSR_WI)

! 3) vapor pressure deficit dependence, follows Dickinson (1984)
  FF3=1.-60.*(SAT_SPHU_A-Q_SP_A_STEP)

! 4) temperature dependence, follows Dickinson (1984) with variable optimum temp
  FF4=1.-0.0016*(T_OPT-TEMP_A_STEP)*(T_OPT-TEMP_A_STEP)

! 5) CO2 dependence, only if there are CO2 values - follows Prino et al. (article)
  IF(CO2_CON_STEP /= ERR_VE)THEN
   IF(CO2_CON_STEP<=400.)THEN
    FF5=EXP(0.0027*(CO2_CON_STEP-400.))
   ELSE
    FF5=1-0.0013*(CO2_CON_STEP-400.)
   END IF
  END IF

! FF* functions must range between 0 and 1
  FF1=MIN(1.,MAX(EPSIL,FF1))
  FF2=MIN(1.,MAX(FF2MIN,FF2))
  FF3=MIN(1.,MAX(FF3MIN,FF3))
  FF4=MIN(1.,MAX(EPSIL,FF4))
  FF5=MIN(1.,MAX(EPSIL,FF5))

  RESIST_LEAF = RESIST_F_MIN/(FF1*FF2*FF3*FF4*FF5)  !eq.7.47 pag 279 dingman
  RESIST_F = MIN(RSMAX,RESIST_LEAF)/LAI_STEP

 ELSE
! UTOPIA uses the calculation performed using the photosynthetic calculation
  RESIST_F = MIN(RSMAX,RESIST_F_PHOTO)

 ENDIF

END IF

!modifications of canopy resistance for frozen soils (set to ZERO)
!claudio
!!!IF (TEMP_ROOT<DTK) RESIST_F=1./EPSIL
```

```
!snow
RESIST_SRF_SN = 150. !s/m

!====================conductances

! aerodynamic above vegetation
CONDUCT_AH = 1.0/RESIST_AH
CONDUCT_AM = 1.0/RESIST_AM
CONDUCT_AV = 1.0/RESIST_AV

! aerodynamic below vegetation
CONDUCT_D = 1.0/RESIST_D ! below canopy

! vegetation
CONDUCT_B=EPSIL !initialized null if not vegetation
CONDUCT_F=EPSIL
COND_F_DRY=EPSIL
COND_F_WET=EPSIL

IF (VEG_COV_STEP /= 0.) THEN
 CONDUCT_B = 2.0/RESIST_B      !laminar for a leaf: 2 because a leaf has 2 sfc
 DRYF = MAX((1.0 - REL_LEA_WAT_CON), 0.1)
 WETF = 1. - DRYF
 COND_F_DRY = DRYF / (RESIST_B+RESIST_F)    !for dry canopy: transpiration
 COND_F_WET = WETF / (RESIST_B+RESIST_F)    !for wet canopy: evaporation
 CONDUCT_F = COND_F_DRY + COND_F_WET            !general for canopy
 CONDUCT_F = MIN(1./EPSIL,CONDUCT_F)   !to avoid 1/0
END IF

! bare soil
COND_SRF_VAP=EPSIL !initialized null if not bare soil
COND_G_DRY=EPSIL
COND_G_WET=EPSIL

IF (VEG_COV_STEP /= 1.) THEN
 DRYG = MAX((1.0 - REL_SRF_WAT_CON),0.1)
 WETG = 1. - DRYG
 COND_G_DRY = DRYG / (RESIST_D + RESIST_SRF)!for wet bare soil: evaporation from soil
 COND_G_WET = WETG / RESIST_D           !for wet bare soil: evaporation from water
 COND_SRF_VAP = COND_G_DRY + COND_G_WET         !general for bare soil
 COND_SRF_VAP = MIN(1./EPSIL,COND_SRF_VAP)   !to avoid 1/0
ENDIF

! snow
CONDUCT_D_SN=1./RESIST_D_SN ! for snow
CONDUCT_SRF_SN=1./(RESIST_D_SN+RESIST_SRF_SN)

RETURN
END SUBROUTINE RESIST

!*****************************************************************************
REAL FUNCTION RH_UMID(Q,AT,APR)
!*****************************************************************************
!   Routine to calculate the relative humidity starting from specific humidity
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: APR ! Pressure [hPa]
REAL,INTENT(IN) :: AT  ! Temperature [K]
REAL :: E
REAL :: ES
REAL,INTENT(IN) :: Q   ! Specific humidity [kg/kg]
```

```
ES=A1*EXP(A2*(AT-A3)/(AT-A4)) ! saturated water vapor pressure (hPa)       !
E=Q*APR/(A6*Q+A5)  ! water vapor pressure (hPa)
RH_UMID=100.*E/ES
IF (RH_UMID > 100. .AND. RH_UMID < 105.) RH_UMID=100. ! manage instrumental errors

RETURN
END FUNCTION RH_UMID

!*****************************************************************************
SUBROUTINE SELECT_SOIL(NUM,ETA_S,PSIS,CAPPAETA_S,B,ETA_WI,ROCI,SSR_FIE_CAPC)
!*****************************************************************************
! Assigns the correct value of parameter according with soil code type. For details
! see module SOIL_VALUES at the beginning of main program.
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
USE SOIL_VALUES
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM
REAL,INTENT(OUT) :: B           ! Parameter depending on type of soil []
REAL,INTENT(OUT) :: CAPPAETA_S  ! Soil moisture conductivity [m2s-1]
REAL,INTENT(OUT) :: ETA_S       ! Soil porosity [m3/m3]
REAL,INTENT(OUT) :: ETA_WI      ! Wilting point [m3/m3]
REAL,INTENT(OUT) :: PSIS        ! Soil moisture potential [m]
REAL,INTENT(OUT) :: SSR_FIE_CAPC        ! Field capacity [units of saturation ratio]
REAL,INTENT(OUT) :: ROCI        ! Soil thermal capacity [Jm-3K-1]

ETA_S = ERR_VE
PSIS = ERR_VE
CAPPAETA_S = ERR_VE
B = ERR_VE
ETA_WI = ERR_VE
ROCI = ERR_VE
SSR_FIE_CAPC = ERR_VE
IF (NUM < 1 .OR. NUM > NS) RETURN

ETA_S=ETA_S_VEC(NUM)
PSIS=PSIS_VEC(NUM)/100.             ! data in table are in cm
CAPPAETA_S=CAPPAETA_S_VEC(NUM)/100. ! data in table are in dm2/s
B=B_VEC(NUM)
ETA_WI=ETA_WI_VEC(NUM)
ROCI=ROCI_VEC(NUM)*1.E6             ! data in table are in uJm-3K-1
SSR_FIE_CAPC=(PSIS/PSIFC)**(1./B)           ! from Dingman (1994)

RETURN
END SUBROUTINE SELECT_SOIL

!*****************************************************************************
SUBROUTINE SETUP_NAMELIST_OUTPUT(FILENML,ACCESSO,DIRIN,DIROU,FORMATO,PREF,&
 ECO_CHOICE,ILOGALB,ILOGCOE,ILOGFUN,L360DAYS_CALENDAR,LDATA,LFREEZE,LUTCFLAG,&
 LTCAN_NEW,LVAP,INP_SOIL,KDATA,LVEG,MIN_FREEZE,MITER,NCODFILE,NDATI,&
 NUM_SOI_LAY,DATA_INI,ORA_INI,ALB_FH,ALAT,ALB_SD,ALON,ESSE,GAMMA,C_DREN,CLEG,D0_VEGPAR,&
 ROOT_DEP,EPSF,EPSG,VEG_HEI_STEP,HEI_SN,LAI_STEP,QUOTA,RESIST_F_MIN,LITUFA2,VEG_COV_STEP,
 TORTUOS,H_OBS_ALL,&
 H_OBS_WIND,INDERR,PTW)
!*****************************************************************************
! Does all the initial setup dealing with the reading of the configuration file.
! Implements the possibility of formatting that file as a namelist. Previously all
! this stuff was done in the main program. All setup procedures are moved in here.
!*****************************************************************************
! Calls: none
! - Author: M. Galli (01-Apr-2011)
```

```
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
  USE CONSTANTS
  USE FIELDS, &
! Utility renaming of some variables
   TEM_SOI_I_FLD => TEM_SOI_I, SR_SOI_I_FLD => SR_SOI_I, DEP_SOI_FLD        => DEP_SOI,&
   CLAY_FLD   => CLAY,   SAND_FLD   => SAND,   OM_FLD         => OM,&
   ETA_S1_FLD  => ETA_S1,  ETA_FC1_FLD => ETA_FC1, CAPPAETA_S1_FLD => CAPPAETA_S1,&
   BSOIL1_FLD => BSOIL1, ETA_WI1_FLD => ETA_WI1, ROC1_FLD       => ROC1,&
   NSO_FLD    => NSO

IMPLICIT NONE
CHARACTER(len=*),INTENT(IN)::FILENML
CHARACTER(len=*),INTENT(OUT)::ACCESSO
CHARACTER(len=100),INTENT(OUT)::DIRIN
CHARACTER(len=100),INTENT(OUT)::DIROU
CHARACTER(len=50),INTENT(OUT)::FORMATO
CHARACTER(len=90),INTENT(OUT)::PREF
LOGICAL,INTENT(OUT)::ECO_CHOICE
LOGICAL,INTENT(OUT)::ILOGALB
LOGICAL,INTENT(OUT)::ILOGCOE
LOGICAL,INTENT(OUT)::ILOGFUN
LOGICAL,INTENT(OUT)::L360DAYS_CALENDAR
LOGICAL,INTENT(OUT)::LDATA
LOGICAL,INTENT(OUT)::LFREEZE
LOGICAL,INTENT(OUT)::LUTCFLAG
LOGICAL,INTENT(OUT)::LTCAN_NEW
LOGICAL,INTENT(OUT)::LVAP
INTEGER,INTENT(OUT)::INDERR
INTEGER,INTENT(OUT)::INP_SOIL
INTEGER,INTENT(OUT)::KDATA
INTEGER,INTENT(OUT)::LVEG
INTEGER,INTENT(OUT)::MIN_FREEZE
INTEGER,INTENT(OUT)::MITER
INTEGER,INTENT(OUT)::NCODFILE
INTEGER,INTENT(OUT)::NDATI
INTEGER,INTENT(OUT)::NUM_SOI_LAY
INTEGER,DIMENSION(3),INTENT(OUT)::DATA_INI
INTEGER,DIMENSION(2),INTENT(OUT)::ORA_INI
INTEGER,DIMENSION(NSOILMAX)::NSO
REAL,INTENT(OUT)::ALB_FH
REAL,INTENT(INOUT)::ALAT
REAL,INTENT(OUT)::ALB_SD
REAL,INTENT(INOUT)::ALON
REAL,INTENT(OUT)::ESSE
REAL,INTENT(OUT)::GAMMA
REAL,INTENT(OUT)::C_DREN
REAL,INTENT(OUT)::CLEG
REAL,INTENT(OUT)::D0_VEGPAR
REAL,INTENT(OUT)::ROOT_DEP
REAL,INTENT(OUT)::EPSF
REAL,INTENT(OUT)::EPSG
REAL,INTENT(OUT)::VEG_HEI_STEP
REAL,INTENT(OUT)::HEI_SN
REAL,INTENT(OUT)::LAI_STEP
INTEGER,INTENT(OUT)::PTW
REAL,INTENT(OUT)::QUOTA
REAL,INTENT(OUT)::RESIST_F_MIN
REAL,INTENT(OUT)::LITUFA2
REAL,INTENT(OUT)::VEG_COV_STEP
REAL,INTENT(OUT)::TORTUOS
REAL,INTENT(OUT)::H_OBS_ALL
REAL,INTENT(OUT)::H_OBS_WIND
REAL,DIMENSION(NSOILMAX)::TEM_SOI_I
REAL,DIMENSION(NSOILMAX)::SR_SOI_I
REAL,DIMENSION(NSOILMAX)::DEP_SOI
REAL,DIMENSION(NSOILMAX)::CLAY
```

```
REAL,DIMENSION(NSOILMAX)::SAND
REAL,DIMENSION(NSOILMAX)::OM
REAL,DIMENSION(NSOILMAX)::ETA_S1
REAL,DIMENSION(NSOILMAX)::ETA_FC1
REAL,DIMENSION(NSOILMAX)::CAPPAETA_S1
REAL,DIMENSION(NSOILMAX)::BSOIL1
REAL,DIMENSION(NSOILMAX)::ETA_WI1
REAL,DIMENSION(NSOILMAX)::ROC1
INTEGER::I
!CHARACTER(len=200)::ioerror_message

!Characters
NAMELIST /utopia_config/ &
 DIRIN,DIROU,FORMATO,PREF
!Logicals
NAMELIST /utopia_config/ &
 ECO_CHOICE,ILOGALB,ILOGCOE,ILOGFUN,L360DAYS_CALENDAR,LDATA,LFREEZE,&
 LUTCFLAG,LTCAN_NEW,LVAP
!Integer scalars
NAMELIST /utopia_config/ &
 INP_SOIL,LVEG,KDATA,MIN_FREEZE,MITER,NCODFILE,NDATI,NUM_SOI_LAY,PTW
!Real scalars
NAMELIST /utopia_config/ &
 ALB_FH,ALAT,ALB_SD,ALON,ESSE,GAMMA,C_DREN,CLEG,D0_VEGPAR,ROOT_DEP,EPSF,EPSG,VEG_HEI_STEP,&
 HEI_SN,LAI_STEP,QUOTA,RESIST_F_MIN,LITUFA2,VEG_COV_STEP,TORTUOS,H_OBS_ALL,H_OBS_WIND
!Integer arrays
NAMELIST /utopia_config/ &
 DATA_INI,IC,ORA_INI,NSO
!Real arrays
NAMELIST /utopia_config/ &
 TEM_SOI_I,SR_SOI_I,DEP_SOI,CLAY,SAND,OM,ETA_S1,ETA_FC1,CAPPAETA_S1,BSOIL1,ETA_WI1,ROC1

!Setting default values for everything
ACCESSO=''
DIRIN='../input/'
DIROU='../output/'
FORMATO=''
PREF=''
ECO_CHOICE=.FALSE.
ILOGALB=.FALSE.
ILOGCOE=.TRUE.
ILOGFUN=.FALSE.
L360DAYS_CALENDAR=.FALSE.
LDATA=.FALSE.
LFREEZE=.FALSE.
LUTCFLAG=.TRUE.
LTCAN_NEW=.TRUE.
LVAP=.TRUE.
INP_SOIL=5
KDATA=1
LVEG=2
PTW=1
MIN_FREEZE=10080
MITER=120
NCODFILE=3
NDATI=10
NUM_SOI_LAY=-9999
DATA_INI=-9999
ORA_INI=-9999
ALB_FH=ERR_VE
ALAT=ERR_VE
ALB_SD=0.3
ALON=ERR_VE
ESSE=0.
GAMMA=0.
C_DREN=1.0
CLEG=0.
```

```
D0_VEGPAR=ERR_VE
ROOT_DEP=ERR_VE
EPSF=ERR_VE
EPSG=ERR_VE
VEG_HEI_STEP=ERR_VE
HEI_SN=0.
LAI_STEP=ERR_VE
QUOTA=0.
RESIST_F_MIN=ERR_VE
LITUFA2=0.
VEG_COV_STEP=ERR_VE
TORTUOS=1000.
H_OBS_ALL=2.
H_OBS_WIND=10.

! Allocation of fields just to read how many soil layers there are
! (READ might fail if namelist vectors unallocated)
CALL ALLOCATE_FIELDS(ALLOCATE_REALS,1,INDERR)
IF(INDERR/=0)THEN
 PRINT *,'FAILED initial allocation of reals'
 RETURN
END IF
CALL ALLOCATE_FIELDS(ALLOCATE_INTEGERS,1,INDERR)
IF(INDERR/=0)THEN
 PRINT *,'FAILED initial allocation of integers'
 RETURN
END IF

! Open namelist file
OPEN(UNIT=1,FILE=TRIM(FILENML),STATUS='OLD',ACTION='READ',IOSTAT=INDERR)
IF(INDERR/=0)THEN
PRINT *,'FAILED to open namelist file:'
PRINT *,TRIM(FILENML)
RETURN
END IF

! Read the namelist to determine NUM_SOI_LAY
READ(1,NML=utopia_config,IOSTAT=INDERR)!,IOMSG=ioerror_message)
IF(INDERR/=0)THEN
PRINT *,'ERROR occurred while preliminarly reading namelist file'
PRINT *,TRIM(FILENML)
PRINT *,'Possible causes:'
PRINT *,'*) Mismatch of a namelist variable'
!PRINT *,TRIM(ioerror_message)
RETURN
END IF

! Re position the file at the beginning
REWIND(UNIT=1)

! Deallocate space to then allocate it for the real size
CALL DEALLOCATE_FIELDS(DEALLOCATE_REALS,INDERR)
IF (INDERR/=0) THEN
 PRINT *,'FAILED initial deallocation of reals'
 RETURN
END IF
CALL DEALLOCATE_FIELDS(DEALLOCATE_INTEGERS,INDERR)
IF (INDERR/=0) THEN
 PRINT *,'FAILED initial deallocation of integers'
 RETURN
END IF

! Consistency check:
IF (NUM_SOI_LAY<1.OR.NUM_SOI_LAY>NSOILMAX) THEN
 PRINT *,'ERROR : wrong number of NUM_SOI_LAY selected'
 PRINT *,'NUM_SOI_LAY = ',NUM_SOI_LAY
 INDERR=1001
```

```
 RETURN
END IF

! Reallocation
CALL ALLOCATE_FIELDS(ALLOCATE_REALS,NUM_SOI_LAY,INDERR)
IF(INDERR/=0)THEN
PRINT *,'FAILED initial allocation of reals'
RETURN
END IF
CALL ALLOCATE_FIELDS(ALLOCATE_INTEGERS,NUM_SOI_LAY,INDERR)
IF(INDERR/=0)THEN
PRINT *,'FAILED initial allocation of integers'
RETURN
END IF
CALL ALLOCATE_FIELDS(ALLOCATE_LOGICALS,NUM_SOI_LAY,INDERR)
IF(INDERR/=0)THEN
PRINT *,'FAILED initial allocation of logicals'
RETURN
END IF

! Initialization of field arrays with default values
TEM_SOI_I=ERR_VE
SR_SOI_I=ERR_VE
DEP_SOI=ERR_VE
CLAY=ERR_VE
SAND=ERR_VE
OM=ERR_VE
ETA_S1=ERR_VE
ETA_FC1=ERR_VE
CAPPAETA_S1=ERR_VE
BSOIL1=ERR_VE
ETA_WI1=ERR_VE
ROC1=ERR_VE
NSO=-9999

! Real read
READ(1,NML=utopia_config,IOSTAT=INDERR)
IF(INDERR/=0)THEN
PRINT *,'ERROR occurred while reading namelist file'
PRINT *,TRIM(FILENML)
PRINT *,'Possible causes:'
PRINT *,'*) Mismatch of a namelist variable'
RETURN
END IF

! Namelist buffers to utopia fields
TEM_SOI_I_FLD=TEM_SOI_I(1:NUM_SOI_LAY)
SR_SOI_I_FLD=SR_SOI_I(1:NUM_SOI_LAY)
DEP_SOI_FLD=DEP_SOI(1:NUM_SOI_LAY)
CLAY_FLD=CLAY(1:NUM_SOI_LAY)
SAND_FLD=SAND(1:NUM_SOI_LAY)
OM_FLD=OM(1:NUM_SOI_LAY)
ETA_S1_FLD=ETA_S1(1:NUM_SOI_LAY)
ETA_FC1_FLD=ETA_FC1(1:NUM_SOI_LAY)
CAPPAETA_S1_FLD=CAPPAETA_S1(1:NUM_SOI_LAY)
BSOIL1_FLD=BSOIL1(1:NUM_SOI_LAY)
ETA_WI1_FLD=ETA_WI1(1:NUM_SOI_LAY)
ROC1_FLD=ROC1(1:NUM_SOI_LAY)
NSO_FLD=NSO(1:NUM_SOI_LAY)

! Consistency checks and so on
IF(LEN_TRIM(PREF)==0)THEN
PRINT *,'No PREF specified.'
INDERR=1002
RETURN
END IF
```

```
IF (NCODFILE < 0)THEN  !file coding
 FORMATO='UNFORMATTED'
 ACCESSO='DIRECT'
ELSE
 FORMATO='FORMATTED'
 ACCESSO='SEQUENTIAL'
END IF

! check about consistency of KDATA and MIN_FREEZE
IF (MIN_FREEZE > 0) THEN
 IF (MOD(MIN_FREEZE,(60/KDATA)) /= 0) THEN
  WRITE(90,*) 'MIN_FREEZE=',MIN_FREEZE,' inconsistent with KDATA=',KDATA,&
   ' and set to ZERO'
  MIN_FREEZE=0
 ENDIF
ENDIF

IF (ANY(TEM_SOI_I_FLD<-DTK)) THEN
 PRINT *,'ERROR : invalid initial soil temperature TEM_SOI_I'
 PRINT *,'TEM_SOI_I = ',TEM_SOI_I_FLD
 INDERR=1003
 RETURN
END IF

IF (ANY(SR_SOI_I_FLD<0.)) THEN
 PRINT *,'ERROR : invalid initial soil moisture SR_SOI_I'
 PRINT *,'SR_SOI_I = ',SR_SOI_I_FLD
 INDERR=1004
 RETURN
END IF

IF (ANY(SR_SOI_I_FLD>1.)) THEN
 PRINT *,'ERROR : invalid initial soil moisture SR_SOI_I'
 PRINT *,'SR_SOI_I = ',SR_SOI_I_FLD
 INDERR=1005
 RETURN
END IF

IF (ANY(DEP_SOI_FLD<0.)) THEN
 PRINT *,'ERROR : invalid initial soil moisture SR_SOI_I'
 PRINT *,'DEP_SOI = ',DEP_SOI_FLD
 INDERR=1006
 RETURN
END IF

! Eventual conversion of temperature from Celsius to Kelvin
IF (ANY(TEM_SOI_I_FLD<100.)) THEN
 TEM_SOI_I_FLD=TEM_SOI_I_FLD+DTK
END IF

! Check about PTW
IF (PTW/=1 .AND. PTW/=0) THEN
 PRINT *,"ERROR : invalid photosynthetic pathway"
 INDERR=1007
 RETURN
END IF

! Only Lveg=1,2,7,8,11,19 can have PTW=0 (C4 pathway)
IF (PTW==0 .AND. LVEG/=1 .AND. LVEG/=2 .AND. LVEG/=7 .AND. LVEG/=8 &
 .AND. LVEG/=11 .AND. LVEG/=19) THEN
 PRINT *,"Error : invalid photosynthetic pathway"
 INDERR=1008
 RETURN
END IF

! If desert or semi-desert the pathway is changed in C4
IF (LVEG==8 .OR. LVEG==11) THEN
```

```
 PTW=0
END IF

CLOSE(UNIT=1)

END SUBROUTINE SETUP_NAMELIST_OUTPUT

!*****************************************************************************
SUBROUTINE SAXTON(CLAY,SAND,SILT,OM,ETA_S1,ETA_FC1,CAPPAETA_S1,BSOIL1,ETA_WI1,&
 ROC1,B,CAPPAETA_S,ETA_S,PSIS,ETA_WI,SSR_FIE_CAPC,ROCI,CATEGORY,DF,RW)
!*****************************************************************************
! Empirical equations to evaluate BSOIL,CAPPAETA_S,ETA_S,PSIS,ETA_WI,SSR_FIE_CAPC,ROCI on the
! basis of soil texture and percentage of organic matter. Those relations are taken
! from: Saxton, K.E.; Rawls, W.J. Soil Water Charateristic Estimates by Texture and
! Organic Matter for Hydrologic Solution; Soil Sci. Soc. Am. J.; Vol. 70, pp
! 1569-1578; 2006
!*****************************************************************************
! Calls: none
! - Author: R. Bonanno (13-Apr-2010)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
USE SOIL_VALUES

IMPLICIT NONE
REAL :: ALPHA
REAL,INTENT(OUT) :: B
REAL,INTENT(IN) :: BSOIL1
REAL,INTENT(OUT) :: CAPPAETA_S
REAL,INTENT(IN) :: CAPPAETA_S1
CHARACTER(len=30) :: CATEGORY
REAL,INTENT(INOUT) :: CLAY
REAL :: CLAY1
REAL :: DELTA
REAL :: DEN_B
REAL :: DEN_DF
REAL :: DEN_N
REAL,INTENT(IN) :: DF ! varies between 0.9 and 1.3, default 1.0
REAL :: ETA_1500
REAL :: ETA_1500T
REAL :: ETA_33
REAL :: ETA_33_DF
REAL :: ETA_33T
REAL :: ETA_S_33_DF
REAL :: ETA_S_DF
REAL :: ETA_S33
REAL :: ETA_S33T
REAL :: ETA_FC
REAL,INTENT(IN) :: ETA_FC1
REAL,INTENT(OUT) :: ETA_S
REAL,INTENT(IN) :: ETA_S1
REAL,INTENT(OUT) :: ETA_WI
REAL,INTENT(IN) :: ETA_WI1
INTEGER :: I
REAL :: KB
REAL :: LAMBDA
INTEGER :: N
REAL,INTENT(INOUT) :: OM ! varies between 0.0 and 7.5, default 0.0
REAL :: OM1
REAL :: PAW
REAL :: PAWB
REAL :: PSI_S
REAL,INTENT(OUT) :: PSIS
REAL,DIMENSION(3) :: R
REAL,INTENT(IN) :: ROC1
REAL,INTENT(OUT) :: ROCI
REAL :: RV
```

```
REAL,INTENT(IN) :: RW ! varies between 0 and 0.7 g/g, default 0.0
REAL,INTENT(INOUT) :: SAND
REAL :: SAND1
REAL,INTENT(INOUT) :: SILT
REAL :: SILT1
REAL,INTENT(OUT) :: SSR_FIE_CAPC

! Saxton's relations do not work with CLAY, SAND or SILT percentages equal to
! zero. The minimum required value is 2%. In this part of the code we set to
! 2% the null values of percentages and readjust the values of the other
! soil components so that the sum of the CLAY,SAND,SILT percentages is
! conserved
! Empirical equations need the value of clay, sand and silt in units (0-1). Since
! these values are in percentage, they are divided by 100.
! Note: CLAY1, SAND1, SILT1 and OM1 are fraction values (0-1). CLAY, SAND and SILT are
! instead percentages. Since CLAY1+SAND1+SILT1+OM1=1 and CLAY+SAND+SILT=100, it is not
! simply CLAY1=CLAY/100, as CLAY, SAND and SILT are renormalized not considering the
! organic matter

CLAY1 = CLAY / 100.
SAND1 = SAND / 100.
SILT1 = SILT / 100.
OM1 = OM

R(1)=CLAY
R(2)=SAND
R(3)=SILT

DELTA=0.

DO I=1,3
 IF (R(I)<2.) THEN
  DELTA=DELTA+(2.-R(I))
  R(I)=2.
 END IF
END DO

N=0

DO I=1,3
 IF (R(I)>2.) THEN
  N=N+1
 END IF
END DO

DO I=1,3
 IF (R(I)>2.) THEN
  R(I)=R(I)-DELTA/N
 END IF
END DO

CLAY=R(1)
SAND=R(2)
SILT=R(3)

! Determination of dry soil termal capacity: on the basis of the value (%) of
! sand silt and clay, the soil category is determined on the basis of soil
! texture classification triangle using the internal database, as this method do not
! gives soil heat capacity.

CATEGORY='null'

IF (ROC1/=ERR_VE) THEN
 ROCI=ROC1*1.E6
ELSE

 IF ((0.<=SAND .AND. 0.<=CLAY)) THEN
```

```
 IF ( (80.<=SILT) .AND. (CLAY<12.) )THEN
  CATEGORY='SILT'
  ROCI=2.133
! silt heat capacity is not present in the internal database; it is evaluated
! according with referenced values

 ELSE IF ((50.<=SILT .AND. (12.<=CLAY .AND. CLAY<=27)) .OR. ((50<=SILT .AND. SILT<=80.) .AND.
 CLAY<12.)) THEN
  CATEGORY='SILT LOAM'
  ROCI=ROCI_VEC(4)

 ELSE IF ((85.<=SAND) .AND. (SILT+1.5*CLAY<15.)) THEN
  CATEGORY='SAND'
  ROCI=ROCI_VEC(1)

 ELSE IF ((70.<=SAND .AND. SAND<=91.) .AND. (15.<=SILT+1.5*CLAY) .AND. (SILT+2*CLAY<30) ) THEN
  CATEGORY='LOAMY SAND'
  ROCI=ROCI_VEC(2)

 ELSE IF (((7.<=CLAY .AND. CLAY<=20.) .AND. (52.<=SAND) .AND. (30.<=(SILT+2.*CLAY))) .OR. ((
 CLAY<7.) .AND. (SILT<50.) .AND. (43.<=SAND))) THEN
   CATEGORY='SANDY LOAM'
   ROCI=ROCI_VEC(3)

 ELSE IF ((7<=CLAY .AND. CLAY<=27.) .AND. (28.<=SILT .AND. SILT<=50.) .AND. (SAND<=52.)) THEN
  CATEGORY='LOAM'
  ROCI=ROCI_VEC(5)

 ELSE IF ((20.<=CLAY .AND. CLAY<=35.) .AND. (SILT<28.) .and. (45.<=SAND)) THEN
   CATEGORY='SANDY CLAY LOAM'
   ROCI=ROCI_VEC(6)

 ELSE IF ((35.<=CLAY) .AND. (45.<=SAND)) THEN
   CATEGORY= 'SANDY CLAY'
   ROCI=ROCI_VEC(9)

 ELSE IF ((27.<=CLAY .AND. CLAY<=40.) .AND. (SAND<=20.)) THEN
   CATEGORY='SILTY CLAY LOAM'
   ROCI=ROCI_VEC(7)

 ELSE IF ((27.<=CLAY .AND. CLAY<=40.) .AND. (20.<=SAND .AND. SAND <=46.)) THEN
   CATEGORY='CLAY LOAM'
   ROCI=ROCI_VEC(8)

 ELSE IF (40.<=CLAY .AND. 40.<=SILT ) THEN
   CATEGORY='SILTY CLAY'
   ROCI=ROCI_VEC(10)

 ELSE IF ((40.<=CLAY) .AND. (SAND<=45.) .AND. (SILT<40.) ) THEN
   CATEGORY='CLAY'
   ROCI=ROCI_VEC(11)
 END IF
 END IF

! Converting ROCI from uJm-3K-1 in Jm-3K-1
 ROCI=ROCI*1.E6          ! data in table are in uJm-3K-1
END IF


! Empirical equation to evaluate the values of BSOIL,CAPPAETA_S,ETA_S,PSIS,ETA_WI

ETA_33T = -0.251*SAND1 + 0.195*CLAY1 + 0.011*OM1 + 0.006*(SAND1*OM1) &
 -0.027*(CLAY1*OM1) + 0.452*(SAND1*CLAY1) + 0.299

! Volumetric water content at field capacity (33 KPa), calculated only if absent
IF (ETA_FC1/=ERR_VE) THEN
```

```
 ETA_33 = ETA_FC1
ELSE
 !Volumetric water content at field capacity (33 KPa)
 ETA_33 = 1.283*(ETA_33T)**2 + 0.626*ETA_33T - 0.015 ! sjlim
END IF

ETA_1500T = -0.024*SAND1 + 0.487*CLAY1 + 0.006*OM1 + 0.005*(SAND1*OM1) &
 - 0.013*(CLAY1*OM1) + 0.068*(SAND1*CLAY1) + 0.031

! Volumetric water content at wilting point (1500 KPa), calculated only if absent
IF (ETA_WI1/=ERR_VE) THEN
 ETA_1500=ETA_WI1
ELSE
 ETA_1500 = 1.14*ETA_1500T - 0.02
END IF

! difference among ETA_S and ETA_33 (not used)
ETA_S33T = 0.278*SAND1 + 0.034*CLAY1 + 0.022*OM1 - 0.018*(SAND1*OM1) &
 - 0.027*(CLAY1*OM1) - 0.584*(SAND1*CLAY1) + 0.078
ETA_S33 = 1.636*ETA_S33T - 0.107 ! sjlim(?)

! B coefficient
IF(BSOIL1/=ERR_VE)THEN
 B = BSOIL1
ELSE
 B = (LOG(1500.0)-LOG(33.0)) / (LOG(ETA_33)-LOG(ETA_1500))
END IF
LAMBDA = 1./B

! Soil porosity
IF (ETA_S1/=ERR_VE) THEN
 ETA_S = ETA_S1
ELSE
 ETA_S = ETA_33 + ETA_S33 - 0.097*SAND1 + 0.043
END IF

DEN_N = (1. - ETA_S) * 2.65 ! normal matric soil density (g/cm3) - 2.65 g/cm3 is particle
density

! density effect
IF (DF /= 1.) THEN

 DEN_DF = DEN_N * DF ! adjusted density (g/m3)

! Adjusted saturated volumetric water content (in place of ETA_S)
 ETA_S_DF = 1. - DEN_DF / 2.65

! Adjusted Volumetric water content at field capacity (33 KPa) (in place of ETA_33)
 ETA_33_DF = ETA_33 - 0.2 * (ETA_S - ETA_S_DF)

! Adjusted SAT-33 kPa moisture (in place of ETA_S33 - not used)
 ETA_S_33_DF = ETA_S_DF - ETA_33_DF

 ETA_S = ETA_S_DF
 ETA_33 = ETA_33_DF
 ETA_S33 = ETA_S_33_DF

ENDIF

KB = 1.
PAW = ETA_33 - ETA_WI ! plant available soil moisture (33-1500 matric soil) %v
ETA_WI = ETA_1500 ! Volumetric water content at wilting point (1500 KPa)
ETA_FC = ETA_33 ! Volumetric water content at field capacity (33 KPa)

! gravel effect (bulk = matric + gravel)
IF (RW > 0.) THEN
```

```
 ALPHA = DEN_N/2.65 ! g/cm3
 RV = ALPHA * RW / ( 1. - RW * ( 1. - ALPHA) ) ! volume fraction of gravel, g/cm3
 DEN_B = DEN_N * (1. - RV) + (RV * 2.65) ! bulk soil density (matric plus gravel)
 PAWB = PAW * (1. - RV) ! plant available soil moisture (33-1500 bulk soil) %v
 KB = (1. - RW) / ( 1. - RW * ( 1. - 1.5*ALPHA )  ) ! factor to multiply CAPPAETAS

ENDIF

! Saturated moisture potential - compared with paper, there is division by porosity at B
PSI_S = EXP ( LOG(33.0) + B * LOG(ETA_33) ) / (ETA_S**B) ! kPa
PSI_S = -PSI_S / GRAVITY ! from kPa to m: divide by gravity, multiply by 1000 (kPa -> Pa) and
divide by 1000. (water density)

! Soil saturated hydraulic conductivity
IF (CAPPAETA_S1/=ERR_VE) THEN
 CAPPAETA_S = CAPPAETA_S1 / 100. ! last operation converts from cm/s to m/s
ELSE
 CAPPAETA_S = ( 1930.*(ETA_S-ETA_33)**(3. - LAMBDA) ) / 3.6E6
! last operation converts from mm/h to m/s
 IF (KB /= 1.) CAPPAETA_S = CAPPAETA_S * KB ! correction for gravel effects
END IF

! Saturated water moisture potential (obtained by inverting the formula for field
! capacity in function of saturated water potential)
PSIS = ( ( ETA_FC / ETA_S)**B ) * (-3.4)

! Evaluation of SSR_FIE_CAPC
SSR_FIE_CAPC = MIN(1., (ETA_FC/ETA_S) )

END SUBROUTINE SAXTON
!****************************************************************************
SUBROUTINE SHORTW(ALB_F,ALB_G,ALB_SN,RAD_S_D,RAD_S_F_D,RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_F_U,
RAD_S_G_D,&
 RAD_S_SN_G_D,RAD_S_SN_G_U,RAD_S_G_U,RAD_S_SN_D,RAD_S_SN_U,RAD_S_U,VEG_COV_STEP,COVER_SN_F,
 COVER_SN_G)
!****************************************************************************
! Evaluates the balance terms for shortwave radiation (all expressed in W/m2)
! following Mutinelli (1998) chapter 4.
!****************************************************************************
! Calls: none
! - Authors: C. Cassardo, J.J. Ji, F. Mutinelli (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
IMPLICIT NONE

REAL,INTENT(IN) :: ALB_F
REAL,INTENT(IN) :: ALB_G
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: ALB_SN
REAL,INTENT(OUT) :: RAD_S_F_D
REAL,INTENT(OUT) :: RAD_S_SN_F_D
REAL,INTENT(OUT) :: RAD_S_SN_F_U
REAL,INTENT(OUT) :: RAD_S_F_U
REAL,INTENT(OUT) :: RAD_S_G_D
REAL,INTENT(OUT) :: RAD_S_SN_G_D
REAL,INTENT(OUT) :: RAD_S_SN_G_U
REAL,INTENT(OUT) :: RAD_S_G_U
REAL,INTENT(OUT) :: RAD_S_SN_D
REAL,INTENT(OUT) :: RAD_S_SN_U
REAL,INTENT(OUT) :: RAD_S_U
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN_F
REAL,INTENT(IN) :: COVER_SN_G

RAD_S_SN_F_D=VEG_COV_STEP*COVER_SN_F*RAD_S_D    ! dwnwd rad over snow-covered vegetation
RAD_S_SN_F_U=RAD_S_SN_F_D*ALB_SN                  ! upward rad from vegetation covered by
snow
```

```
RAD_S_F_D=VEG_COV_STEP*(1.-COVER_SN_F)*RAD_S_D  ! downward radiation over vegetation
RAD_S_F_U=RAD_S_F_D*ALB_F                       ! upward radiation from vegetation
!RSLBD=RAD_S_F_D*EXP(CAPPA*LAI_STEP/COSD(SOL_ANG) ! k=0.5 Wenge Ni-Meister and Huilin Gas 2011
RAD_S_SN_G_D=(1.-VEG_COV_STEP)*COVER_SN_G*RAD_S_D       ! dwnw rad over bare soil covered by
snow
RAD_S_SN_G_U=RAD_S_SN_G_D*ALB_SN                    ! upward radiation from bare soil covered
by snow
RAD_S_G_D=(1.-VEG_COV_STEP)*(1.-COVER_SN_G)*RAD_S_D ! downward radiation over bare soil
RAD_S_G_U=RAD_S_G_D*ALB_G                    ! upward radiation from bare soil
RAD_S_SN_D=RAD_S_SN_F_D+RAD_S_SN_G_D                    ! downward radiation over snow
RAD_S_SN_U=RAD_S_SN_F_U+RAD_S_SN_G_U                    ! upward radiation from snow
RAD_S_U=RAD_S_G_U+RAD_S_F_U+RAD_S_SN_U                  ! upward total radiation

RETURN
END SUBROUTINE SHORTW

!****************************************************************************
SUBROUTINE SNOW(ALB_SN,TEMP_A_STEP,DENS_SN,TIMESTEP_SEC,EVA_SN,LHF_SN,SHF_SN,HM_SN,HEI_SN,
HW_SN,&
 LSNOW,PREC_ON_SNOW,P_HM_SN,P_HW_SN,PRECSN,P_TEMP_SN,BAL_FLU_F,BAL_FLU_G,CONF_SN_SRF,NR_SN,
 RUNSN,&
 VEG_COV_STEP,TEMP_SN,TWET,BAL_FLU_SN)
!****************************************************************************
! Adapted from Dingman (1994) chapter 5, following Mutinelli (1998), chapter 4.
!****************************************************************************
! Calls: none
! - Authors: C. Cassardo, F. Mutinelli (01-Apr-1998)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(INOUT) :: ALB_SN
REAL,INTENT(IN) :: TEMP_A_STEP
! following parameter values [cm/hr]: 2.778 (Sun et al.);
! 2.7E-4 (corresponding to 0.01 h/cm, Anderson);
! 7.2E-4 (corresponding to 0.026 h/cm, Kojima)
!REAL,PARAMETER ::  C1=2.7778E-7 ! m2 / (s kg)
! cntrl: c1=3E-5   c2=1.7   c3=15 c4=1.5 c5=0.08 c6=0.021
! exp01: c1=3E-5   c2=1.7   c3=15 c4=1.5 c5=0.32 c6=0.021
! exp02: c1=3E-5   c2=0.029 c3=15 c4=3.0 c5=0.08 c6=0.021
! exp03: c1=12E-5  c2=1.7   c3=15 c4=1.5 c5=0.08 c6=0.021
! exp04: c1=3E-5   c2=1.7   c3=30 c4=1.5 c5=0.08 c6=0.021
! exp05: c1=3E-5   c2=1.7   c3=15 c4=6.0 c5=0.08 c6=0.021
! exp06: c1=3E-5   c2=6.8   c3=15 c4=1.5 c5=0.08 c6=0.021
! exp07: c1=3E-5   c2=1.7   c3=15 c4=1.5 c5=0.08 c6=0.084
! exp08: c1=7.5E-6 c2=1.7   c3=15 c4=1.5 c5=0.08 c6=0.021
! exp09: c1=3E-5   c2=0.4   c3=15 c4=1.5 c5=0.08 c6=0.021
! exp10: c1=3E-5   c2=1.7   c3=4  c4=1.5 c5=0.08 c6=0.021
! exp11: c1=3E-5   c2=1.7   c3=15 c4=0.4 c5=0.08 c6=0.021
! exp12: c1=3E-5   c2=1.7   c3=15 c4=1.5 c5=0.02 c6=0.021
! exp13: c1=3E-5   c2=1.7   c3=15 c4=1.5 c5=0.08 c6=0.005
! exp14: c1=3E-5   c2=0.029 c3=15 c4=3.0 c5=0.16 c6=0.021
! exp15: c1=3E-5   c2=0.4   c3=15 c4=1.5 c5=0.08 c6=0.005
! exp16: c1=3E-5   c2=0.4   c3=15 c4=1.5 c5=0.08 c6=0.010
! exp17: c1=3E-5   c2=0.4   c3=15 c4=1.5 c5=0.08 c6=0.015
REAL,PARAMETER ::  C1=3.E-5 ! m2 / (s kg)
REAL,PARAMETER ::  C2=0.4
REAL,PARAMETER ::  C3=15.
REAL,PARAMETER ::  C4=1.5
REAL,PARAMETER ::  C5=0.08 !
REAL,PARAMETER ::  C6=0.015 !
REAL,PARAMETER ::  CC1=2.778E6 ! [s-1] Sun et al. (1999) pag. 19589 eq. 8
REAL,PARAMETER ::  CC2=0.04 ! [K-1] Sun et al. (1999) pag. 19589 eq. 8
REAL,PARAMETER ::  CCK1=2.778E-3 ! [m-1 s-1] Koren et al (1999) p 19571-2 eq 8
REAL,PARAMETER ::  CCK2=2.1E-2 ! [m3 kg-1] Koren et al (1999) p 19571-2 eq 8
REAL :: CC3 ! Sun et al. (1999) eq. 8
```

```
REAL :: CC4 ! Sun et al. (1999) eq. 8
REAL,PARAMETER ::  CC5=0.08 ! [K-1] Sun et al. (1999) pag. 19589 eq. 12
REAL,PARAMETER ::  CC6=0.021 ! [m3 kg-1] Sun et al. (1999) pag. 19589 eq. 12
REAL,PARAMETER :: CMAX=0.10
REAL,PARAMETER :: CMIN=0.00
REAL :: CR1                 ! snow compaction rate [1/m]
REAL :: CR2
REAL,INTENT(INOUT) :: DENS_SN   ! snow density at time T+DT [kg/m3]
REAL :: DENS_SN_0
REAL :: DENS_SN_OLD
REAL :: DENS_SN_NEW
REAL :: DH
REAL :: DQ                  ! energy available after snow warming at DTK [J/m2]
REAL,INTENT(IN) :: BAL_FLU_SN
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_SN
REAL,PARAMETER ::  ETA0= 3.6E6 ! [N s m-2] Sun et al. (1999) pag. 19589 eq. 11
REAL :: ETAA
REAL,INTENT(IN) :: LHF_SN       ! latent heat flux in the atmosphere [W/m2]
REAL :: GAMIC
REAL :: GAMWA
REAL,INTENT(IN) :: SHF_SN       ! sensibile heat flux in the atmosphere [W/m2]
REAL,INTENT(INOUT) :: HM_SN ! water equivalent [m]
REAL,INTENT(INOUT) :: HEI_SN    ! snow depth [m]
REAL,INTENT(INOUT) :: HW_SN ! water into snow [m]
REAL :: HW_SN_RET               ! liquid-water-retaining capacity [m]
LOGICAL,INTENT(INOUT) :: LSNOW
REAL,INTENT(IN) :: PREC_ON_SNOW
REAL,INTENT(INOUT) :: P_HM_SN   ! previous water equivalent [m]
REAL,INTENT(INOUT) :: P_HW_SN   ! previous water into snow [m]
REAL,INTENT(IN) :: PRECSN   ! precipitating snow [m/s]
REAL :: PS
REAL :: PSELF
REAL :: PTOP
REAL,INTENT(INOUT) :: P_TEMP_SN ! current snow temperature [K]
REAL :: QAV                 ! available energy during DT [J/m2]
REAL,INTENT(INOUT) :: BAL_FLU_F
REAL,INTENT(INOUT) :: BAL_FLU_G
REAL :: QMELTTOT            ! energy necessary to melt all snow [J/m2]
REAL :: QRAIN               ! heat by rain [W/m2]
REAL :: QRIPE               ! energy necessary to saturate snow with melt snow [J/m2]
REAL :: QRISC               ! energy to warm snow at DTK [J/m2]
REAL :: QSCALD
REAL,INTENT(IN) :: CONF_SN_SRF  ! conductive heat flux versus the ground [W/m2]
REAL :: QSOLID
REAL :: RHOD
REAL :: RHOD0
REAL,INTENT(IN) :: NR_SN        ! net radiation [W/m2]
REAL,INTENT(INOUT) :: RUNSN ! runoff [m]
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,PARAMETER :: TAU1=86400. ! [s] Douville et al. (1995) p 23 eq 13
REAL,PARAMETER :: TAUF=0.24 ! [] Douville et al. (1995) - expon. e-folding time of 4 days p
23 eq 13
REAL :: THETARET            ! maximum water retention capacity of snow [-]
REAL,INTENT(INOUT) :: TEMP_SN   ! new snow temperature [K]
REAL,INTENT(IN) :: TWET ! wet bulb temperature [K]
REAL :: WEIGHTMEAN
SAVE RHOD

IF (DENS_SN==ERR_VE) THEN
 RHOD=DENS_SN_MIN
 DENS_SN_OLD=DENS_SN_MIN
ELSE
 RHOD=DENS_SN
 DENS_SN_OLD=DENS_SN
ENDIF
RHOD0=RHOD
```

```
! Calculation of heat flux coming from rain on snow (if one)
! Rain over snow is considered to have wet bulb temperature (Ujihashi pag 157)
! If rain over snow at snow temp < 0 C: first water over snow will cold to 0^C and
! releases sensible heat flux, then can freeze releasing fusion latent heat flux
! (Dingman pag. 192)
IF (P_TEMP_SN<DTK) THEN
 QRAIN=PREC_ON_SNOW*DENS_WAT*CSW*(TWET-DTK)+DENS_WAT*CLF*PREC_ON_SNOW ! W/m2
ELSE
 QRAIN=PREC_ON_SNOW*DENS_WAT*CSW*(TWET-P_TEMP_SN)
END IF
!??questa formula non è giusta ??

IF (DENS_SN <= 0.) THEN !to avoid initial case of snowfall in which DENS_SN is not defined
 DENS_SN_0=DENS_SN_MIN
ELSE
 DENS_SN_0=DENS_SN
ENDIF

! Snow retention capacity
! 1) Mutinelli formulation (following Dingman and Eagleson)
! 2nd coefficient in HW_SN_RET devided for 1000 because DENS_SN is in kg/m3 not in g/cm3
! The limit value 0.0069825 is the minimum value of () below which () become negative
!HW_SN_RET=P_HM_SN*MAX(0.0069825,(-0.0735+2.67E-4*DENS_SN)) ! m
!
! 2) Dingman formulation
!THETARET=-0.0735*DENS_SN/DENS_WAT+0.267E-3*DENS_SN*DENS_SN/DENS_WAT
!THETARET=MAX(0.00001,THETARET)         ! to avoid negative values
!HW_SN_RET=P_HM_SN*THETARET
!
! 3) Ujihashi formulation:
!IF(DENS_SN<=400.) THEN
! HW_SN_RET=0.136*EXP(-3.05*DENS_SN/1000.)
!ELSE
! HW_SN_RET=0.111*DENS_SN/1000.
!ENDIF
!
! 4) Dingman (same as Mutinelli), for densities larger than 275.28 kg/m3.
! For smaller ones it can be used Loth and Graf (1993) formulation
!ROE=200.
!CMIN=0.03
!IF(DENS_SN>=500.) THEN
! THETARET=-0.0735*DENS_SN/DENS_WAT+0.267E-3*DENS_SN*DENS_SN/DENS_WAT
! THETARET=MAX(EPSIL,THETARET)
!ELSE IF(DENS_SN>=ROE.AND.DENS_SN<500.) THEN
! THETARET=CMIN
!ELSE
! THETARET=CMIN+(CMAX-CMIN)*(ROE-DENS_SN)/ROE
!ENDIF
!HW_SN_RET=P_HM_SN*THETARET
!
! 5) mixture of Mutinelli (Dingman) and Loth and Graf (1993) formulation
IF(DENS_SN_0>=275.28) THEN         !variante alla alt 3
 THETARET=-0.0735*DENS_SN_0/DENS_WAT+0.267E-3*DENS_SN_0*DENS_SN_0/DENS_WAT
 THETARET=MAX(EPSIL,THETARET)
ELSE
 THETARET=CMIN+(CMAX-CMIN)*(275.28-DENS_SN_0)/275.28
 THETARET=MAX(EPSIL,THETARET)
ENDIF
HW_SN_RET=P_HM_SN*THETARET

! The Energy Balance

! QAV = available energy;
QAV=BAL_FLU_SN*TIMESTEP_SEC  ! (sign + = snow warming) J/m2

IF(P_TEMP_SN>DTK) THEN
```

```
 P_TEMP_SN=DTK ! it should never happen P_TEMP_SN>DTK, but in the case...
END IF

! QRISC = necessary energy to warm snow at DTK
QRISC=DENS_WAT*CIICE*P_HM_SN*(DTK-P_TEMP_SN) ! J/m2

! DQ = energy available after the warning of snow at DTK
DQ=QAV-QRISC ! J/m2

! QMELTTOT = energy necessary to melt all snow
QMELTTOT=P_HM_SN*DENS_WAT*CLF ! J/m2

! QRIPE = energy to saturate the snow with melt snow = energy to melt the maximum
! water retention capacity - liquid water yet present
QRIPE=(HW_SN_RET-P_HW_SN)*DENS_WAT*CLF ! J/m2

! QSOLID= energy necessary to refreeze all ripe snow  !
QSOLID=P_HW_SN*DENS_WAT*CLF ! J/m2  ! energy necessary to melt snow equivalent of P_HW_SN
! meters of water

IF (P_HM_SN>0.) THEN
! Case A: there is old snow (P_HM_SN>0)

 IF (QAV<=QRISC.AND.QAV>0.) THEN
! Case A1: QAV>0 but QAV<QRISC: just warming: TEMP_SN increases at <= DTK
  HM_SN=P_HM_SN ! m    snow doesn't melt
  HW_SN=P_HW_SN ! m
  TEMP_SN=P_TEMP_SN+QAV/(DENS_WAT*CIICE*P_HM_SN) ! K

 ELSE IF (QAV>QRISC.AND.QAV>0.) THEN
! Case A2: QAV>0 and QAV>QRISC: warming TEMP_SN = DTK and some snow can ripe or melt
  TEMP_SN=DTK ! K
  IF (DQ>=QMELTTOT) THEN
! Case A2a: DQ>QMELTTOT: all snow melts and disappears and QSCALD is the remaining
! energy, added to canopy and soil energy balance; runoff water infiltrates
   QSCALD=DQ-QMELTTOT ! J/m2
   BAL_FLU_F=BAL_FLU_F+QSCALD/TIMESTEP_SEC*VEG_COV_STEP ! W/m2
   BAL_FLU_G=BAL_FLU_G+QSCALD/TIMESTEP_SEC*(1.-VEG_COV_STEP) ! W/m2
   HM_SN=0. ! m
   HW_SN=0. ! m

  ELSE IF (DQ>=QRIPE.AND.DQ<QMELTTOT) THEN
! Case A2b: DQ>QRIPE but DQ<QMELTTOT: all snow will be ripe (at HW_SN_RET) but only
! part of the snow will melt
   HM_SN=P_HM_SN-(DQ-QRIPE)/(DENS_WAT*CLF) ! m
   HW_SN=HW_SN_RET ! We need to adjust also HW_SN

  ELSE IF (DQ<QRIPE.AND.DQ>0.) THEN
! Case A2c: DQ<QRIPE: part of the snow will be ripe (less than HW_SN_RET), no melt
   HW_SN=P_HW_SN+DQ/(DENS_WAT*CLF) ! m
   HM_SN=P_HM_SN     ! m - does not vary

  ELSE
   PRINT *,'DQ = ',DQ
   STOP 'Impossible option in routine SNOW, case A2'
  END IF

 ELSE IF (QAV<=0.) THEN
! Case A3: QAV<0: snow cools

  IF (P_HW_SN>0.)THEN

! Case A3a: there is saturated (ripe) snow: the melt ripe snow can refreeze

   IF (-QAV>QSOLID) THEN
! Case A3a1: -QAV>QSOLID: the ripe snow freezes and TEMP_SN decreases by DQAV
    HW_SN=0. ! m  all water retained in snow freezes and remain in the snow volume
```

```
    HM_SN=P_HM_SN ! m
    TEMP_SN=P_TEMP_SN+(QAV+QSOLID)/(DENS_WAT*CIICE*P_HM_SN) ! K

   ELSE
! Case A3a2: -QAV<QSOLID> only a part of ripe snow refreezes
    HW_SN=P_HW_SN+QAV/(DENS_WAT*CLF) ! m
    HM_SN=P_HM_SN ! m
    TEMP_SN=P_TEMP_SN   ! (=DTK, in K)

   END IF

  ELSE IF (P_HW_SN <= 0.) THEN

! Case A3b: QAV<0 but HW_SN=0: no ripe (saturated) snow: just cooling of all snow
   HM_SN=P_HM_SN ! m
   HW_SN=P_HW_SN ! m  ( = 0)
   TEMP_SN=P_TEMP_SN+QAV/(DENS_WAT*CIICE*P_HM_SN) ! K

  ELSE
   PRINT *,'QAV = ',QAV
   STOP 'Impossible option in routine SNOW, case A3'
  END IF

 END IF

! Snow density
! Preliminary check about current snow density
 IF (DENS_SN<=0.) THEN
  PRINT *,'DENS_SN=ERR_VE but P_HM_SN>0.: it should never occur!!!'
  PRINT *,'DENS_SN, P_HM_SN = ',DENS_SN,P_HM_SN
  PRINT *,'Snow density re-initialized to ',DENS_SN_MIN
  DENS_SN=DENS_SN_MIN
 ENDIF

!---1) METEOFRANCE (Douville et al., 1995)
! DENS_SN=(MAX(DENS_SN_MIN,DENS_SN)-DENS_SN_MAX)*EXP(-TAUF*TIMESTEP_SEC/TAU1)+DENS_SN_MAX !
Douville et al. (1995) p 23 eq 13
! DENS_SN=MAX(DENS_SN_MIN,MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN)))

!---2) Sun et al.
! GAMIC=(P_HM_SN-P_HW_SN)/P_HM_SN*DENS_SN ! [kg m-3] bulk density of ice, Sun et al. (1999)
pag. 19589 eq. 8
! GAMWA=P_HW_SN/P_HM_SN*DENS_WAT ! [kg m-3] bulk density of liquid water, Sun et al. (1999)
pag. 19589 eq. 8
! CC3=1.
! IF (GAMIC>150.) CC3=EXP(-0.06*(GAMIC-DENS_SN_MIN)) ! [] Sun et al. (1999) pag. 19589 eq. 8
! CC4=2.
! IF (GAMWA==0.) CC4=1. ! [] Sun et al. (1999) pag. 19589 eq. 8
! CR1=-CC1*CC3*CC4*EXP(-CC2*(DTK-P_TEMP_SN)) ! [s-1] compact rate for destructive
metamorphism
! PTOP=0.                   ! [Pa] snow load pressure over layer surface, put =0 since
single layer
! PSELF=4.9*DENS_SN*GRAVITY*HEI_SN ! [Pa] snow load pres. of layer sfc, Sun et al. (1999)
pag. 19589 eq. 10
! PS=PTOP+PSELF             ! [Pa] Sun et al. (1999) pag. 19589 eq. 10
! ETAA=ETA0*EXP(CC5*(DTK-P_TEMP_SN)+CC6*DENS_SN) ! [Pa s] Sun et al. (1999) pag. 19589 eq. 11
! CR2=-PS/ETAA ! [s-1] compaction rate due to snow overburden, Sun et al. (1999) pag. 19589
eq. 9
! DENS_SN=DENS_SN*EXP(-(CR1+CR2)*TIMESTEP_SEC) ! [kg m-3] Sun et al. (1999) pag. 19589 eq.
12-13
! DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))

!---3) Koren et al.
! CR1=TIMESTEP_SEC*CCK1*EXP(0.08*(P_TEMP_SN-DTK)-CCK2*DENS_SN) ! Koren et al (1999) p 19571
eq 8
! DENS_SN=DENS_SN*(EXP(CR1*P_HM_SN)-1.)/(CR1*P_HM_SN) ! Koren et al (1999) p 19571 eq 7
! DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))
```

```
! 4) fit on Russian data
! from experimental fit by Mutinelli (1998) on Ogurtsovo, Khabarovsk and Tulun
! Ruswet Siberian stations 1978-1984 (only values referring to the maximum annual
! snow heigth were selected)
! DENS_SN = DENS_SN_MIN + 4.537433 * HEI_SN * 100.
!
! 5) Mutinelli (1998) thesis
! IF (HM_SN>0.0001) THEN
!! the factor 0.06 because only 6% of porosity is used by water (D94 pag 183)
!  DENS_SN=DENS_ICE*MAX(0.,(1.-HW_SN_RET/HM_SN/0.06))+DENS_WAT*HW_SN/HM_SN
! !ELSE IF (HM_SN<=0.01 .AND. HM_SN>0.) THEN ! to avoid computational problems
!  !DENS_SN=DENS_SN_MIN
! !ELSE ! no snow
!  !DENS_SN=ERR_VE
! ENDIF

! IF (HEI_SN>0.001) THEN
!  DENS_SN=DENS_ICE*(HM_SN-HW_SN)/HEI_SN+DENS_WAT*HW_SN/HEI_SN
!  DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))
! ENDIF

! 6) Trevisan (2005) thesis: starting from alternative 5, propose:
! DENS_SN=DENS_ICE/(1+DENS_ICE*HW_SN_RET/0.06/DENS_WAT/HM_SN-HW_SN/HM_SN)

! 7) De Michele et al. (2013)
 IF (TWET<=(DTK-C3)) THEN
  DENS_SN_NEW=DENS_SN_MIN
! ELSE IF (TWET>(DTK-C3) .AND. TWET<DTK) THEN
 ELSE IF (TWET>(DTK-C3) ) THEN
  DENS_SN_NEW=MIN(DENS_SN_MAX,(DENS_SN_MIN+C2*(TWET-DTK+C3)**C4))
! ELSE
!  DENS_SN_NEW=C2*C3**C4+DENS_SN_MIN
 ENDIF

 RHOD = (C1*HEI_SN*RHOD0*EXP(C5*(TEMP_SN-DTK)-C6*RHOD0))*TIMESTEP_SEC+RHOD0

! compare density just calculated with old density and take the max value (old snow density
cannot decrease)
 RHOD=MAX(RHOD,DENS_SN_OLD)

! IF (PRECSN>0. .AND. HEI_SN>0.) RHOD = RHOD+((DENS_SN_NEW-RHOD0)/HEI_SN*PRECSN)*TIMESTEP_SEC
 IF (PRECSN>0. .AND. HEI_SN>0.) RHOD = (RHOD0*HEI_SN + DENS_WAT*PRECSN*TIMESTEP_SEC)/&
  (HEI_SN+DENS_WAT/DENS_SN_NEW*PRECSN*TIMESTEP_SEC)
!!!! DENS_SN=RHOD+DENS_WAT*HW_SN/HEI_SN
 DENS_SN=RHOD!+HW_SN/HM_SN
 DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))

!??? questo Commento seguente da riportare sul manuale

! Both Sun and Koren formulations use approximations surely good for a multilayer
! scheme but questionable for monolayer scheme
! In particular: snow temperature and density are constant over all snowpack

! DENS_SN is calculated as weighted averaged of old and new snow
! Next lines following G80

! new TEMP_SN is calculated as weighted averaged of old and new snow temperatures
! old snow temperature s TEMP_SN; new snow temperature is TWET<0^C
 IF (PRECSN>0.) THEN
  TEMP_SN=WEIGHTMEAN(MIN(DTK,TWET),TEMP_SN,(PRECSN*TIMESTEP_SEC),HM_SN) ! K
 ENDIF

ELSE IF (PRECSN>0..AND.P_HM_SN<=0.) THEN

! case B: there is new snow but not old snow (snow precipitation begins)
 TEMP_SN=MIN(DTK,TWET) ! K
```

```
! density of fresh show increases with air temp up to 0^C
 IF (TWET<=(DTK-C3)) THEN
  DENS_SN_NEW=DENS_SN_MIN
! ELSE IF (TWET>(DTK-C3) .AND. TWET<DTK) THEN
 ELSE IF (TWET>(DTK-C3) ) THEN
  DENS_SN_NEW=MIN(DENS_SN_MAX,(DENS_SN_MIN+C2*(TWET-DTK+C3)**C4))
! ELSE
!  DENS_SN_NEW=C2*C3**C4+DENS_SN_MIN
 ENDIF
 IF (DENS_SN>=0.) THEN
  PRINT *,'strange: there is not old snow but density is not error'
  PRINT *,'Snow density initialized to ',DENS_SN_NEW
 ENDIF
 DENS_SN=DENS_SN_NEW
 DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))

END IF

! Infiltration from snow to soil
IF (HW_SN>HW_SN_RET) THEN
 RUNSN=RUNSN+HW_SN-HW_SN_RET
 HM_SN=HM_SN-(HW_SN-HW_SN_RET)
 HW_SN=HW_SN_RET
ENDIF

! add contributions of snow prec and evaporation from snowy surface
! (normally so low that density is not affected)
DH=MAX(0.,(PRECSN-EVA_SN/DENS_ICE)*TIMESTEP_SEC)  ! m
IF (DH > 0.) THEN
 IF ( (HM_SN>=0. .AND. DENS_SN<=0.) ) DENS_SN = DENS_SN_MIN
 HM_SN=HM_SN+DH  ! m
ENDIF

! Increment of snow water content due to rain on snow (PREC_ON_SNOW)
IF (PREC_ON_SNOW>0.) THEN ! if it rains over snow
 IF (HW_SN >= HW_SN_RET) THEN
  RUNSN=RUNSN+PREC_ON_SNOW
 ELSE
  IF ( (PREC_ON_SNOW*TIMESTEP_SEC) > (HW_SN_RET-HW_SN) ) THEN
   DH = HW_SN_RET-HW_SN
   RUNSN = RUNSN+PREC_ON_SNOW*TIMESTEP_SEC-DH
   HM_SN = HM_SN + DH
   HW_SN = HW_SN + DH
   DENS_SN = WEIGHTMEAN(DENS_SN,DENS_WAT,HM_SN,DH)
   TEMP_SN = MIN( WEIGHTMEAN(TEMP_SN,TEMP_A_STEP,HM_SN,DH), DTK)
  ELSE
   HM_SN = HM_SN + PREC_ON_SNOW*TIMESTEP_SEC
   HW_SN = HW_SN + PREC_ON_SNOW*TIMESTEP_SEC
   DENS_SN = WEIGHTMEAN(DENS_SN,DENS_WAT,HM_SN,(PREC_ON_SNOW*TIMESTEP_SEC))
   TEMP_SN = MIN( WEIGHTMEAN(TEMP_SN,TEMP_A_STEP,HM_SN,(PREC_ON_SNOW*TIMESTEP_SEC)), DTK)
  ENDIF
 ENDIF
ENDIF ! from here exit with at max HW_SN = HW_SN_RET
DENS_SN=MIN(DENS_SN_MAX,MAX(DENS_SN_MIN,DENS_SN))

! Rules of thumb to avoid undesiderate values
! numerical filter to smooth variations: TEMP_SN arithmetic mean btw new & old snow temp
IF (TEMP_SN/=ERR_VE.AND.P_TEMP_SN/=ERR_VE) TEMP_SN=0.5*TEMP_SN+0.5*P_TEMP_SN

! to prevent too much refreezement - TEMP_SN >= AT - 50^C
IF (TEMP_SN < TEMP_A_STEP - 50.) TEMP_SN=MIN(DTK,MAX(TEMP_SN,(TEMP_A_STEP-50.)))

! lower limit of snowfall below which snow is considered absent
IF (HM_SN<=1.E-6) THEN ! to prevent negative or too little values
 HM_SN=0.
 HW_SN=0.
 TEMP_SN=ERR_VE
```

```
 DENS_SN=ERR_VE
 ALB_SN=ERR_VE
 LSNOW=.false.
END IF

! Infiltration from snow to soil
RUNSN=RUNSN+MAX(0.,(P_HM_SN-HM_SN)) ! m

! calculation of snow heigth by water equivalent and snow & water densities
HEI_SN=(DENS_WAT/DENS_SN)*MAX(0.,(HM_SN-HW_SN)) ! m

RETURN
END SUBROUTINE SNOW

!****************************************************************************
SUBROUTINE SNOW_FRACTION(VEG_HEI_STEP,HEI_SN,LSNOW,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,&
 COVER_SN,Z0_G_M,Z0_SN)
!****************************************************************************
! Calculates all snow fractions over canopy and bare soil
!****************************************************************************
! Calls: none
! - Author: C. Cassardo (13-Jan-2000)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: VEG_HEI_STEP
REAL,INTENT(IN) :: HEI_SN
LOGICAL,INTENT(IN) :: LSNOW
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(OUT) :: COVER_SN_F
REAL,INTENT(OUT) :: COVER_SN_G
REAL,INTENT(OUT) :: COVER_SN
REAL,INTENT(IN) :: Z0_G_M
REAL,INTENT(IN) :: Z0_SN
REAL :: ZZERO

! Calculation of snow fractions
IF (LSNOW) THEN

! follows an original idea of Cassardo
!---for canopy
 ZZERO=MAX(0.13*VEG_HEI_STEP,Z0_SN)
 COVER_SN_F=1.
 IF (HEI_SN<=(ZZERO/0.13)) COVER_SN_F=MAX(1.E-5,MIN(1.,(0.26*HEI_SN/ZZERO-&
   .0169*HEI_SN*HEI_SN/ZZERO/ZZERO)))
!---for bare soil
 ZZERO=MAX(Z0_G_M,Z0_SN)
 COVER_SN_G=1.
 IF (HEI_SN<=(ZZERO/0.13)) COVER_SN_G=MAX(1.E-5,MIN(1.,(0.26*HEI_SN/ZZERO-&
   .0169*HEI_SN*HEI_SN/ZZERO/ZZERO)))

! From above values calculation of snow fraction
 COVER_SN=VEG_COV_STEP*COVER_SN_F+(1.-VEG_COV_STEP)*COVER_SN_G ! snowy veget + bare soil

ELSE ! no snow

 COVER_SN=0.
 COVER_SN_F=0.
 COVER_SN_G=0.

END IF

! after these calculations the soil is partitioned in 4 zones:
! 1) bare soil without snow        (1.-VEG_COV_STEP)*(1.-COVER_SN_G)
! 2) bare soil with snow           (1.-VEG_COV_STEP)*COVER_SN_G
```

```
! 3) vegetation without snow       VEG_COV_STEP*(1.-COVER_SN_F)
! 4) vegetation with snow          VEG_COV_STEP*COVER_SN_F
RETURN
END SUBROUTINE SNOW_FRACTION

!******************************************************************************
SUBROUTINE SOIL(NUM_SOI_LAY,PREC_STEP,C_DREN,CAPPAETA,CAPPAETAM,CAPPAETA_S,DIFFSUMM,&
 DIFFVTM,THEDIFM,DEP_SOI,EVA_A,LEA_WAT_CON,SRF_WAT_CON,SRF_ICE_CON,SRF_WAT_CON_MAX,ETA_S,&
 INP_SOIL,LVEG,SR_SOI,P_SRF_WAT_CON,P_SRF_ICE_CON,P_TEM_SOI,ROCIVOL,ESSE,TIMESTEP_SEC,&
 EVA_G,EVA_G_W,EVATRA,PREC_G,Q_A_F,SO_SRAT_RO_DE,QSG_RH,RUNSN,VEL_MOD_STEP,ALAT,LHF_G,&
 SHF_A,IORA,IMINUTI,BAL_FLU_G,RAD_L_D,RAD_L_U,RAD_S_D,RAD_S_U,TEM_SOI,MID_LAY_DEP,SUM_DRAI,
 SUM_RUNO,&
 SUM_PREC,SUM_EVAP,SUM_SOI_HEA_STO,SUM_DRAI_TMP,SUM_RUNO_TMP,SUM_MIN_LW_TMP,SUM_PREC_TMP,
 SUM_EVAP_TMP,ETA_W,P_ETA_W,ETA_I,&
 P_ETA_I,SSR_FIE_CAPC,VEG_COV_STEP,MOIS_POT,BSOIL,MOIS_POT_SAT)
!******************************************************************************
!  Drives all soil processes
!******************************************************************************
! Calls: AV_NS_MOI, TSOIL, USOIL
! - Author: C. Cassardo (01-Apr-1999)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: ALAT
REAL,INTENT(IN) :: PREC_STEP
REAL,INTENT(IN) :: VEL_MOD_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: BSOIL
REAL,INTENT(IN) :: ESSE
REAL,INTENT(IN) :: C_DREN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: CAPPAETA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: CAPPAETAM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: THEDIFM
REAL,PARAMETER :: DELTAQMAX=1.5E-4
REAL,PARAMETER :: DELTATMAX=3.E-4
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: DIFFSUMM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: DIFFVTM
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_A
REAL,INTENT(IN) :: EVA_G
REAL,INTENT(IN) :: EVA_G_W
REAL,INTENT(IN) :: LEA_WAT_CON
REAL,INTENT(INOUT) :: SRF_WAT_CON
REAL,INTENT(INOUT) :: SRF_ICE_CON
REAL,INTENT(IN) :: SRF_WAT_CON_MAX
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_W
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: EVATRA
REAL,INTENT(IN) :: LHF_G
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MOIS_POT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MOIS_POT_SAT
REAL :: FT
REAL,INTENT(IN) :: SHF_A
INTEGER :: I
INTEGER, INTENT(IN) :: IMINUTI
INTEGER, INTENT(IN) :: INP_SOIL
INTEGER,INTENT(IN) :: IORA
INTEGER, INTENT(IN) :: LVEG
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_W
REAL,INTENT(INOUT) :: P_SRF_WAT_CON
REAL,INTENT(INOUT) :: P_SRF_ICE_CON
```

```
REAL,INTENT(INOUT) :: PREC_G
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: SR_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_TEM_SOI
REAL,INTENT(IN) :: Q_A_F
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SSR_FIE_CAPC
REAL :: BAL_FLU_G_BOT
REAL,INTENT(IN) :: BAL_FLU_G
REAL,INTENT(IN) :: SO_SRAT_RO_DE
REAL,INTENT(IN) :: QSG_RH
REAL,INTENT(IN) :: RAD_L_D
REAL,INTENT(IN) :: RAD_L_U
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: ROCIVOL
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U
REAL :: RUNG
REAL,INTENT(INOUT) :: RUNSN
REAL :: RUNU
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(INOUT) :: SUM_PREC
REAL,INTENT(INOUT) :: SUM_PREC_TMP
REAL,INTENT(INOUT) :: SUM_EVAP
REAL,INTENT(INOUT) :: SUM_EVAP_TMP
REAL,INTENT(INOUT) :: SUM_SOI_HEA_STO
REAL,INTENT(INOUT) :: SUM_MIN_LW_TMP
REAL,INTENT(INOUT) :: SUM_RUNO
REAL,INTENT(INOUT) :: SUM_RUNO_TMP
REAL,INTENT(INOUT) :: SUM_DRAI
REAL,INTENT(INOUT) :: SUM_DRAI_TMP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: TEM_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP

CALL TSOIL(NUM_SOI_LAY,VEL_MOD_STEP,ALAT,THEDIFM,DEP_SOI,TIMESTEP_SEC,LHF_G,SHF_A,LVEG,&
           IORA,IMINUTI,P_TEM_SOI,BAL_FLU_G,BAL_FLU_G_BOT,RAD_L_D,RAD_L_U,RAD_S_D,RAD_S_U,
           ROCIVOL,TEM_SOI,&
           MID_LAY_DEP,P_ETA_I,P_ETA_W,INP_SOIL,SSR_FIE_CAPC,ETA_S,VEG_COV_STEP)

! Soil moisture calculation (done only if not ice or water surface)
IF (LVEG/=12 .OR. LVEG/=14 .OR. LVEG/=15) THEN

 CALL USOIL(ESSE,C_DREN,CAPPAETA,CAPPAETAM,CAPPAETA_S,DIFFSUMM,DIFFVTM,DEP_SOI,TIMESTEP_SEC,&
  EVA_G,EVA_G_W,SRF_WAT_CON,SRF_ICE_CON,SRF_WAT_CON_MAX,ETA_I,ETA_S,ETA_W,EVATRA,&
  MOIS_POT,INP_SOIL,NUM_SOI_LAY,P_ETA_W,P_ETA_I,P_SRF_WAT_CON,P_SRF_ICE_CON,PREC_G,SR_SOI,&
  P_TEM_SOI,TEM_SOI,Q_A_F,SO_SRAT_RO_DE,QSG_RH,ROCIVOL,RUNG,RUNSN,RUNU,VEG_COV_STEP,
  SSR_FIE_CAPC,&
  BSOIL,MOIS_POT_SAT)

END IF

! modification of soil temperature due to the freezing: this is caused by variation
! of soil moisture, but only in Viterbo et al. modified formulation
DO I=1,NUM_SOI_LAY
IF (P_TEM_SOI(I)<DTK) THEN
  IF (INP_SOIL == 5) THEN
   IF (P_TEM_SOI(I) > TFRZ1) THEN ! all moisture is liquid water
    FT=0.
   ELSE IF (P_TEM_SOI(I) < TFRZ2) THEN ! all moisture is ice
    FT=1
   ELSE ! there is ice and moisture depending on temperature
    FT=0.5*(1.-SIN((PI*(P_TEM_SOI(I)-0.5*TFRZ1-0.5*TFRZ2))/(TFRZ1-TFRZ2)))
   ENDIF
   TEM_SOI(I)=TEM_SOI(I)+ROW*CLF/(ROCIVOL(I)/DEP_SOI(I))*FT*(ETA_W(I)+ETA_I(I)-P_ETA_W(I)-
   P_ETA_I(I))
  ENDIF
 ENDIF
ENDDO

!upgrade old soil temperature
```

```
P_TEM_SOI(:)=TEM_SOI(:)

!upgrade old volumetric soil water and ice content
P_ETA_W(:)=ETA_W(:)
P_ETA_I(:)=ETA_I(:)

! for ice soil type, all incident water becomes runoff and drainage is set to zero
IF (LVEG==12) THEN
 RUNU = PREC_G*TIMESTEP_SEC+RUNSN !m
 RUNG = 0. !m
END IF

! for soil type different from water, cumulative variables are incremented
IF (LVEG/=14 .OR. LVEG/=15) THEN
 SUM_DRAI_TMP = SUM_DRAI_TMP + RUNU !m
 SUM_RUNO_TMP = SUM_RUNO_TMP + RUNG !m
 IF (LEA_WAT_CON > LEA_WET_THRE) THEN
  SUM_MIN_LW_TMP  = SUM_MIN_LW_TMP + TIMESTEP_SEC/60.
 END IF
 SUM_DRAI     = SUM_DRAI + RUNU !m
 SUM_RUNO     = SUM_RUNO + RUNG !m
END IF

! cumulated variables
SUM_PREC_TMP=SUM_PREC_TMP+(PREC_STEP/3.6e6)*TIMESTEP_SEC     ! m (AP converted from mm/h to
m/s)
SUM_PREC=SUM_PREC+(PREC_STEP/3.6e6)*TIMESTEP_SEC             ! m (AP converted from mm/h to
m/s)
SUM_EVAP_TMP=SUM_EVAP_TMP+EVA_A*TIMESTEP_SEC/DENS_WAT ! m
SUM_EVAP=SUM_EVAP+EVA_A*TIMESTEP_SEC/DENS_WAT         ! m
SUM_SOI_HEA_STO=SUM_SOI_HEA_STO+(BAL_FLU_G-BAL_FLU_G_BOT)*1.E-6     ! MW/m2

RETURN
END SUBROUTINE SOIL

!*****************************************************************************
SUBROUTINE SOL_ANG(NDAY,IORA,IMI,ALON,ALAT,CLEG,SOLAR_ANGLE,L360DAYS_CALENDAR)
!*****************************************************************************
!  Calculates the angle of solar elevation (Page)
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Mar-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: ALAT ! Latitude in degrees (-90 -> 90)
REAL,INTENT(IN) :: ALON ! Longitude in degrees (-180 -> 180)
REAL :: ALONST
REAL,EXTERNAL :: ASIND
REAL,EXTERNAL :: ATAND
REAL,INTENT(IN) :: CLEG ! Code legal hour (1 -> yes, 0 -> no)
REAL,EXTERNAL :: COSD
REAL :: DEL
REAL :: ET
REAL :: GEIP
INTEGER :: II
INTEGER,INTENT(IN) :: IMI  ! Minutes (0 -> 60)
INTEGER,INTENT(IN) :: IORA ! Hour (0 -> 24)
INTEGER :: KK
LOGICAL,INTENT(IN) :: L360DAYS_CALENDAR
INTEGER,INTENT(IN) :: NDAY ! Julian (=progressive number of) day (1->365, 1=1 Jan)
REAL :: OMEG
REAL,EXTERNAL :: SIND
REAL :: SING
REAL,INTENT(OUT) :: SOLAR_ANGLE ! Solar elevation (sunrise = 0deg, zenith = 90deg)
```

```
REAL :: T
REAL :: TT

ALONST=0.

! Determination of longitude of time reference zones (deg, Page pag 393):
! it is the slice of earth of 15 degrees of time zone (of mean
! longitude ALONST) in which there is the point of coordinates ALON, ALAT
! Attention: point ALON-ALAT could not be in the middle of the slice thus each
! point is assigned to a slice depending on ALONST
DO II=1,24
 KK=II-13   ! search interval of ALON from -180 to +180
! at the point of longitude ALON we assign the slice with the superior extreme of
! the 15deg interval
 IF (ALON > REAL(KK*15) .AND. ALON <= REAL((KK+1)*15)) THEN
  ALONST=REAL((KK+1)*15)
  EXIT
 END IF
END DO

TT=REAL(IORA)+REAL(IMI)/60. !local standard time, hours
IF(L360DAYS_CALENDAR)THEN
 GEIP=REAL(NDAY)
ELSE
!day angle (deg), Page eq. A1.2, conversion day_of_year - solar_angle
 GEIP=360.*REAL(NDAY)/365.25
END IF

! equation of time, hours, Page eq. A1.5.1
ET=-0.128*SIND(GEIP-2.8)-0.165*SIND(2*GEIP+19.7)

! local apparent time, h, Page eq. A1.5.2
T=TT+(ALON-ALONST)/15.+ET-CLEG

! solar declination, deg, Page eq. A1.3
DEL=ASIND(0.3978*SIND(GEIP-80.2+1.92*SIND(GEIP-2.80)))

OMEG=15.*(T-12) ! solar hour angle, deg, Page eq. A1.6

SING=SIND(ALAT)*SIND(DEL)+COSD(OMEG)*COSD(ALAT)*COSD(DEL)
SOLAR_ANGLE=ASIND(SING) ! solar altitude, deg, Page eq. A1.7

RETURN
END SUBROUTINE SOL_ANG

!*************************************************************************
SUBROUTINE SOLARHS(NUM_SOI_LAY,DEP_SOI,RAD_S_D,RAD_S_U,SRHS,SRHS0,MID_LAY_DEP)
!*************************************************************************
! This subroutine is to calculate the solar radiation heat source term
! BTAU is the fraction of net solar radiation absorbed in the surface
!      layer(input)
! BAS_SFC_BS_LAY is the base of the surface absorption layer (meter) (input)
! EXT_COEF is the light extinction coefficient for water(input)
! SRHS(I) is the solar radiation heat source (output)
! SRHS0 is the solar radiation heat source at surface (output)
! MID_LAY_DEP(I) is the depth of the atithmetic mean point of each water layer
!*************************************************************************
! Calls: none
! - Author : Minwei Qian
! - Date   : 28-11-2002
! - Last revision : 06-04-2004  M.Trevisan (f90 translation)
!*************************************************************************
USE CONSTANTS
IMPLICIT NONE

!..... Global variables declaration
INTEGER,INTENT(IN) :: NUM_SOI_LAY
```

```
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
INTEGER :: I
!INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: SRHS
REAL,INTENT(OUT) :: SRHS0
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP

SRHS0=EXT_COEF*(1.-BTAU)*(RAD_S_D-RAD_S_U)*EXP(EXT_COEF*BAS_SFC_BS_LAY)
DO I=1, NUM_SOI_LAY
 SRHS(I)=EXT_COEF*(1.-BTAU)*(RAD_S_D-RAD_S_U)*EXP(-EXT_COEF*(MID_LAY_DEP(I)-BAS_SFC_BS_LAY))
!?? what if MID_LAY_DEP<BAS_SFC_BS_LAY???
 IF (MID_LAY_DEP(I)<BAS_SFC_BS_LAY) STOP 'In SOLARHS MID_LAY_DEP<BAS_SFC_BS_LAY'
END DO

RETURN
END SUBROUTINE SOLARHS

!*****************************************************************************
SUBROUTINE SPECUM(RH_STEP,TEMP_A_STEP,PRES_A_STEP,SPEC_UM,E,ES)
!*****************************************************************************
! Calculates actual and saturated water vapor and specific humidity using Clausius
! Clapeyron relation
!*****************************************************************************
! Calls: none
! - Author: C. Cassardo (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: PRES_A_STEP
REAL,INTENT(IN) :: TEMP_A_STEP
REAL,INTENT(OUT) :: E
REAL,INTENT(OUT) :: ES
REAL,INTENT(IN) :: RH_STEP
REAL,INTENT(OUT) :: SPEC_UM
REAL :: T

IF (TEMP_A_STEP >= DTK) THEN
 ES=A1*EXP(A2*(TEMP_A_STEP-A3)/(TEMP_A_STEP-A4)) !sat. water vap. pressure (hPa)
ELSE
 T=TEMP_A_STEP-273.15
 ES=(B0+T*(B1+T*(B2+T*(B3+T*(B4+T*(B5+T*B6))))))
ENDIF

E=RH_STEP/100.*ES ! water vapor pressure (hPa)

SPEC_UM=A5*E/(PRES_A_STEP-A6*E) ! specific humidity (kg/kg)

RETURN
END SUBROUTINE SPECUM

!*****************************************************************************
SUBROUTINE SURFPROC(ALB_FH,ALAT,ALB_SD,ALB_SN,ALB_SNT,ALB_TOT,ALON,PREC_STEP,PRES_A_STEP,&
  Q_SP_A_STEP,ESSE,TEMP_A_STEP,VEL_U_STEP,VEL_V_STEP,VEL_MOD_STEP,GAMMA,C_DREN,CAPPAETA,
  CAPPAETAM,&
  CAPPAMIXF,CAPPAMIXG,DRAG_H,DRAG_M,DRAG_V,VEG_HEA_CAP,CLEG,CL_TOT_STEP,CO2_CON_STEP,D0_VEGPAR
  ,&
  AIR_DENS,DENS_SN,DIFFSUM,DIFFSUMM,DIFFVT,DIFFVTM,ZER_DIS_LEV,&
  TIMESTEP_SEC,EVA_G,EVA_G_W,EVA_F_W,LEA_WAT_CON,SRF_WAT_CON,SRF_ICE_CON,&
  LEA_WAT_CON_MAX,SRF_WAT_CON_MAX,EPSF,EPSG,EVATRA,LHF_A,LHF_F,LHF_G,MOIS_POT,&
  LHF_SN,LHF_F_D,VEG_HEI_STEP,SHF_A,SHF_F,SHF_G,SHF_SN,SHF_SN_F,SHF_SN_G,HAZE_IND,HM_SN,HEI_SN
  ,HW_SN,&
  ILOGALB,ILOGFUN,LLONGWFLAG,LVAP,LVEG,NUM_SOI_LAY,LSNOW,LSNOW2,LTCAN_NEW,IMINUTI,&
  INP_SOIL,IORA,L360DAYS_CALENDAR,LAI_STEP,JU_DAY,PRECSN,SUM_PREC_SN,P_HM_SN,&
```

```
  P_HW_SN,P_LEA_WAT_CON,P_SRF_WAT_CON,P_SRF_ICE_CON,PREC_G,P_TEMP_F_STEP,P_TEMP_SN,SSR_FIE_CAP
  ,&
  SO_SRAT_RO_DE,SSR_WI,Q_A_F,BAL_FLU_F,BAL_FLU_G,QSF,QSG,CONF_SN_SRF,RESIST_AH,RESIST_AM,&
  RESIST_AV,RESIST_B,RESIST_D,RESIST_F,RESIST_LEAF,RGL_NOILHAN,RH_STEP,RESIST_F_MIN,&
  ROCIVOL,RP_TOTAL,REL_LEA_WAT_CON,REL_SRF_WAT_CON,REL_SRF_ICE_CON,RAD_L_D,NR_L_F_D,NR_L_F_U,
  RAD_L_G_TOT_D,&
  RAD_L_G_TOT_U,RAD_L_U,RAD_L_FG_D,RAD_L_FG_U,NR_A,NR_F,NR_G,NR_SN,RAD_S_D,RAD_S_F_D,RAD_S_F_U
  ,RAD_S_G_D,&
  RAD_S_G_U,RAD_S_U,SOL_ALFA,COVER_SN,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,SUM_DRAI,SUM_RUNO,&
  SUM_PREC,SUM_EVAP,SUM_SOI_HEA_STO,SUM_DRAI_TMP,SUM_RUNO_TMP,SUM_MIN_LW_TMP,SUM_PREC_TMP,&
  SUM_EVAP_TMP,TORTUOS,TEMP_A_F,TEMP_F_STEP,TEMP_ROOT,TEMP_SN,H_OBS_ALL,Z0_M,Z0_H,Z0_V,BSOIL,&
  CAPPAETA_S,ETA_S,THEDIFM,RP_LAYER,DEP_SOI,MOIS_POT_SAT,SR_SOI,P_TEM_SOI,TEM_SOI,ROC,&
  MID_LAY_DEP,USTAR,Z0_SN,ETA_W,P_ETA_W,ETA_I,SSR_FIE_CAPC,ERR_ECO,Z0_M_ECO,Z0_H_ECO,P_ETA_I,&
  H_OBS_WIND,NSO,GRO_CAR_ASS_RAT,NET_CAR_ASS_RAT,HAJ,HAV,HDJ,HDV,JOPT,LMEDLYN,MRS,PTW,
  RESIST_F_PHOTO,RES_RAT,TOPTJ,&
  TOPTV,VOPT,PVMAX25,LPHOTO,BAL_FLU_SN,BAL_FLU_TOT,&
  RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_SN_G_D,RAD_S_SN_G_U,RAD_S_SN_D,RAD_S_SN_U,RAD_L_F_D,
  RAD_L_F_U,RAD_L_G_D,RAD_L_G_U,&
  RAD_L_SN_F_U,RAD_L_SN_G_D,RAD_L_SN_G_U,RAD_L_SN_U,NR_L_SN_D,NR_L_SN_U,LHF_F_W,LHF_G_D,
  LHF_G_W,LHF_SN_F,LHF_SN_G,&
  EVA_A,EVA_F,EVA_F_D,EVA_G_D,EVA_SN,EVA_SN_F,EVA_SN_G,UPWP,VPWP,CONF_SN_F,CONF_SN_G,Q_RAIN_F,
  Q_RAIN_G,RESIST_D_SN,RESIST_SRF,RESIST_SRF_SN,&
  CONDUCT_AH,CONDUCT_AM,CONDUCT_AV,CONDUCT_B,CONDUCT_D,CONDUCT_D_SN,COND_F_DRY,CONDUCT_F,
  COND_F_WET,COND_G_DRY,&
  COND_G_WET,PREC_F,QSG_RH,RAD_L_SN_F_D)
!*****************************************************************************
! It is the main driver of the model and does the calculations of all output in a
! given timestep DT
!*****************************************************************************
! Calls: ALBEDO, AV_NS_MOI, D_SPEC_UM, DRAG, FLUXES, ISTHERESNOW, RADIATION,
!        RESIST, SNOW, SNOW_FRACTION, SOIL, SPECUM, TCAN, WATER_MASS
! - Authors: C. Cassardo, J.J. Ji, E. Carena (13-Jul-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL, INTENT(OUT) :: GRO_CAR_ASS_RAT    ! gross rate of carbon assimilation by photosynthesis
[umol/m2s]
REAL, INTENT(OUT) :: NET_CAR_ASS_RAT    ! net rate of carbon assimilation by photosynthesis
[umol/m2s]
REAL, INTENT(OUT) :: RES_RAT            ! mitocondrial respiration [umol/m2s]
REAL :: ALB_F
REAL,INTENT (IN) :: ALB_FH
REAL :: ALB_G
REAL,INTENT (IN) :: ALAT
REAL,INTENT (IN) :: ALB_SD
REAL,INTENT(OUT) :: ALB_SN
REAL,INTENT(INOUT) :: ALB_SNT
REAL,INTENT(OUT) :: ALB_TOT
REAL,INTENT (IN) :: ALON
REAL,INTENT (IN) :: PREC_STEP
REAL,INTENT (IN) :: PRES_A_STEP
REAL,INTENT (IN) :: Q_SP_A_STEP
REAL,INTENT (IN) :: ESSE
REAL,INTENT (IN) :: TEMP_A_STEP
REAL,INTENT (IN) :: VEL_U_STEP
REAL,INTENT (IN) :: VEL_MOD_STEP
REAL,INTENT (IN) :: VEL_V_STEP
REAL,INTENT (IN) :: GAMMA
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: BSOIL
REAL,INTENT(IN) :: C_DREN
REAL, DIMENSION(NUM_SOI_LAY), INTENT(OUT) :: CAPPAETA
REAL, DIMENSION(NUM_SOI_LAY), INTENT(OUT) :: CAPPAETAM
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: CAPPAETA_S
REAL, DIMENSION(NUM_SOI_LAY), INTENT(OUT) :: THEDIFM
```

```
REAL,INTENT(OUT) :: CAPPAMIXF
REAL,INTENT(OUT) :: CAPPAMIXG
REAL :: CAPPASN
REAL :: CAPPAUNO
REAL,INTENT(OUT) :: DRAG_H
REAL,INTENT(OUT) :: DRAG_M
REAL,INTENT(OUT) :: DRAG_V
REAL,INTENT (IN) :: VEG_HEA_CAP
REAL,INTENT (IN) :: CLEG
REAL,INTENT (IN) :: CL_TOT_STEP
REAL,INTENT (IN) :: CO2_CON_STEP
REAL :: D_BAL_FLU_F_D_TF
REAL :: D_NR_F_D_TF
REAL,INTENT (IN) :: D0_VEGPAR
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: RP_LAYER
REAL,INTENT (IN) :: AIR_DENS
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: DEP_SOI
REAL,INTENT(INOUT) :: DENS_SN
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFSUM
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFSUMM
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFVT
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFVTM
REAL,INTENT(INOUT) :: ZER_DIS_LEV
REAL,INTENT (IN) :: TIMESTEP_SEC
REAL,INTENT(OUT) :: BAL_FLU_SN
REAL :: E_CAN
REAL,INTENT(OUT) :: EVA_A
REAL,INTENT(OUT) :: EVA_F
REAL,INTENT(INOUT) :: EVA_F_D
REAL,INTENT(INOUT) :: EVA_F_W
REAL,INTENT(INOUT) :: EVA_G
REAL,INTENT(INOUT) :: EVA_G_D
REAL,INTENT(INOUT) :: EVA_G_W
REAL,INTENT(INOUT) :: EVA_SN_F
REAL,INTENT(INOUT) :: EVA_SN_G
REAL,INTENT(INOUT) :: LEA_WAT_CON
REAL,INTENT(INOUT) :: SRF_WAT_CON
REAL,INTENT(INOUT) :: SRF_ICE_CON
REAL,INTENT (IN) :: LEA_WAT_CON_MAX
REAL,INTENT (IN) :: SRF_WAT_CON_MAX
REAL,INTENT (IN) :: EPSF
REAL,INTENT (IN) :: EPSG
REAL,INTENT (IN) :: ERR_ECO
REAL :: ES_CAN
REAL :: ES_CANOPY
REAL,INTENT(OUT) :: EVA_SN
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: ETA_I
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: ETA_S
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: ETA_W
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: EVATRA
REAL,INTENT(OUT) :: LHF_A
REAL,INTENT(OUT) :: LHF_F
REAL,INTENT(OUT) :: LHF_G
REAL :: SRF_REL_HUM
REAL, DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: MOIS_POT
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: MOIS_POT_SAT
REAL,INTENT(OUT) :: LHF_SN
REAL,INTENT(OUT) :: LHF_SN_F
REAL,INTENT(OUT) :: LHF_SN_G
REAL,INTENT(OUT) :: LHF_F_D
REAL,INTENT(OUT) :: LHF_F_W
REAL,INTENT(OUT) :: LHF_G_D
REAL,INTENT(OUT) :: LHF_G_W
REAL,INTENT(OUT) :: SHF_A
REAL,INTENT(IN)  :: HAJ
REAL,INTENT(IN)  :: HAV
REAL,INTENT(OUT) :: HAZE_IND
```

```
REAL,INTENT(IN)  :: HDJ
REAL,INTENT(IN)  :: HDV
REAL,INTENT(OUT) :: SHF_F
REAL,INTENT(OUT) :: SHF_G
REAL,INTENT(OUT) :: SHF_SN
REAL,INTENT (IN) :: VEG_HEI_STEP
REAL,INTENT(OUT) :: HM_SN
REAL,INTENT(INOUT) :: HEI_SN
REAL,INTENT(OUT) :: SHF_SN_F
REAL,INTENT(OUT) :: SHF_SN_G
REAL,INTENT(INOUT) :: HW_SN
LOGICAL, INTENT(IN) :: ILOGALB
LOGICAL, INTENT(IN) :: ILOGFUN
INTEGER,INTENT(IN) :: IMINUTI
INTEGER,INTENT(IN) :: INP_SOIL
INTEGER,INTENT(IN) :: IORA
REAL,INTENT(IN) :: JOPT
LOGICAL,INTENT(IN) :: L360DAYS_CALENDAR
REAL,INTENT(IN) :: LAI_STEP
LOGICAL, INTENT(IN) :: LLONGWFLAG
LOGICAL, INTENT(IN) :: LMEDLYN ! If true, it activates Medlyn parameterization
LOGICAL, INTENT(IN) :: LPHOTO  ! If true, it activates photosynthesis parameterization
LOGICAL, INTENT(INOUT) :: LSNOW
LOGICAL :: LSNOW1
LOGICAL, INTENT(OUT) :: LSNOW2
LOGICAL :: LSNOW3
LOGICAL,INTENT(IN) :: LTCAN_NEW
LOGICAL, INTENT(INOUT) :: LVAP
INTEGER,INTENT(IN) :: LVEG
REAL,INTENT(IN) :: MRS
INTEGER,INTENT(IN) :: JU_DAY
INTEGER,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: NSO
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: P_ETA_I
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: P_ETA_W
REAL :: PREC_ON_SNOW
REAL,INTENT(INOUT) :: P_HM_SN
REAL,INTENT(INOUT) :: P_HW_SN
REAL :: PIPPO
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: SR_SOI
REAL,INTENT(INOUT) :: PRECSN
REAL,INTENT(OUT) :: SUM_PREC_SN
REAL,INTENT (IN) :: P_LEA_WAT_CON
REAL,INTENT (INOUT) :: P_SRF_WAT_CON
REAL,INTENT (INOUT) :: P_SRF_ICE_CON
REAL,INTENT(OUT) :: PREC_F
REAL,INTENT(OUT) :: PREC_G
REAL,INTENT (IN) :: P_TEMP_F_STEP
REAL, DIMENSION(NUM_SOI_LAY), INTENT(INOUT) :: P_TEM_SOI
REAL,INTENT (INOUT) :: P_TEMP_SN
INTEGER,INTENT(IN) :: PTW
REAL,INTENT(OUT) :: Q_A_F
REAL,INTENT (IN) :: SSR_FIE_CAP
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: SSR_FIE_CAPC
REAL,INTENT(OUT) :: BAL_FLU_F
REAL,INTENT(OUT) :: BAL_FLU_G
REAL, INTENT(OUT) :: BAL_FLU_TOT    ! effective imbalance between radiative and turbulent
heat fluxes [W m-2]
REAL :: QSAT_CANOPY !for TCAN
REAL,INTENT(OUT) :: QSF
REAL,INTENT(OUT) :: QSG
REAL,INTENT(OUT) :: QSG_RH
REAL,INTENT (IN) :: SO_SRAT_RO_DE
REAL,INTENT(OUT) :: CONF_SN_SRF
REAL,INTENT(OUT) :: CONF_SN_F
REAL,INTENT(OUT) :: CONF_SN_G
REAL :: QSSN
REAL :: QSURF
```

```
REAL,INTENT (IN) :: SSR_WI
REAL,INTENT(OUT) :: RESIST_AH
REAL,INTENT(OUT) :: RESIST_AM
REAL,INTENT(OUT) :: RESIST_AV
REAL,INTENT(OUT) :: RESIST_B
REAL,INTENT(OUT) :: RESIST_D
REAL,INTENT(OUT) :: RESIST_D_SN
REAL,INTENT(OUT) :: REL_LEA_WAT_CON
REAL,INTENT(OUT) :: REL_SRF_WAT_CON
REAL,INTENT(OUT) :: REL_SRF_ICE_CON
REAL :: REYNOLDS
REAL,INTENT(OUT) :: RESIST_LEAF
REAL,INTENT(OUT) :: RESIST_F
REAL,INTENT(OUT) :: RAD_L_G_D
REAL,INTENT(OUT) :: RAD_L_G_U
REAL,INTENT (IN) :: RGL_NOILHAN
REAL,INTENT (IN) :: RH_STEP
REAL,INTENT(INOUT) :: RAD_L_D
REAL,INTENT(OUT) :: NR_L_F_D
REAL,INTENT(OUT) :: NR_L_F_U
REAL,INTENT(OUT) :: RAD_L_SN_F_D
REAL,INTENT(OUT) :: RAD_L_SN_F_U
REAL,INTENT(OUT) :: RAD_L_G_TOT_D
REAL,INTENT(OUT) :: RAD_L_G_TOT_U
REAL,INTENT(OUT) :: RAD_L_SN_G_D
REAL,INTENT(OUT) :: RAD_L_SN_G_U
REAL,INTENT(OUT) :: RAD_L_SN_U
REAL,INTENT(OUT) :: NR_L_SN_D
REAL,INTENT(OUT) :: NR_L_SN_U
REAL,INTENT(OUT) :: RAD_L_U
REAL,INTENT(OUT) :: RAD_L_F_D
REAL,INTENT(OUT) :: RAD_L_F_U
REAL,INTENT(OUT) :: RAD_L_FG_D
REAL,INTENT(OUT) :: RAD_L_FG_U
REAL,INTENT (IN) :: RESIST_F_MIN
REAL,INTENT(OUT) :: NR_F
REAL,INTENT(OUT) :: NR_G
REAL,INTENT(OUT) :: NR_A
REAL,INTENT(OUT) :: NR_SN
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: ROC
REAL, DIMENSION(NUM_SOI_LAY), INTENT(OUT) :: ROCIVOL
REAL, DIMENSION(NUM_SOI_LAY), INTENT (IN) :: RP_TOTAL
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(OUT) :: RAD_S_F_D
REAL,INTENT(OUT) :: RAD_S_F_U
REAL,INTENT(OUT) :: RAD_S_SN_F_D
REAL,INTENT(OUT) :: RAD_S_SN_F_U
REAL,INTENT(OUT) :: RAD_S_G_D
REAL,INTENT(OUT) :: RAD_S_G_U
REAL,INTENT(OUT) :: RAD_S_SN_G_D
REAL,INTENT(OUT) :: RAD_S_SN_G_U
REAL,INTENT(OUT) :: RESIST_F_PHOTO
REAL,INTENT(OUT) :: RESIST_SRF
REAL,INTENT(OUT) :: RESIST_SRF_SN
REAL,INTENT(OUT) :: RAD_S_SN_D
REAL,INTENT(OUT) :: RAD_S_SN_U
REAL,INTENT(OUT) :: RAD_S_U
REAL :: RUNSN
REAL,INTENT(OUT) :: CONDUCT_AH
REAL,INTENT(OUT) :: CONDUCT_AM
REAL,INTENT(OUT) :: CONDUCT_AV
REAL,INTENT(OUT) :: CONDUCT_B
REAL,INTENT(OUT) :: CONDUCT_D
REAL,INTENT(OUT) :: CONDUCT_D_SN
REAL,INTENT(OUT) :: COND_F_DRY
REAL,INTENT(OUT) :: CONDUCT_F
REAL,INTENT(OUT) :: COND_F_WET
```

```
REAL,INTENT(OUT) :: COND_G_DRY
REAL,INTENT(OUT) :: COND_G_WET
REAL,INTENT (IN) :: VEG_COV_STEP
REAL,INTENT(OUT) :: COVER_SN
REAL,INTENT(OUT) :: COVER_SN_F
REAL,INTENT(OUT) :: COVER_SN_G
REAL,INTENT(OUT) :: SOL_ALFA
REAL :: SOL_ZEN   ! sol_alfa and sol_zen are in truth local variables
                  !of albedo (no other  routines call it) ??? e allora?
REAL :: COND_SRF_VAP
REAL :: CONDUCT_SRF_SN
REAL,INTENT(INOUT) :: SUM_PREC
REAL,INTENT(INOUT) :: SUM_PREC_TMP
REAL,INTENT(INOUT) :: SUM_EVAP
REAL,INTENT(INOUT) :: SUM_EVAP_TMP
REAL,INTENT(INOUT) :: SUM_SOI_HEA_STO
REAL,INTENT(INOUT) :: SUM_MIN_LW_TMP
REAL,INTENT(INOUT) :: SUM_RUNO_TMP
REAL,INTENT(INOUT) :: SUM_RUNO
REAL,INTENT(INOUT) :: SUM_DRAI_TMP
REAL,INTENT(INOUT) :: SUM_DRAI
REAL,INTENT(OUT) :: TEMP_A_F
REAL,INTENT(OUT) :: TEMP_F_STEP
REAL, DIMENSION(NUM_SOI_LAY), INTENT(OUT) :: TEM_SOI
REAL,INTENT(IN) :: TOPTJ
REAL,INTENT(IN) :: TOPTV
REAL,INTENT (IN) :: TORTUOS
REAL,INTENT(OUT) :: TEMP_ROOT
REAL, INTENT(INOUT) :: TEMP_SN
REAL :: TSURF
REAL :: TWET
REAL :: UAF
REAL,INTENT(INOUT) :: UPWP
REAL,INTENT(INOUT) :: USTAR
REAL,INTENT(INOUT) :: VPWP
REAL,INTENT(IN) :: PVMAX25
REAL,INTENT(IN) :: VOPT
REAL :: Z0_F_H
REAL :: Z0_F_M
REAL :: Z0_G_H
REAL :: Z0_G_M
REAL :: Z0_G_V
REAL,INTENT(INOUT) :: Z0_H
REAL,INTENT (IN) :: Z0_H_ECO
REAL,INTENT(INOUT) :: Z0_M
REAL,INTENT (IN) :: Z0_M_ECO
REAL,INTENT(INOUT) :: Z0_SN
REAL,INTENT(INOUT) :: Z0_V
REAL,INTENT(INOUT) :: H_OBS_ALL
REAL :: ACT_VEG_HEI           !  local of drag !!!!!!!!!!!
REAL, INTENT(INOUT) :: H_OBS_WIND       !screen level height for wind
REAL, DIMENSION(NUM_SOI_LAY), INTENT(IN) :: MID_LAY_DEP

REAL,INTENT(OUT) :: Q_RAIN_F
REAL,INTENT(OUT) :: Q_RAIN_G
REAL :: NU

! calculation of the haze effect
IF(LLONGWFLAG)THEN
 HAZE_IND=0.
ELSE
 CALL HAZE(ILOGFUN,RH_STEP,VEL_MOD_STEP,RAD_S_D,HAZE_IND)
END IF

! precipitation and drainage of canopy
IF (VEG_COV_STEP /= 0.) THEN
 REL_LEA_WAT_CON = MAX( MIN(1., P_LEA_WAT_CON/LEA_WAT_CON_MAX), 0. ) ! relative canopy wetness
```

```
ELSE
 REL_LEA_WAT_CON = 0.
END IF

REL_SRF_WAT_CON=0.
REL_SRF_ICE_CON=0.
IF (LVEG/=14 .OR. LVEG/=15) THEN ! excludes water and ocean
 REL_SRF_WAT_CON = MAX( MIN(1., P_SRF_WAT_CON/SRF_WAT_CON_MAX), 0. ) ! relative soil wetness
 REL_SRF_ICE_CON= MAX( MIN(1., P_SRF_ICE_CON/SRF_WAT_CON_MAX), 0.) ! relative soil iceness
END IF

PREC_G=0. ! Initialize water reaching the soil

! anticipates from DRAG definition of Z0_G_M, Z0_SN
IF (LSNOW)  THEN
        Z0_G_M = 0.005         !table A6 pag.290 garratt
! Evapotranspiration in the Soil-Plant-Atmosphere System, Viliam Novak, p.231 table 11.5
!        Z0_G_M = 0.05
        IF(LVEG==27 .OR. LVEG==28) Z0_G_M=0.01
        NU = 1.3432E-5 + 9.3571E-8 * (TEMP_A_STEP - DTK)     ! air kinematic viscosity
        Z0_SN=(0.135*NU/USTAR)+0.035*USTAR*USTAR/GRAVITY*(1+5*EXP(-((USTAR-0.18)/0.10)**2))
END IF

! calculates snow fractions
CALL SNOW_FRACTION(VEG_HEI_STEP,HEI_SN,LSNOW,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,
Z0_G_M,Z0_SN)

! evaluates drag coefficient
CALL DRAG(LSNOW,LVEG,PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,VEL_MOD_STEP,DRAG_H,DRAG_M,DRAG_V,
ZER_DIS_LEV,VEG_HEI_STEP,&
 P_TEMP_F_STEP,P_TEM_SOI(1),REYNOLDS,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,UAF,USTAR,
 Z0_F_H,&
 Z0_F_M,Z0_G_H,Z0_G_M,Z0_G_V,Z0_H,Z0_M,Z0_SN,Z0_V,H_OBS_ALL,ACT_VEG_HEI,H_OBS_WIND,ERR_ECO,
 Z0_M_ECO,Z0_H_ECO)

! calculates albedoes
CALL ALBEDO(ALB_F,ALB_FH,ALB_G,ALB_SD,ALB_SN,ALB_SNT,ALAT,ALB_TOT,ALON,ESSE,GAMMA,CLEG,&
 TIMESTEP_SEC,HEI_SN,ILOGALB,IORA,IMINUTI,L360DAYS_CALENDAR,LSNOW,JU_DAY,NUM_SOI_LAY,&
 PRECSN,SR_SOI,P_TEM_SOI,P_TEMP_SN,VEG_COV_STEP,SOL_ALFA,SOL_ZEN,COVER_SN)

! radiation
CALL RADIATION(ALB_F,ALB_G,ALB_SN,Q_SP_A_STEP,TEMP_A_STEP,CL_TOT_STEP,D_NR_F_D_TF,EPSF,&
 EPSG,HAZE_IND,LLONGWFLAG,NUM_SOI_LAY,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,RAD_L_G_D,RAD_L_G_U,
 RAD_L_D,NR_L_F_D,RAD_L_SN_F_D,&
 RAD_L_SN_F_U,NR_L_F_U,RAD_L_G_TOT_D,RAD_L_G_TOT_U,RAD_L_SN_G_D,RAD_L_SN_G_U,RAD_L_SN_U,
 NR_L_SN_D,NR_L_SN_U,RAD_L_U,RAD_L_F_D,RAD_L_F_U,RAD_L_FG_D,RAD_L_FG_U,&
 NR_A,NR_F,NR_G,NR_SN,RAD_S_D,RAD_S_F_D,RAD_S_SN_F_D,RAD_S_SN_F_U,RAD_S_F_U,RAD_S_G_D,
 RAD_S_SN_G_D,RAD_S_SN_G_U,RAD_S_G_U,&
 RAD_S_SN_D,RAD_S_SN_U,RAD_S_U,VEG_COV_STEP,COVER_SN_F,COVER_SN_G)

! calculation of new averaged variables related with soil moisture (to be used in
! the next step)
CALL AV_NS_MOI(PRES_A_STEP,BSOIL,CAPPAETA,CAPPAETA_S,CAPPAETAM,THEDIFM,CAPPAMIXF,CAPPAMIXG,&
 CAPPAUNO,CAPPASN,DEP_SOI,DENS_SN,DIFFSUM,DIFFSUMM,DIFFVT,DIFFVTM,ETA_S,MOIS_POT,&
 MOIS_POT_SAT,NUM_SOI_LAY,HEI_SN,VEG_HEI_STEP,INP_SOIL,LAI_STEP,LVAP,LVEG,LSNOW,SR_SOI,
 P_TEM_SOI,&
 ROC,ROCIVOL,RP_TOTAL,TORTUOS,P_ETA_W,SSR_FIE_CAPC,VEG_COV_STEP,P_ETA_I,NSO)

! canopy, soil and snow saturated specific humidities
CALL SPECUM (100.,P_TEMP_F_STEP,PRES_A_STEP,QSF,PIPPO,PIPPO) ! over canopy
CALL SPECUM (100.,P_TEM_SOI(1),PRES_A_STEP,QSG,PIPPO,PIPPO) ! over bare soil
IF (LSNOW)THEN
 CALL SPECUM (100.,P_TEMP_SN,PRES_A_STEP,QSSN,PIPPO,PIPPO) ! over snow
ELSE
 QSSN=0.
END IF
```

```
! soil surface relative humidity, from Philip (1957)
SRF_REL_HUM=1.
! water vapor is assumed saturated over water (but not over ice) and if atmosph.
! moisture is larger than saturated soil specific humidity
IF (LVEG/=14 .AND. LVEG/=15) THEN
 IF (QSG >= Q_A_F) SRF_REL_HUM=EXP(GRAVITY*MOIS_POT(1)/(WAT_VAP_CON*P_TEM_SOI(1)))
ENDIF

! in presence of snow, the wet canopy fraction is scaled to (1-COVER_SN_F)
! REL_LEA_WAT_CON ->0. when COVER_SN_F->1.
IF (LSNOW) THEN
 REL_LEA_WAT_CON=REL_LEA_WAT_CON*(1-COVER_SN_F)
END IF

! in presence of snow, the wet (or frozen) soil fraction is scaled to (1-COVER_SN_G)
! REL_SRF_WAT_CON ->0. when COVER_SN_G->1.
IF (LSNOW) THEN
 REL_SRF_WAT_CON=REL_SRF_WAT_CON*(1-COVER_SN_G)
 REL_SRF_ICE_CON=REL_SRF_ICE_CON*(1-COVER_SN_G)
END IF

! specific humidity at contact with soil
QSG_RH=QSG*(REL_SRF_WAT_CON+REL_SRF_ICE_CON) + QSG*SRF_REL_HUM*(1.-REL_SRF_WAT_CON-&
 REL_SRF_ICE_CON)

! snowless canopy and bare soil temperatures
TSURF= (1.-COVER_SN_G)*P_TEM_SOI(1)

! snowless bare soil specific humidities
QSURF  = (1.-COVER_SN_G)*QSG_RH

! photosynthesis
IF (VEG_COV_STEP > 0. .AND. LAI_STEP > 0. .AND. LPHOTO) THEN

 CALL SPECUM(100.,P_TEMP_F_STEP,PRES_A_STEP,QSAT_CANOPY,PIPPO,ES_CANOPY)
 E_CAN=ES_CANOPY*(Q_A_F/QSAT_CANOPY)*100.   ! [hPa]
 ES_CAN=ES_CANOPY*100.                  ! hPa-->Pa

 CALL PHOTOSYNTHESIS(GRO_CAR_ASS_RAT,ALB_FH,NET_CAR_ASS_RAT,PRES_A_STEP,CO2_CON_STEP,E_CAN,
 ES_CAN,HAJ,&
  HAV,HDJ,HDV,JOPT,LAI_STEP,LMEDLYN,LVEG,MRS,PTW,SSR_FIE_CAP,SO_SRAT_RO_DE,SSR_WI,RESIST_B,&
  RAD_S_F_D,RESIST_F_PHOTO,RES_RAT,TEMP_F_STEP,TOPTJ,TOPTV,PVMAX25,VOPT)

ELSE ! if no vegetation

 GRO_CAR_ASS_RAT=0.
 NET_CAR_ASS_RAT=0.
 RES_RAT=0.
 RESIST_F_PHOTO=ERR_VE

END IF

! evaluation of resistances and conductances
CALL RESIST(CO2_CON_STEP, COND_F_DRY, COND_F_WET, COND_G_DRY, COND_G_WET, COND_SRF_VAP,&
 CONDUCT_AH, CONDUCT_AM, CONDUCT_AV, CONDUCT_B, CONDUCT_D, CONDUCT_D_SN, CONDUCT_F,
 CONDUCT_SRF_SN,&
 D0_VEGPAR, DRAG_H, DRAG_M, DRAG_V, LAI_STEP, LVEG, NUM_SOI_LAY, SR_SOI, P_TEM_SOI,
 PRES_A_STEP,&
 Q_SP_A_STEP, RAD_S_D, REL_LEA_WAT_CON, RGL_NOILHAN, REL_SRF_WAT_CON, RESIST_AH, RESIST_AM,&
 RESIST_AV, RESIST_B, RESIST_D, RESIST_D_SN, RESIST_F, RESIST_F_MIN, RESIST_F_PHOTO,
 RESIST_LEAF,&
 RESIST_SRF, RESIST_SRF_SN, SO_SRAT_RO_DE, SSR_FIE_CAP, SSR_WI, TEMP_A_STEP, TEMP_ROOT, UAF,
 USTAR,&
 VEG_COV_STEP, VEG_HEI_STEP, VEL_MOD_STEP, Z0_F_H, Z0_G_H, Z0_SN, ZER_DIS_LEV)

! air-canopy temperature and humidity, see G94 (1994) eqq 8.32a,b
Q_A_F = (Q_SP_A_STEP*CONDUCT_AV/COND_SRF_VAP + (1.-COVER_SN)*VEG_COV_STEP*CONDUCT_F/
```

```
COND_SRF_VAP*QSF +&
    (1.-COVER_SN)*(1.-VEG_COV_STEP)*QSG_RH + COVER_SN*CONDUCT_SRF_SN/COND_SRF_VAP*QSSN) /&
    (CONDUCT_AV/COND_SRF_VAP + (1.-COVER_SN)*VEG_COV_STEP*CONDUCT_F/COND_SRF_VAP +&
    (1.-COVER_SN)*(1.-VEG_COV_STEP) + COVER_SN*CONDUCT_SRF_SN/COND_SRF_VAP)

TEMP_A_F = (TEMP_A_STEP*CONDUCT_AH/CONDUCT_D + (1.-COVER_SN)*VEG_COV_STEP*CONDUCT_B/CONDUCT_D*
P_TEMP_F_STEP +&
    (1.-COVER_SN)*(1.-VEG_COV_STEP)*P_TEM_SOI(1) + COVER_SN*CONDUCT_D_SN/CONDUCT_D*TEMP_SN) /&
    (CONDUCT_AH/CONDUCT_D + (1.-COVER_SN)*VEG_COV_STEP*CONDUCT_B/CONDUCT_D +&
    (1.-COVER_SN)*(1.-VEG_COV_STEP) + COVER_SN*CONDUCT_D_SN/CONDUCT_D)

! check if there is snow
RUNSN=0.
CALL ISTHERESNOW(PREC_STEP,PRES_A_STEP,TEMP_A_STEP,AIR_DENS,TIMESTEP_SEC,EVA_F_W,LEA_WAT_CON,
LEA_WAT_CON_MAX,&
 HM_SN,LSNOW,LSNOW1,LSNOW2,LSNOW3,P_LEA_WAT_CON,PREC_F,PREC_G,PREC_ON_SNOW,P_HM_SN,PRECSN,&
 SUM_PREC_SN,P_TEMP_F_STEP,RH_STEP,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,TWET)

! calculation of fluxes
CALL FLUXES(LSNOW,LVEG,NUM_SOI_LAY,PRES_A_STEP,Q_SP_A_STEP,TEMP_A_STEP,VEL_U_STEP,VEL_V_STEP,&
 CAPPAMIXF,CAPPAMIXG,RP_LAYER,AIR_DENS,DEP_SOI,EVA_A,EVA_F,EVA_F_D,EVA_F_W,EVA_G,EVA_G_D,
 EVA_G_W,&
 EVA_SN,EVA_SN_F,EVA_SN_G,EVATRA,LHF_A,LHF_F,LHF_F_D,LHF_F_W,LHF_G,LHF_G_D,LHF_G_W,LHF_SN,
 LHF_SN_F,&
 LHF_SN_G,SHF_A,SHF_F,SHF_G, SHF_SN,HEI_SN,SHF_SN_F,SHF_SN_G,CONDUCT_D_SN,CONDUCT_SRF_SN,
 PREC_F,&
 PREC_G,P_TEMP_F_STEP,P_TEM_SOI,P_TEMP_SN,Q_A_F,BAL_FLU_F,BAL_FLU_G,QSF,QSG,QSG_RH,CONF_SN_SRF
 ,&
 CONF_SN_F,CONF_SN_G,QSSN,NR_F,NR_G,NR_SN,CONDUCT_AH,CONDUCT_AM,CONDUCT_AV,CONDUCT_B,CONDUCT_D
 ,&
 COND_F_DRY,COND_F_WET,COND_G_DRY,COND_G_WET,VEG_COV_STEP,COVER_SN_F,COVER_SN_G,COVER_SN,&
 COND_SRF_VAP,TEMP_A_F,TEMP_ROOT,USTAR,UPWP,VPWP,H_OBS_ALL,Q_RAIN_F,Q_RAIN_G,JU_DAY,IORA,&
 IMINUTI,D_NR_F_D_TF,D_BAL_FLU_F_D_TF,BAL_FLU_SN,BAL_FLU_TOT)

IF (LSNOW) THEN
 CALL SNOW(ALB_SN,TEMP_A_STEP,DENS_SN,TIMESTEP_SEC,EVA_SN,LHF_SN,SHF_SN,HM_SN,HEI_SN,HW_SN,
 LSNOW,PREC_ON_SNOW,&
  P_HM_SN,P_HW_SN,PRECSN,P_TEMP_SN,BAL_FLU_F,BAL_FLU_G,CONF_SN_SRF,NR_SN,RUNSN,VEG_COV_STEP,
  TEMP_SN,TWET,BAL_FLU_SN)
ELSE
 HEI_SN=0.
 TEMP_SN=ERR_VE
 DENS_SN=ERR_VE
 ALB_SN=ERR_VE
END IF

! soil processes
CALL SOIL(NUM_SOI_LAY,PREC_STEP,C_DREN,CAPPAETA,CAPPAETAM,CAPPAETA_S,DIFFSUMM,DIFFVTM,&
 THEDIFM,DEP_SOI,EVA_A,LEA_WAT_CON,SRF_WAT_CON,SRF_ICE_CON,SRF_WAT_CON_MAX,ETA_S,INP_SOIL,&
 LVEG,SR_SOI,P_SRF_WAT_CON,P_SRF_ICE_CON,P_TEM_SOI,ROCIVOL,ESSE,TIMESTEP_SEC,EVA_G,&
 EVA_G_W,EVATRA,PREC_G,Q_A_F,SO_SRAT_RO_DE,QSG_RH,RUNSN,VEL_MOD_STEP,ALAT,LHF_G,SHF_A,&
 IORA,IMINUTI,BAL_FLU_G,RAD_L_D,RAD_L_U,RAD_S_D,RAD_S_U,TEM_SOI,MID_LAY_DEP,SUM_DRAI,SUM_RUNO,
 SUM_PREC,SUM_EVAP,&
 SUM_SOI_HEA_STO,SUM_DRAI_TMP,SUM_RUNO_TMP,SUM_MIN_LW_TMP,SUM_PREC_TMP,SUM_EVAP_TMP,ETA_W,
 P_ETA_W,ETA_I,P_ETA_I,&
 SSR_FIE_CAPC,VEG_COV_STEP,MOIS_POT,BSOIL,MOIS_POT_SAT)

! vegetation temperature
IF (VEG_COV_STEP > 0. .AND. LAI_STEP > 0.) THEN

 CALL TCAN(PRES_A_STEP,TEMP_A_STEP,VEG_HEA_CAP,AIR_DENS,TIMESTEP_SEC,EPSF,EPSG,ES_CANOPY,
 LTCAN_NEW,&
  PREC_F,P_TEMP_F_STEP,BAL_FLU_F,Q_A_F,QSAT_CANOPY,QSF,CONDUCT_B,COND_F_DRY,COND_F_WET,
  VEG_COV_STEP,&
  COVER_SN_F,TEMP_F_STEP,LHF_F,SHF_F,P_TEMP_SN,Q_RAIN_F,CONF_SN_F,TEMP_A_F,D_BAL_FLU_F_D_TF)

ELSE
```

```
 TEMP_F_STEP=TEMP_A_STEP

END IF

RETURN
END SUBROUTINE SURFPROC

!******************************************************************************
SUBROUTINE TCAN(PRES_A_STEP,TEMP_A_STEP,VEG_HEA_CAP,AIR_DENS,TIMESTEP_SEC,EPSF,EPSG,ES_CANOPY,
&
 LTCAN_NEW,PREC_F,P_TEMP_F_STEP,BAL_FLU_F,Q_A_F,QSAT_CANOPY,QSF,CONDUCT_B,COND_F_DRY,
 COND_F_WET,&
 VEG_COV_STEP,COVER_SN_F,TEMP_F_STEP,LHF_F,SHF_F,P_TEMP_SN,Q_RAIN_F,CONF_SN_F,TEMP_A_F,
 D_BAL_FLU_F_D_TF)
!******************************************************************************
! Calculates the canopy temperature
!******************************************************************************
! Calls: none
! - Authors: C. Cassardo, J.J. Ji, P. Ramieri (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL :: ADEN
REAL, INTENT(IN) :: TEMP_A_STEP
REAL, INTENT(IN) :: PRES_A_STEP
!REAL :: LAT_HEA_EVA
REAL, INTENT(IN) :: VEG_HEA_CAP
REAL,INTENT(IN) :: D_BAL_FLU_F_D_TF
REAL :: D_QF_ON_D_TF
REAL :: D_SPEC_UM
REAL, INTENT(IN) :: AIR_DENS
REAL, PARAMETER :: DELTA_TCAN_MAX = 60. ! maximum difference btw TCAN and Tair
REAL :: DP_TEMP_F_STEP
REAL, INTENT(IN) :: TIMESTEP_SEC
REAL, INTENT(IN) :: EPSF
REAL, INTENT(IN) :: EPSG
REAL, INTENT(OUT) :: ES_CANOPY
REAL, INTENT(IN) :: LHF_F
REAL, INTENT(IN) :: SHF_F
LOGICAL, INTENT(IN) :: LTCAN_NEW
REAL, INTENT(IN) :: PREC_F
REAL :: PTHF
REAL :: PIPPO
REAL, INTENT(IN) :: P_TEMP_F_STEP
REAL, INTENT(IN) :: P_TEMP_SN
REAL, INTENT(IN) :: Q_A_F
REAL, INTENT(IN) :: BAL_FLU_F
REAL, INTENT(OUT) :: QSAT_CANOPY
REAL, INTENT(IN) :: Q_RAIN_F
REAL, INTENT(IN) :: QSF
REAL, INTENT(IN) :: CONF_SN_F
REAL, INTENT(IN) :: CONDUCT_B
REAL, INTENT(IN) :: COND_F_DRY,COND_F_WET
REAL, INTENT(IN) :: COVER_SN_F
REAL, INTENT(IN) :: VEG_COV_STEP
REAL,PARAMETER :: SMIN = 1.E-5
REAL, INTENT(IN) :: TEMP_A_F
REAL :: THETA_F
REAL :: THETA_V
REAL, INTENT(OUT) :: TEMP_F_STEP

IF (LTCAN_NEW) THEN ! new scheme applied

! CALL SPECUM(100.,P_TEMP_F_STEP,PRES_A_STEP,QSAT_CANOPY,PIPPO,ES_CANOPY)
```

```
! CALL POTENTIAL(PRES_A_STEP,QSF,P_TEMP_F_STEP,PTHF,THETA_V)
! CALL POTENTIAL(PRES_A_STEP,QSF,TEMP_A_F,THETA_F,THETA_V)

!  D_QF_ON_D_TF = -4.*EPSF*BOZ*VEG_COV_STEP*P_TEMP_F_STEP**3 * (EPSG/(EPSF+EPSG-EPSF*EPSG) + &
!   (1.-COVER_SN_F) ) - LHF_F*D_SPEC_UM(P_TEMP_F_STEP,PRES_A_STEP)

!  IF (P_TEMP_SN/=ERR_VE .AND. ABS(P_TEMP_F_STEP-P_TEMP_SN)>SMIN)  D_QF_ON_D_TF =
D_QF_ON_D_TF - CONF_SN_F/&
!   (P_TEMP_F_STEP-P_TEMP_SN)

!  IF (ABS(THETA_F-PTHF)>SMIN) D_QF_ON_D_TF = D_QF_ON_D_TF - SHF_F/&
!   (PTHF-THETA_F)*PTHF/TEMP_F_STEP

!  IF (ABS(TEMP_A_STEP-TEMP_A_F)>SMIN) D_QF_ON_D_TF = D_QF_ON_D_TF - Q_RAIN_F/&
!   (TEMP_A_STEP-TEMP_A_F)

  !ADEN = VEG_HEA_CAP*100./TIMESTEP_SEC - 0.5 * D_QF_ON_D_TF
  !ADEN = VEG_HEA_CAP/TIMESTEP_SEC - 0.5 * D_QF_ON_D_TF
  ADEN = VEG_HEA_CAP/TIMESTEP_SEC - 0.5 * D_BAL_FLU_F_D_TF
ELSE ! Old scheme applied
 ADEN = VEG_HEA_CAP/TIMESTEP_SEC
END IF

IF (ABS(ADEN) > SMIN) THEN
 DP_TEMP_F_STEP =BAL_FLU_F/ADEN
ELSE
 DP_TEMP_F_STEP=0.
END IF

TEMP_F_STEP = P_TEMP_F_STEP + DP_TEMP_F_STEP

IF (ABS(TEMP_F_STEP-TEMP_A_STEP) > DELTA_TCAN_MAX) THEN
 print *,'PFT,DP_TEMP_F_STEP = ',P_TEMP_F_STEP,DP_TEMP_F_STEP
 WRITE (*,'(3(A,F6.1))') 'TEMP_F_STEP TOO LARGE: |',TEMP_F_STEP,'-',TEMP_A_STEP,'| > ',&
  DELTA_TCAN_MAX
 WRITE(99,'(3(A,F6.1))') 'TEMP_F_STEP TOO LARGE: |',TEMP_F_STEP,'-',TEMP_A_STEP,'| > ',&
  DELTA_TCAN_MAX
 TEMP_F_STEP = MIN(TEMP_A_STEP+DELTA_TCAN_MAX,MAX(TEMP_A_STEP-DELTA_TCAN_MAX,TEMP_F_STEP))
END IF

RETURN
END SUBROUTINE TCAN

!*****************************************************************************
SUBROUTINE TSOIL(NUM_SOI_LAY,VEL_MOD_STEP,ALAT,THEDIFM,DEP_SOI,TIMESTEP_SEC,LHF_G,SHF_A,LVEG,&
 IORA,IMINUTI,P_TEM_SOI,BAL_FLU_G,BAL_FLU_G_BOT,RAD_L_D,RAD_L_U,RAD_S_D,RAD_S_U,ROCIVOL,
 TEM_SOI,MID_LAY_DEP,&
 P_ETA_I,P_ETA_W,INP_SOIL,SSR_FIE_CAPC,ETA_S,VEG_COV_STEP)
!*****************************************************************************
! Calculates soil temperatures
!*****************************************************************************
! Calls: ABCI, ABCT, ABCW, DIAG3
! - Authors: C. Cassardo, P. Ramieri (10-Nov-1997)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

! Global variables declaration
INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY) :: AA
REAL,INTENT(IN) :: ALAT
REAL,INTENT(IN) :: VEL_MOD_STEP
REAL,DIMENSION(NUM_SOI_LAY) :: BB
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: THEDIFM
REAL,DIMENSION(NUM_SOI_LAY) :: CC
REAL :: DAYANGLE
```

```
REAL,DIMENSION(NUM_SOI_LAY) :: DD
REAL :: DDD
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL :: DFT
REAL,DIMENSION(NUM_SOI_LAY) :: DIFFWATER
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_S
REAL,INTENT(IN) :: LHF_G
REAL,INTENT(IN) :: SHF_A
INTEGER :: I
INTEGER,INTENT(IN) :: IMINUTI
INTEGER,INTENT(IN) :: INP_SOIL
INTEGER,INTENT(IN) :: IORA
INTEGER, INTENT(IN) :: LVEG
!INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,PARAMETER :: OMEGA=7.292E-5
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_ETA_W
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_TEM_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SSR_FIE_CAPC
REAL,INTENT(OUT) :: BAL_FLU_G_BOT
REAL,INTENT(IN) :: BAL_FLU_G
REAL,INTENT(IN) :: RAD_L_D
REAL,INTENT(IN) :: RAD_L_U
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ROCIVOL
REAL,DIMENSION(NUM_SOI_LAY) :: ROCIVOL1
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,EXTERNAL ::SIND
REAL,DIMENSION(NUM_SOI_LAY) :: SRHS
REAL :: SRHS0
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: TEM_SOI
REAL :: WAT_SUR_FLU
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP
REAL :: ZBOT

DFT=0.

! calculation of time to evaluate BAL_FLU_G_BOT under last layer; DAYANGLE is the hourly
! angle, defined in order than its sine has the maximum at local noon and its zeroes
! at 6 a.m. and p.m.
DAYANGLE = (REAL(IORA)+REAL(IMINUTI)/60.)/24.*360.-90.
DDD=SQRT(2.*THEDIFM(1)/OMEGA/ROCIVOL(1)*DEP_SOI(1))
ZBOT=MID_LAY_DEP(NUM_SOI_LAY)+DEP_SOI(NUM_SOI_LAY)/2.
BAL_FLU_G_BOT=BAL_FLU_G*EXP(-ZBOT/DDD)*SIND(DAYANGLE-ZBOT/DDD/2./PI*360.+45.)

TEM_SOI(:)=P_TEM_SOI(:)

IF (LVEG==14 .OR. LVEG==15) THEN ! if surface soil is water
 CALL ABCW(NUM_SOI_LAY,VEL_MOD_STEP,ALAT,DEP_SOI,DIFFWATER,TEM_SOI,MID_LAY_DEP,SRHS,SRHS0,
 WAT_SUR_FLU,&
  RAD_S_D,RAD_S_U,RAD_L_D,RAD_L_U,LHF_G,SHF_A,TIMESTEP_SEC,AA,BB,CC,DD)
 CALL DIAG3(AA,BB,CC,DD,TEM_SOI,NUM_SOI_LAY)
ELSE

 IF(LVEG==12) THEN ! ice
  CALL ABCI(NUM_SOI_LAY,RAD_S_D,RAD_S_U,TIMESTEP_SEC,TEM_SOI,DEP_SOI,DIFFWATER,SRHS,SRHS0,
  WAT_SUR_FLU,&
   RAD_L_D,RAD_L_U,LHF_G,SHF_A,MID_LAY_DEP,AA,BB,CC,DD)
  CALL DIAG3(AA,BB,CC,DD,TEM_SOI,NUM_SOI_LAY)
 ELSE

! evaluation of modified soil heat capacity (param. of Viterbo et al., and modified)
  DO I=1,NUM_SOI_LAY
   IF (INP_SOIL >= 3 .AND. INP_SOIL <= 5) THEN
    IF (TEM_SOI(I)>TFRZ1 .OR. TEM_SOI(I)<TFRZ2) THEN
```

```
      DFT=0.
     ELSE
      DFT=-PI*0.5*COS(PI*(TEM_SOI(I)-0.5*TFRZ1-0.5*TFRZ2)/(TFRZ1+TFRZ2))/(TFRZ1+TFRZ2)
     ENDIF
    ENDIF
! parameterization of Viterbo et al.
    IF (INP_SOIL == 3 .OR. INP_SOIL == 4) THEN
     ROCIVOL1(I)=ROCIVOL(I)-ROW*CLF*DFT*VEG_COV_STEP*SSR_FIE_CAPC(I)*ETA_S(I)*DEP_SOI(I)
    ELSE IF (INP_SOIL == 5) THEN
     ROCIVOL1(I)=ROCIVOL(I)-ROW*CLF*DFT*(P_ETA_W(I)+P_ETA_I(I)-ETAMIN)*DEP_SOI(I)
    ELSE
     ROCIVOL1(I)=ROCIVOL(I)
    ENDIF
   ENDDO

   CALL ABCT(NUM_SOI_LAY,TIMESTEP_SEC,BAL_FLU_G,BAL_FLU_G_BOT,THEDIFM,DEP_SOI,ROCIVOL1,&
    TEM_SOI,AA,BB,CC,DD)
   CALL DIAG3(AA,BB,CC,DD,TEM_SOI,NUM_SOI_LAY)

  END IF

END IF

RETURN
END SUBROUTINE TSOIL

!*****************************************************************************
SUBROUTINE USOIL(ESSE,C_DREN,CAPPAETA,CAPPAETAM,CAPPAETA_S,DIFFSUMM,DIFFVTM,DEP_SOI,&
 TIMESTEP_SEC,EVA_G,EVA_G_W,SRF_WAT_CON,SRF_ICE_CON,SRF_WAT_CON_MAX,ETA_I,ETA_S,&
 ETA_W,EVATRA,MOIS_POT,INP_SOIL,NUM_SOI_LAY,P_ETA_W,P_ETA_I,P_SRF_WAT_CON,P_SRF_ICE_CON,PREC_G
 ,&
 SR_SOI,P_TEM_SOI,TEM_SOI,Q_A_F,SO_SRAT_RO_DE,QSG_RH,ROCIVOL,RUNG,RUNSN,RUNU,VEG_COV_STEP,&
 SSR_FIE_CAPC,BSOIL,MOIS_POT_SAT)
!*****************************************************************************
! Calculates soil moisture, runoff and drainage in the soil layers
!*****************************************************************************
! Calls: ABCQ, DIAG3
! - Authors: C. Cassardo, P. Ramieri (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,DIMENSION(NUM_SOI_LAY):: AA
REAL,DIMENSION(NUM_SOI_LAY):: BB
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: BSOIL
REAL,INTENT(IN) :: C_DREN
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETAM
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: CAPPAETA_S
REAL,DIMENSION(NUM_SOI_LAY):: CC
REAL,EXTERNAL :: COSD
REAL,DIMENSION(NUM_SOI_LAY):: DD
REAL :: DE
REAL :: DELTAWI
REAL :: DELTAWIMAX
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL :: DETA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DIFFSUMM
REAL,DIMENSION(NUM_SOI_LAY) :: DIFFSUMM1
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DIFFVTM
REAL,DIMENSION(NUM_SOI_LAY) :: DIFFVTM1
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_G
REAL,INTENT(IN) :: EVA_G_W
REAL,INTENT(INOUT) :: SRF_WAT_CON
```

```
REAL,INTENT(INOUT) :: SRF_ICE_CON
REAL,INTENT(IN) :: SRF_WAT_CON_MAX
REAL,INTENT(IN) :: ESSE
REAL :: ETA_TOT
REAL :: ETA_TOT_CORRECT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_I
REAL :: ETA_I_TOT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ETA_S
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: ETA_W
REAL,DIMENSION(NUM_SOI_LAY) :: ETA_W1
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: EVATRA
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MOIS_POT
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MOIS_POT_SAT
REAL :: FT
REAL :: GAMMARUNOFF
INTEGER :: I
!???temporanea
INTEGER, PARAMETER :: INF_CHOICE = 3
!!!!!claudiooooo era 2
INTEGER :: J
INTEGER,INTENT(IN) :: INP_SOIL
!INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL :: P_INFILTR
REAL :: P_INFILTR_MAX
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_I
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_ETA_W
REAL,INTENT(INOUT) :: P_SRF_WAT_CON
REAL,INTENT(INOUT) :: P_SRF_ICE_CON
REAL :: PG_EFF
REAL,INTENT(IN) :: PREC_G
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: SR_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(INOUT) :: P_TEM_SOI
REAL,INTENT(IN) :: Q_A_F
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: SSR_FIE_CAPC
REAL,INTENT(IN) :: QSG_RH
REAL,INTENT(IN) :: SO_SRAT_RO_DE
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: ROCIVOL
REAL,INTENT(OUT) :: RUNG
REAL,DIMENSION(NUM_SOI_LAY):: RUNLEV
REAL,INTENT(IN) :: RUNSN
REAL,INTENT(OUT) :: RUNU
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,EXTERNAL :: SIND
REAL,PARAMETER :: T0 = 273.15
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: TEM_SOI
REAL :: X

GAMMARUNOFF=0.
ETA_TOT_CORRECT=0.

! initialization of intermediate runoff to zero
DO I=1,NUM_SOI_LAY
 RUNLEV(I)=0.
END DO

DO I=1,NUM_SOI_LAY

! Schrodin and Heise calculation
 IF (INP_SOIL == 1 .OR. INP_SOIL == 2) THEN ! Schrodin et al., 2001

! the Schrodin and Heise parameterization applies ONLY if TEM_SOI crosses 0^C
  IF ( (P_TEM_SOI(I) <= T0 .AND. TEM_SOI(I) >= T0) .OR. (P_TEM_SOI(I) >= T0 .AND. &
   TEM_SOI(I) <= T0) ) THEN
   DELTAWIMAX = -ROCIVOL(I)/(CLF*ROW)*(TEM_SOI(I)-T0)
   IF (DELTAWIMAX.LT.0.) THEN    ! melting
    DELTAWI = -MIN(-DELTAWIMAX,P_ETA_I(I)*DEP_SOI(I))
   ELSE ! freezing
```

```
    DELTAWI = MIN(DELTAWIMAX,(P_ETA_W(I)-ETAMIN-P_ETA_I(I))*DEP_SOI(I))
   END IF
   TEM_SOI(I) = TEM_SOI(I)+DELTAWI/ROCIVOL(I)*CLF*ROW

   IF (INP_SOIL == 2) THEN
! update liquid water and ice content by DELTAWI - division by DEP_SOI(I) to take into
! account that dimension of DELTAWI is [m]
    DETA=DELTAWI/DEP_SOI(I)
! check that ETA_W(I) is not lower than ETAMIN
    IF ( (P_ETA_W(I)-DETA) < ETAMIN)THEN
     DETA=P_ETA_W(I)-ETAMIN
    END IF
    ETA_W(I) = P_ETA_W(I) - DETA
    ETA_I(I) = P_ETA_I(I) + DETA
   END IF

  ENDIF

 ELSE IF (INP_SOIL == 0 .OR. INP_SOIL == 3) THEN
! no freezing or only thermal part of Viterbo et al.
  ETA_I(I) = 0.
  ETA_W(I) = P_ETA_W(I)

 ELSE IF (INP_SOIL >= 4 .AND. INP_SOIL <= 5) THEN
! Viterbo et al. (1999) parametrization and its modifications

  IF (P_TEM_SOI(I) > TFRZ1) THEN
   FT=0.
  ELSE IF (P_TEM_SOI(I) < TFRZ2) THEN
   FT=1
  ELSE
   FT=0.5*(1.-SIN((PI*(P_TEM_SOI(I)-0.5*TFRZ1-0.5*TFRZ2))/(TFRZ1-TFRZ2)))
  ENDIF

  IF (INP_SOIL == 4) THEN
! Viterbo et al. (1999) parametrization

   DETA = FT*VEG_COV_STEP*SSR_FIE_CAPC(I)*ETA_S(I)
   IF ( (P_ETA_W(I)-(DETA-P_ETA_I(I))) < ETAMIN) DETA=P_ETA_W(I)-ETAMIN+P_ETA_I(I)
   ETA_W(I)=P_ETA_W(I)-(DETA-P_ETA_I(I))
   ETA_I(I)=DETA

  ELSE IF (INP_SOIL == 5) THEN
! Viterbo et al. modified parametrization

   DETA=MAX(0.,FT*(P_ETA_W(I)+P_ETA_I(I)))
   IF ( (P_ETA_W(I)-(DETA-P_ETA_I(I))) < ETAMIN) DETA=P_ETA_W(I)-ETAMIN+P_ETA_I(I)
   ETA_W(I)=P_ETA_W(I)-(DETA-P_ETA_I(I))
   ETA_I(I)=DETA

  ENDIF

! check lower limit of ETA_W: ETA_W cannot be lower than ETAMIN
  IF (ETA_W(I) < ETAMIN) THEN
   ETA_I(I)=ETA_I(I)-(ETAMIN-ETA_W(I))
   ETA_W(I)=ETAMIN
  ENDIF

 ENDIF

ENDDO

! Calculation of effective precipitation. The effective precipitation PREC_G1 is
! calculated as PREC_G*COSD(ESSE) taking into account that the effective area
! affected by rain is larger by a factor COSD(ESSE) with respect to the horiz area
! (the factor 0.1 has been imposed by Cassardo without verification)
PG_EFF=(PREC_G+RUNSN/TIMESTEP_SEC)*MAX(0.,(COSD(ESSE)-0.1*SIND(ESSE))) ! m/s
```

```
!???meglio inserire un'infiltrazione minima

! Surface runoff contribution due to inclinated surface
RUNLEV(1)=PREC_G+RUNSN/TIMESTEP_SEC-PG_EFF

!Calculation of wet (or frozen) soil fraction.
CALL WATER_MASS_GROUND(TIMESTEP_SEC,EVA_G,EVA_G_W,SRF_WAT_CON,SRF_ICE_CON,&
 SRF_WAT_CON_MAX,P_SRF_WAT_CON,P_SRF_ICE_CON,PG_EFF,Q_A_F,QSG_RH,P_ETA_I(1),ETA_I(1))

! calculation of infiltration
! Three options:
! 1) MAX infiltration is calculated using the scheme of Boone and Wetzel (1996),
!    then real infiltration is choosen as the min between PG_EFF and P_INFILTR_MAX
! 2) same as 1) but NCAR model scheme is used for MAX infiltration
! 3) surface RUNOFF is calculated, then effective precipitation is evaluated on the
! basis of runoff

P_INFILTR=0. ! m/s
P_INFILTR_MAX=0. ! m/s

IF ((PREC_G+RUNSN)>0.) THEN ! there is water input

 IF (INF_CHOICE == 1) THEN
! calculation of MAX infiltration according to the Boone and Wetzel (1996) scheme in
! which DELTAT has been set to 1 sec (Cassardo suggestion to avoid too high values
! of infiltration in case of dry soil)
  P_INFILTR_MAX=CAPPAETA_S(1)-2.*ETA_S(1)*(SR_SOI(1)-1.)*SQRT(-BSOIL(1)*MOIS_POT(1)*&
   CAPPAETA_S(1)/ETA_S(1)/PI) ! m/s
 ENDIF

 IF (INF_CHOICE == 2) THEN
! scheme of NCAR model
  P_INFILTR_MAX=CAPPAETA_S(1)*(1.-MOIS_POT_SAT(1)/0.5/DEP_SOI(1)*BSOIL(1)*(1.-(P_ETA_W(1)/&
   ((ETA_S(1)-ETA_I(1))))))
 ENDIF

 IF (INF_CHOICE == 1 .OR. INF_CHOICE == 2) THEN
! infiltration is choosen as the min value between PG_EFF and P_INFILTR_MAX
  P_INFILTR=MIN(P_INFILTR_MAX,PG_EFF) ! m/s
! Surface runoff contribution due to the difference between the effective
! precipitation and the infiltration
  RUNLEV(1)=RUNLEV(1)+MAX(0.,(PG_EFF-P_INFILTR_MAX)*TIMESTEP_SEC) ! m
 ENDIF

 IF (INF_CHOICE == 3) THEN
! surface RUNOFF is calculated, then effective precipitation is evaluated on the
! basis of runoff taken from BATS: first calculates runoff and then infiltration;
! power is decided according to presence of ice, which may reduce infiltration.
! In BATS formula there is the mean soil moisture in the fist two layers (on 3), in
! which there are roots. Here we use SO_SRAT_RO_DE which is the mean root zone soil
! moisture zone

!??? rivedi tutto e inserisci scelta di INF_CHOICE
! Since in Viterbo parametrization there is ice production if T< 1C, the dependence
! of infiltration by soil temperature is correct only for Schrodin parametrization,
! and not for all Viterbo parametrizations (see Bonanno (2009) pp 99-100)
  IF (ETA_I(1) < 0.001 ) GAMMARUNOFF=SO_SRAT_RO_DE**4
  IF (ETA_I(1) >= 0.001) GAMMARUNOFF=SO_SRAT_RO_DE

! in the original BATS formulations there was PREC_G minus evaporation
  RUNLEV(1)=RUNLEV(1)+GAMMARUNOFF*PG_EFF*TIMESTEP_SEC ! in m

! evaluation of infiltration
  P_INFILTR=PG_EFF*(1.-GAMMARUNOFF) ! in m/s

 ENDIF
```

```
 IF (INP_SOIL == 3 .OR. INP_SOIL == 4 .OR. INP_SOIL == 5) THEN
  IF (P_TEM_SOI(1)<TFRZ2) P_INFILTR=0. !no infiltration for frozen soil
 ENDIF

 IF (P_TEM_SOI(1)<DTK) P_INFILTR=0. !no infiltration for frozen soil

ENDIF

! the following instructions involving ETA_I_TOT impose the water mass conservation
! during freezing. ETA_I_TOT is the total soil volumetric ice content
ETA_I_TOT=0.
DO I=1,NUM_SOI_LAY
 ETA_I_TOT = ETA_I_TOT + ETA_I(I)
END DO

IF(ETA_I_TOT >= 0.01)THEN

! Solution of soil humidity equation whit hydraulic diffusivity related to liquid
! part of water in soil setted to zero

 ETA_W1=ETA_W
 DIFFSUMM1=0.
 DIFFVTM1=0.

 CALL ABCQ(NUM_SOI_LAY,AA,BB,C_DREN,CAPPAETA,CAPPAETAM,CC,DD,DIFFSUMM1,DIFFVTM1,&
  DEP_SOI,TIMESTEP_SEC,EVA_G,EVATRA,P_INFILTR,ETA_W1)

 CALL DIAG3(AA,BB,CC,DD,ETA_W1,NUM_SOI_LAY)

! Evaluation of variable ETA_TOT_CORRECT. This variable is not affected by the
! problem of non conservation of water mass in soil during freezing

 ETA_TOT_CORRECT=0.
 DO I=1,NUM_SOI_LAY
  ETA_TOT_CORRECT = ETA_TOT_CORRECT + ETA_W1(I)+ETA_I(I)
 END DO

END IF

! evaluation of new soil moisture content ETA_W
CALL ABCQ(NUM_SOI_LAY,AA,BB,C_DREN,CAPPAETA,CAPPAETAM,CC,DD,DIFFSUMM,DIFFVTM,&
 DEP_SOI,TIMESTEP_SEC,EVA_G,EVATRA,P_INFILTR,ETA_W)
CALL DIAG3(AA,BB,CC,DD,ETA_W,NUM_SOI_LAY)

!Correction of ETA_W in soil layers where is a significant ice production
!(ETA_I>0.01) Bonanno (2009) pp 97-98

!IF(ETA_I_TOT>=0.01)THEN
!
! DO J=1,30
!
!! Evaluation of variable ETA_TOT. This variable is affected by the problem of non
!! conservation of water mass in soil during freezing
!
!  ETA_TOT=0.
!  DO I=1,NUM_SOI_LAY
!   ETA_TOT = ETA_TOT + ETA_W(I)+ETA_I(I)
!  END DO
!
!  DO I=1,NUM_SOI_LAY
!   IF(ETA_I(I)>=0.01) ETA_W(I)=ETA_W(I)*(ETA_TOT_CORRECT/ETA_TOT)
!  END DO
!
! END DO
!
!END IF
!!End for Riccardo Bonanno's instructions
```

```
! determination of runoff due to the exceeding of the maximum value of soil moisture
DO I=1,NUM_SOI_LAY
 IF ( (ETA_W(I)+ETA_I(I)) > ETA_S(I) ) THEN
  DE=ETA_W(I)+ETA_I(I)-ETA_S(I)
  X=ETA_W(I) ! store previous value
  ETA_W(I)=MAX(ETAMIN,(ETA_W(I)-DE))
  RUNLEV(I)=RUNLEV(I)+MAX(0.,(X-ETA_W(I)))*DEP_SOI(I)
 ENDIF
ENDDO

DO I=1,NUM_SOI_LAY

!check for minimum values
 IF ((ETA_W(I)+ETA_I(I)) < ETAMIN) THEN
  ETA_W(I) = ETAMIN
 ENDIF
 SR_SOI(I)=ETA_W(I)/ETA_S(I)

END DO

! Total surface runoff, which includes also the runoff affecting first soil layer
RUNU=RUNLEV(1) ! [m]

! Total bottom drainage - all terms involving interfacial runoff, saturation runoff
! and drainage are summed into total drainage
RUNG=0.
DO I=2,NUM_SOI_LAY
 RUNG=RUNG+RUNLEV(I) ! [m]
ENDDO
RUNG=RUNG+C_DREN*CAPPAETA(NUM_SOI_LAY)*TIMESTEP_SEC ! [m]

RETURN
END SUBROUTINE USOIL

!****************************************************************************
SUBROUTINE VEGPAR_FIX(ALB_FH,D0_VEGPAR,ROOT_DEP,EPSF,HAJ,HAV,HDJ,HDV,VEG_HEI_STEP,JOPT,LVEG,&
 MRS,NTIPOSUOLO,PTW,RGL_NOILHAN,RESIST_F_MIN,TOPTJ,TOPTV,PVMAX25,VOPT,VEGSHR)
!****************************************************************************
! This routine calculates some soil and vegetation parameters not depending on
! yearly cycle.
!****************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Sep-1994)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
USE VEGET_PARAM
IMPLICIT NONE

REAL,INTENT(OUT) :: ALB_FH ! shortwave albedo (<700 nm) [-]
REAL,INTENT(OUT) :: D0_VEGPAR ! Minor leaf characteristic dimension [m]
REAL,INTENT(OUT) :: ROOT_DEP ! Vegetation root depth [m]
REAL,INTENT(OUT) :: EPSF ! Emissivity of vegetation [-], from lw albedo (>700 nm)
REAL,INTENT(OUT) :: HAV ! Activation energy of Rubisco carboxylation [J/mol]
REAL,INTENT(OUT) :: HAJ ! Activation energy of the e- transport reactions [J/mol]
REAL,INTENT(OUT) :: HDV ! Deactivation energy of Rubisco carboxylation [J/mol]
REAL,INTENT(OUT) :: HDJ ! Deactivation energy of the e- transport reaction [J/mol]
REAL,INTENT(OUT) :: VEG_HEI_STEP ! Vegetation heigth [m], from BATS z0=0.13*hcanopy
REAL,INTENT(OUT) :: JOPT ! Rate of e- transport at opt temperature [uml m-2 s-1]
INTEGER,INTENT(IN) :: LVEG ! Code for vegetation type (1 <-> NVEG) [-]
REAL,INTENT(OUT) :: MRS !Coefficient m in the computation of stomatal resistance
INTEGER,INTENT(OUT) :: NTIPOSUOLO ! Code for soil texture type [-]
INTEGER,INTENT(OUT) :: PTW ! C4 patway code
REAL,INTENT(OUT) :: RGL_NOILHAN ! Parameter (see Noilhan, 89) [W/m2], used in stomatal resist
REAL,INTENT(OUT) :: RESIST_F_MIN ! Minimum stomatal resistance [s/m]
REAL,INTENT(OUT) :: TOPTJ ! Optimum temperature of the electron transport chain [K]
```

```
REAL,INTENT(OUT) :: TOPTV ! Optimum temperature of Rubisco carboxylation [K]
REAL,INTENT(OUT) :: PVMAX25 ! Max rate of Rubisco carboxylat at 25^C  [uml m-2 s-1]
LOGICAL,INTENT(OUT) :: VEGSHR ! flag to allow vegetation shrinking during winter season
REAL,INTENT(OUT) :: VOPT

! Typical leaf dimention (follows D94)
D0_VEGPAR=1./(SQRTDI(LVEG)*SQRTDI(LVEG))

! Minimum stomatal resistance
RESIST_F_MIN=RSMIN(LVEG)

! Soil types
NTIPOSUOLO = IEXSOL(LVEG)

! Albedo of vegetation
ALB_FH = ALB_VGS(LVEG)

! Emissivity of vegetation, assumed as (1.00 - longwave albedo)
EPSF = 1. - ALB_VGL(LVEG)

! parameter used in stomatal resistance
RGL_NOILHAN=RGL_NOILHAN1(LVEG)

! vegetation heigth
VEG_HEI_STEP = HCANOPY(LVEG)

! root depth - rule of thumb for generic vegetation: root depth equal to half vegetation
heigth, with minimum value taken at least 15 cm
ROOT_DEP = MAX(0.15,VEG_HEI_STEP/2.)

IF(LVEG==26)THEN    !vineyard
 ROOT_DEP=3.
END IF

! activation/deactivation energies in photosynthesis reactions
HAV=HAV1(LVEG)
HAJ=HAJ1(LVEG)
HDV=200000.
HDJ=HDJ1(LVEG)

! otimum temperature of photosynthesis reactions
JOPT=JOPT1(LVEG)
VOPT=VOPT1(LVEG)

! rate of photosynthesis reaction at the optimum temperature
TOPTJ=ERR_VE
TOPTV=ERR_VE
IF (TOPTJ1(LVEG) /= ERR_VE) TOPTJ=TOPTJ1(LVEG)+DTK
IF (TOPTV1(LVEG) /= ERR_VE) TOPTV=TOPTV1(LVEG)+DTK

! C4 pathway code
PTW = PTW1(LVEG)

! rate of Rubisco carboxylation at 25 C
IF (PTW==1) THEN  !C3 pathway
 PVMAX25 = PVMAX25I_1(LVEG)
 MRS = MRS1_1(LVEG)
ELSE IF (PTW==0) THEN  !C4 pathway
 PVMAX25 = PVMAX25I_0(LVEG)
 MRS = MRS1_0(LVEG)
END IF

! flag for vegetation shrinking during winter season
VEGSHR = VEGSHR1(LVEG)

RETURN
END SUBROUTINE VEGPAR_FIX
```

```
!******************************************************************************
SUBROUTINE VEGPAR_VAR(LVEG,TEMP_ROOT,VEG_COV_NOECO,LAI_STEP_NOECO)
!******************************************************************************
! This routine calculates some vegetation parameters depending on yearly cycle.
! Derived from BATS1e. For the vegetation list see VEGPAR_FIX.
!******************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Sep-1995)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
USE VEGET_PARAM
IMPLICIT NONE

REAL :: AMXTEM
REAL,INTENT(OUT) :: LAI_STEP_NOECO   ! Leaf area index [-]
INTEGER,INTENT(IN) :: LVEG          ! Code for vegetation type [-]
REAL :: SFAC
REAL,INTENT(OUT) :: VEG_COV_NOECO    ! Vegetation cover [%]
REAL,INTENT(IN) :: TEMP_ROOT           ! Ground surface temperature [K]

! Seasonal dependance func
AMXTEM = MAX(DTK+25.-TEMP_ROOT,0.)
SFAC = 1.-MAX(0.,1.-0.0016*AMXTEM**2)

! Vegetation cover

IF (TEMP_ROOT > DTK+25.) VEG_COV_NOECO = VEGC(LVEG)
IF (TEMP_ROOT <= (DTK+25.) .AND. TEMP_ROOT > DTK ) VEG_COV_NOECO = &
 VEGC(LVEG)-SEASF(LVEG)*SFAC
IF (TEMP_ROOT <= DTK) VEG_COV_NOECO = VEGC(LVEG)-SEASF(LVEG)

! Leaf area index
LAI_STEP_NOECO = XLA(LVEG)+XLAI0(LVEG)*(1.-SFAC)

RETURN
END SUBROUTINE VEGPAR_VAR

!******************************************************************************
REAL FUNCTION X_WEI_MEA(X1,X2,IC,ERR_VE)
!******************************************************************************
! Calculates averaged quantities between 2 adiacent layers. The function is selected
! according to the value of ICOD.
! If ICOD<0 the derivative of X_WEI_MEA with respect to X2 is calculated.
!  ABS(ICOD)    type        function
!   1       logarythmic     X_WEI_MEA = exp { [ ln(x1)+ln(x2) ] / 2 }
!   2       linear          X_WEI_MEA = (x1+x2) / 2
!   3       minimum         X_WEI_MEA = MIN(x1,x2)
!   4       maximum         X_WEI_MEA = MAX(x1,x2)
!  -1       logaryth deriv  X_WEI_MEA = +/-exp { [ ln(x1)+ln(x2) ] / 2 } / (2 * x2)
!  -2       linear deriv    X_WEI_MEA = 1/2
!  -3       minimum deriv   X_WEI_MEA = 0
!  -4       maximum deriv   X_WEI_MEA = 0
!******************************************************************************
! Calls: none
! - Author: C. Cassardo (27-May-1994)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
IMPLICIT NONE

REAL,INTENT(IN) :: ERR_VE
INTEGER,INTENT(IN) :: IC
INTEGER :: ICOD
REAL,INTENT(IN) :: X1
REAL,INTENT(IN) :: X2
```

```
X_WEI_MEA = ERR_VE
IF (X1==ERR_VE .OR. X2==ERR_VE) RETURN

ICOD=IC
IF (ICOD==1 .AND. (X1==0. .OR. X2==0.)) ICOD=2 ! impossible to have logarithmic mean

IF (ICOD==1) THEN ! logarythmic
 X_WEI_MEA = EXP(0.5*(LOG(ABS(X1))+LOG(ABS(X2))))
 IF (X1<0.) X_WEI_MEA = SIGN(X_WEI_MEA,X1)
ENDIF

IF (ICOD==2) THEN ! linear interpolation
 X_WEI_MEA = 0.5 * (X1+X2)
ENDIF

IF (ICOD==3) THEN ! minimum
 X_WEI_MEA = MIN (X1,X2)
ENDIF

IF (ICOD==4) THEN ! maximum
 X_WEI_MEA = MAX (X1,X2)
END IF

IF (ICOD==-1) THEN ! logarythmic derivative
 X_WEI_MEA = EXP(0.5*(LOG(ABS(X1))+LOG(ABS(X2))))
 IF (X1<0.) X_WEI_MEA = SIGN(X_WEI_MEA,X1)
 IF (ICOD==-1) X_WEI_MEA = X_WEI_MEA * X2 * 0.5
ENDIF

IF (ICOD==-2) THEN ! linear derivative
 X_WEI_MEA = 0.5
ENDIF

IF (ICOD==-3 .OR. ICOD==-4) THEN ! derivative of minumum, maximum
 X_WEI_MEA = 0.
END IF

RETURN
END FUNCTION X_WEI_MEA

!*****************************************************************************
SUBROUTINE WATERDIFF(NUM_SOI_LAY,VEL_MOD_STEP,ALAT,DEP_SOI,DIFFWATER,P_TEM_SOI,MID_LAY_DEP )

!???eventually DA eliminARE

!*****************************************************************************
! This subroutine is to calculate the sum of the molecular and eddy diffusion coefficients
(m*m/s)
! TH_CO_W is the thermal conductivity of water (W m-1 K-1)
! ROCI_W is the heat capacity of water (J m-3 K-1)
!*****************************************************************************
! Calls: none
! - Author : Minwei Qian
! - Date   : 28-11-2002
! - Last revision : 06-04-2004  M.Trevisan (f90 translation)
!*****************************************************************************
USE CONSTANTS
IMPLICIT NONE

INTEGER,INTENT(IN) :: NUM_SOI_LAY
REAL,INTENT(IN) :: ALAT
REAL,DIMENSION(NUM_SOI_LAY) :: ANN
REAL,INTENT(IN) :: VEL_MOD_STEP
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: DEP_SOI
REAL,DIMENSION(NUM_SOI_LAY),INTENT(OUT) :: DIFFWATER
INTEGER :: I
!INTEGER,INTENT(IN) :: NUM_SOI_LAY
```

```
REAL :: P0
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: P_TEM_SOI
REAL,DIMENSION(NUM_SOI_LAY)  :: ROUWATER
REAL,EXTERNAL :: SIND
REAL :: SXYZ
REAL,DIMENSION(NUM_SOI_LAY),INTENT(IN) :: MID_LAY_DEP
REAL,DIMENSION(NUM_SOI_LAY) :: RIC_NUM
REAL :: WEDDY
REAL :: WKSTAR
REAL :: WSTAR

WSTAR=0.0012*VEL_MOD_STEP
WKSTAR=6.6*SQRT(ABS(SIND(ALAT)))*VEL_MOD_STEP**(-1.84)

DO I=1, NUM_SOI_LAY
 ROUWATER(I)=(1.-1.9549e-5*(ABS(P_TEM_SOI(I)-277.))**1.68)*1000.
END DO

DO I=1, NUM_SOI_LAY-1
 ANN(I)=-GRAVITY/ROUWATER(I)*(ROUWATER(I+1)-ROUWATER(I))/DEP_SOI(I)
END DO

DO I=1, NUM_SOI_LAY-1
 SXYZ=1.+40.*ANN(I)*VON_KARMAN*VON_KARMAN*MID_LAY_DEP(I)*MID_LAY_DEP(I)&
  /(WSTAR*WSTAR*EXP(-2.*WKSTAR*MID_LAY_DEP(I)))
 RIC_NUM(I)=0.
END DO

P0=1.
DO I=1, NUM_SOI_LAY-1
 WEDDY=VON_KARMAN*WSTAR*MID_LAY_DEP(I)*EXP(-WKSTAR*MID_LAY_DEP(I))&
  /(P0*(1.+37.*RIC_NUM(I)**2))
 IF (P_TEM_SOI(1) < 0.0) THEN
  DIFFWATER(I)=TH_CO_I/ROCI_I ! here it is better to use thermal conductivity of ice
 ELSE
  DIFFWATER(I)=TH_CO_W/ROCI_W+WEDDY
 ENDIF
ENDDO

RETURN
END SUBROUTINE WATERDIFF

!****************************************************************************
SUBROUTINE WATER_MASS(PREC_STEP,TIMESTEP_SEC,EVA_F_W,LEA_WAT_CON,LEA_WAT_CON_MAX,LSNOW2,LSNOW3
,&
 P_LEA_WAT_CON,PREC_F,PREC_G,PREC_ON_SNOW,PRECSN,SUM_PREC_SN,P_TEMP_F_STEP,VEG_COV_STEP,
 COVER_SN_F,&
 COVER_SN_G,COVER_SN)
!****************************************************************************
! Calculates mass and water balance on the canopy
!****************************************************************************
! Calls: none
! - Author: C. Cassardo (27-May-1994)
! - Last revision: C. Cassardo (14-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: PREC_STEP
REAL :: VEG_DRAIN
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_F_W
REAL,INTENT(INOUT) :: LEA_WAT_CON
REAL,INTENT(IN) :: LEA_WAT_CON_MAX
LOGICAL,INTENT(IN) :: LSNOW2
LOGICAL,INTENT(OUT) :: LSNOW3
REAL,INTENT(IN) :: P_LEA_WAT_CON
```

```
REAL,INTENT(OUT) :: PREC_F
REAL,INTENT(INOUT) :: PREC_G
REAL,INTENT(OUT) :: PREC_ON_SNOW
REAL,INTENT(OUT) :: PRECSN
REAL,INTENT(INOUT) :: SUM_PREC_SN
REAL,INTENT(IN) :: P_TEMP_F_STEP
REAL,INTENT(IN) :: VEG_COV_STEP
REAL,INTENT(IN) :: COVER_SN_F
REAL,INTENT(IN) :: COVER_SN_G
REAL,INTENT(IN) :: COVER_SN

! setting to zero or false initial values of variables
PREC_F=0. ! canopy rainfall
PRECSN=0. ! snowfall
VEG_DRAIN=0. ! canopy rain drainage
LSNOW3=.false.
PREC_ON_SNOW=0. ! snow rainfall

! balance equation
IF (VEG_COV_STEP==0.) THEN ! no canopy: thus no water and rain over canopy
 LEA_WAT_CON=0.
ELSE

! if there is rainfall, the precipitation on the canopy PREC_F is calculated
 IF ((.NOT.LSNOW2).AND.PREC_STEP>0.) THEN ! if there is prec and it is snow
  PREC_F=(PREC_STEP/3.6E6)*VEG_COV_STEP*(1.-COVER_SN_F) ! m/s (AP mm/h --> m/s)
 END IF

! variation of water storage in the canopy
 IF (P_TEMP_F_STEP<=DTK) THEN ! if TF<0^C (frost) (could be introduced frost T)

! the eventual frozen dew is treated as snow
! the heat quantity involved is already included in LHF_F term through LAT_HEA_EVA
  IF (LEA_WAT_CON>0.) THEN
   PRECSN=PRECSN+LEA_WAT_CON/TIMESTEP_SEC

! Frost - once frost is formed, liquid water interception by canopy is set to ZERO
   LEA_WAT_CON=0. ! it means all dew is frozen
   LSNOW3=.true.
  END IF
 ELSE ! if T>=0C
! water on leaves = old + (rainfall - drained water - water evaporated from leaves)
  LEA_WAT_CON=MAX(0.,(P_LEA_WAT_CON+TIMESTEP_SEC*(PREC_F-VEG_DRAIN-EVA_F_W/DENS_WAT))) ! m
 ENDIF
!???note ! I use VEG_DRAIN which is always zero, so it is like VEG_DRAIN is useless ???

 IF (LEA_WAT_CON>LEA_WAT_CON_MAX) THEN
! drainage from leaves to soil: this can only happen for P_TEMP_F_STEP>0^C
  VEG_DRAIN=(LEA_WAT_CON-LEA_WAT_CON_MAX)/TIMESTEP_SEC ! drainage, in m/s
  LEA_WAT_CON=MIN(LEA_WAT_CON,LEA_WAT_CON_MAX)  ! leaf water content, in m, i.e.
  LEA_WAT_CON_MAX
 END IF

END IF

! if there is snowfall all precipitation is assumed to be snow (no drainage)
! it could be possible to introduce averaging, to take into account wet snow

IF (LSNOW2) PRECSN=PRECSN+(PREC_STEP/3.6E6) ! m/s (AP converted from mm/h to m/s)
SUM_PREC_SN=SUM_PREC_SN+PRECSN*TIMESTEP_SEC

! if there is rainfall, rain entering on the soil and snow are calculated
IF ((.NOT.LSNOW2).AND.PREC_STEP>0.) THEN
! m/s (AP converted from mm/h to m/s)
 PREC_G=((PREC_STEP/3.6E6)-PREC_F+VEG_DRAIN)*(1.-COVER_SN_G) !m/s prec reaching soil
 PREC_ON_SNOW=((PREC_STEP/3.6E6)-PREC_F+VEG_DRAIN)*COVER_SN ! m/s prec reaching snow
END IF
```

```
RETURN
END SUBROUTINE WATER_MASS

!******************************************************************************
SUBROUTINE WATER_MASS_GROUND(TIMESTEP_SEC,EVA_G,EVA_G_W,SRF_WAT_CON,&
 SRF_ICE_CON,SRF_WAT_CON_MAX,P_SRF_WAT_CON,P_SRF_ICE_CON,PG_EFF,Q_A_F,QSG_RH,P_ETA_I1,ETA_I1)
!******************************************************************************
! Calculates mass and water balance on the soil
!******************************************************************************
! Calls: none
! - Author: R. Bonanno (01-Sep-2010)
! - Last revision: C. Cassardo (15-May-2014)
!******************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL :: DELTA
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL,INTENT(IN) :: EVA_G
REAL,INTENT(IN) :: EVA_G_W
REAL,INTENT(OUT) :: SRF_WAT_CON
REAL,INTENT(INOUT) :: SRF_ICE_CON
REAL,INTENT(IN) :: SRF_WAT_CON_MAX
REAL,INTENT(IN) :: ETA_I1
REAL,INTENT(INOUT) :: P_SRF_WAT_CON
REAL,INTENT(INOUT) :: P_SRF_ICE_CON
REAL,INTENT(IN) :: P_ETA_I1
REAL,INTENT(INOUT) :: PG_EFF
REAL,INTENT(IN) :: Q_A_F
REAL,INTENT(IN) :: QSG_RH

! Soil surface water freezing occur when the first soil layer is frozen

IF(P_ETA_I1==0. .AND. ETA_I1>0.)THEN !Freezing
 P_SRF_ICE_CON=P_SRF_WAT_CON
 SRF_WAT_CON=0.
END IF

IF(P_ETA_I1>0. .AND. ETA_I1==0.)THEN !Melting
 P_SRF_WAT_CON=P_SRF_ICE_CON
 SRF_ICE_CON=0.
END IF

IF (ETA_I1>0.) THEN
 IF (QSG_RH>=Q_A_F) THEN ! sublimation (ice --> water vapor)
  SRF_ICE_CON = P_SRF_ICE_CON + TIMESTEP_SEC*(PG_EFF-EVA_G_W/DENS_WAT)
 ELSE ! sublimation (water vapor --> ice)
  SRF_ICE_CON = P_SRF_ICE_CON + TIMESTEP_SEC*(PG_EFF-EVA_G/DENS_WAT)
 END IF
ELSE
 IF (QSG_RH>=Q_A_F) THEN ! evaporation
  SRF_WAT_CON = P_SRF_WAT_CON + TIMESTEP_SEC*(PG_EFF-EVA_G_W/DENS_WAT)
 ELSE ! condensation
  SRF_WAT_CON = P_SRF_WAT_CON + TIMESTEP_SEC*(PG_EFF-EVA_G/DENS_WAT)
 END IF
ENDIF

SRF_WAT_CON=MAX(0.,SRF_WAT_CON)
SRF_ICE_CON=MAX(0.,SRF_ICE_CON)

! Setting to zero PG_EFF because the water stored in the soil surface can't
! infiltrate into the soil. Usually PG_EFF*TIMESTEP_SEC << SRF_WAT_CON_MAX but even if
! PG_EFF > SRF_WAT_CON_MAX, we are in the case SRF_WAT_CON (or SRF_ICE_CON) > SRF_WAT_CON_MAX
! (see intructions below): thus the excess of water (DELTA) that can't be stored in the soil
! surface and can infiltrate in the soil.
PG_EFF=0.
```

```
IF (SRF_WAT_CON>SRF_WAT_CON_MAX) THEN
 DELTA=(SRF_WAT_CON - SRF_WAT_CON_MAX)/TIMESTEP_SEC  ! drainage, in m/s
 PG_EFF=PG_EFF+DELTA
 SRF_WAT_CON=SRF_WAT_CON_MAX ! value set equal to its maximum
END IF

IF (SRF_ICE_CON>SRF_WAT_CON_MAX) THEN
 DELTA=(SRF_ICE_CON - SRF_WAT_CON_MAX)/TIMESTEP_SEC  ! drainage, in m/s
 PG_EFF=PG_EFF+DELTA
 SRF_ICE_CON=SRF_WAT_CON_MAX ! value set equal to its maximum
END IF

RETURN
END SUBROUTINE WATER_MASS_GROUND

!****************************************************************************
REAL FUNCTION WATERSURFLUX(RAD_S_D,RAD_S_U,RAD_L_D,RAD_L_U,LHF_G,SHF_A)
!****************************************************************************
! Calculates the flux into the water surface. Snow melt is not presently included.
! BTAU is the fraction of net solar radiation absorbed in the surface
! layer (input). WAT_SUR_FLU is the flux into the water surface, used as surface boundary
! for water surface (output)
!****************************************************************************
! Calls: none
! - Author: M. W. Qian (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: LHF_G
REAL,INTENT(IN) :: SHF_A
REAL,INTENT(IN) :: RAD_L_D
REAL,INTENT(IN) :: RAD_L_U
REAL,INTENT(IN) :: RAD_S_D
REAL,INTENT(IN) :: RAD_S_U

!WAT_SUR_FLU=BTAU*(RAD_S_D-RAD_S_U)-(RAD_L_D-RAD_L_U+LHF_G+SHF_A)
WATERSURFLUX=BTAU*(RAD_S_D-RAD_S_U)+(RAD_L_D-RAD_L_U)-(LHF_G+SHF_A)

RETURN
END FUNCTION WATERSURFLUX

!****************************************************************************
REAL FUNCTION WEIGHTMEAN(AAA1,AAA2,DDD1,DDD2)
!****************************************************************************
! Calculates the weigthed average of quantities AAA1 and AAA2
!****************************************************************************
! Calls: none
! - Author: C. Cassardo (16-Jun-1994)
! - Last revision: C. Cassardo (15-May-2014)
!****************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: AAA1 ! First value
REAL,INTENT(IN) :: AAA2 ! Second value
REAL,INTENT(IN) :: DDD1 ! Weight of A1
REAL,INTENT(IN) :: DDD2 ! Weight of A2

WEIGHTMEAN = ERR_VE
IF (AAA1==ERR_VE.OR.AAA2==ERR_VE.OR.DDD1==ERR_VE.OR.DDD2==ERR_VE) RETURN
WEIGHTMEAN = ( AAA1 * DDD1 + AAA2 * DDD2 ) / ( DDD1 + DDD2 )

RETURN
END FUNCTION WEIGHTMEAN
```

```
!***************************************************************************
REAL FUNCTION WETBULB_TEMP(RH_STEP,TEMP_A_STEP,PRES_A_STEP,AIR_DENS )
!***************************************************************************
! Computes the wet bulb temperature given the air temperature, pressure and specific
! humidity
!***************************************************************************
! Calls: DEWPOINT, SPECUM
! - Authors: M. Manfrin, N. Loglisci (28-Nov-2002)
! - Last revision: C. Cassardo (15-May-2014)
!***************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(IN) :: PRES_A_STEP
REAL,INTENT(IN) :: TEMP_A_STEP
REAL,EXTERNAL :: LAT_HEA_EVA
REAL,INTENT(IN) :: AIR_DENS
REAL,EXTERNAL :: DEWPOINT
REAL :: ES
REAL :: PIPPO
REAL,INTENT(IN) :: RH_STEP
REAL :: SPEC_UM
REAL :: SPEC_UMTEMP
REAL :: STEPTEMP
! amount of water progressively added to reach saturation [kg/m3]
REAL,PARAMETER :: STEPWATER=0.05E-3
REAL :: TDEWTEMP
REAL :: TTEMP

CALL SPECUM(RH_STEP,TEMP_A_STEP,PRES_A_STEP,SPEC_UM,PIPPO,ES)
SPEC_UMTEMP=SPEC_UM
TTEMP=TEMP_A_STEP

DO
! temperature decrease due to the evaporation of the water added
 STEPTEMP=-LAT_HEA_EVA(TTEMP)*STEPWATER/(CPAIR*AIR_DENS)
 TTEMP=TTEMP+STEPTEMP
 SPEC_UMTEMP=SPEC_UMTEMP+STEPWATER

! water vapor pressure at new temperature (ttemp) with new spec hum spec_umtemp
 TDEWTEMP=DEWPOINT(SPEC_UMTEMP,PRES_A_STEP)
 IF(TDEWTEMP>=TTEMP) EXIT
END DO

WETBULB_TEMP=(TDEWTEMP+TTEMP)/2.
! interpolation in the middle between TDEWTEMP and TTEMP - not correct but we do not
! know exactly where is it. The precision depends on number of iterations and steps

RETURN
END FUNCTION WETBULB_TEMP
!***************************************************************************
SUBROUTINE MANAGEVEG(CUMTOT,DTK,TIMESTEP_SEC,HEI_SN,HVEG,HVEG0,LAI,LAI0,VEG_COV_STEP,&
 VEG_COV_STEP0,TAIR,TCAN,TROOT,VEGSHR)
!***************************************************************************
! Routine that modifies the properties of some kinds of seasonal vegetation (grass,
! shrubs, ...) according with weather type. The reason is that, in case of very
! cold weather or snow cover, some kind of vegetation shrinks, reducing its cover
! and LAI and height. When dormancy ends, vegetation restart to develop, gradually
! reaching its nominal coverage. The code VEGSHR allows this kind of calculation.
! Coefficient are empirical and no tests with experimental data on these routines
! have been done.
!***************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Mar-2016)
! - Last revision: C. Cassardo (22-Mar-2016)
!***************************************************************************
```

```
IMPLICIT NONE

REAL,INTENT(INOUT) :: CUMTOT
REAL,PARAMETER :: CUMCMAX=0.
REAL,PARAMETER :: CUMMIN=-15.
REAL,PARAMETER :: CUMMAX=25.
REAL,PARAMETER :: CUMWMIN=15.
REAL :: DHVEG
REAL :: DLAI
REAL :: DVEG_COV
REAL,INTENT(IN) :: DTK
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL :: FACTOR
REAL,INTENT(IN) :: HEI_SN
REAL,INTENT(INOUT) :: HVEG
REAL,INTENT(IN) :: HVEG0
REAL,INTENT(INOUT) :: LAI
REAL,INTENT(IN) :: LAI0
REAL,INTENT(INOUT) :: VEG_COV_STEP
REAL,INTENT(IN) :: VEG_COV_STEP0
REAL,INTENT(IN) :: TAIR
REAL,INTENT(IN) :: TCAN
REAL,INTENT(IN) :: TROOT
LOGICAL,INTENT(IN) :: VEGSHR

FACTOR=1.

! calculation performed only for selected vegetation type (in which VEGSHR is TRUE)
IF (VEGSHR) THEN

! call routines for evaluating indices of coldness CUMCOLD and warmness CUMWARM
! cold index CUMCOLD is limited to CUMCOLDMAX
 CALL WARMCOLDVEG(TIMESTEP_SEC,HEI_SN,TAIR,TCAN,TROOT,CUMTOT)
 CUMTOT=MIN(CUMMAX,MAX(CUMMIN,CUMTOT))

 IF (CUMTOT<=CUMCMAX) THEN
  FACTOR=(1./1.001-1.)
  DHVEG   = FACTOR*0.9*HVEG0 ! hypothesis of variation between 0.1*HVEG0 and HVEG0
  DVEG_COV = FACTOR*0.9*VEG_COV_STEP0
  DLAI    = FACTOR*0.9*LAI0
 ELSE IF (CUMTOT>CUMWMIN) THEN
  FACTOR=(1.001-1.)
  DHVEG   = FACTOR*0.9*HVEG0 ! hypothesis of variation between 0.1*HVEG0 and HVEG0
  DVEG_COV = FACTOR*0.9*VEG_COV_STEP0
  DLAI    = FACTOR*0.9*LAI0
 ELSE
  DHVEG   = 0.
  DVEG_COV = 0.
  DLAI    = 0.
 ENDIF

 HVEG=MIN(HVEG0,MAX(HVEG0*0.1,HVEG+DHVEG))
 VEG_COV_STEP=MIN(VEG_COV_STEP0,MAX(VEG_COV_STEP0*0.1,VEG_COV_STEP+DVEG_COV))
 LAI=MIN(LAI0,MAX(LAI0*0.1,LAI+DLAI))

ELSE
 HVEG=HVEG0
 VEG_COV_STEP=VEG_COV_STEP0
 LAI=LAI0
ENDIF

RETURN
END
!******************************************************************************
SUBROUTINE WARMCOLDVEG(TIMESTEP_SEC,HEI_SN,TAIR,TCAN,TROOT,CUMTOT)
!******************************************************************************
! Routine that evaluates the index of heat CUMTOT based on air, canopy,
```

```
! soil root zone temperatures and snow cover. All values are weighted by the
! number of seconds of the timestep, adimensionalized with one day. Also all
! values are weighted for the respective thresholds, in order that, if one
! threshold is systematically exceeded for an entire day, CUMTOT is incremented
! by one unity. In case thresholds are exceeded by more than one variable, the
! maximum factor is considered.
! Coefficient by now are empirical and no tests with experimental data on this
! routine have been done yet.
!*************************************************************************
! Calls: none
! - Author: C. Cassardo (22-Mar-2016)
! - Last revision: C. Cassardo (22-Mar-2016)
!*************************************************************************
USE CONSTANTS
IMPLICIT NONE

REAL,INTENT(INOUT) :: CUMTOT
REAL,INTENT(IN) :: TIMESTEP_SEC
REAL, PARAMETER :: FACTOR=48.   ! experimental factor
REAL,INTENT(IN) :: HEI_SN
REAL,INTENT(IN) :: TAIR
REAL,INTENT(IN) :: TCAN
REAL :: TMPC,TMPC1,TMPC2,TMPC3,TMPC4,TMPW,TMPW1,TMPW2,TMPW3
REAL,INTENT(IN) :: TROOT
REAL, PARAMETER :: H_THRES=0.05 ![m]
!REAL, PARAMETER :: T_THRESNEG=-10.+DTK ![°C]
!REAL, PARAMETER :: T_THRESZER=0.+DTK  ![°C]
!REAL, PARAMETER :: T_THRESPOS=10.+DTK ![°C]
REAL, PARAMETER :: T_THRESNEG=-5.+DTK ![°C]
REAL, PARAMETER :: T_THRESZER=5.+DTK  ![°C]
REAL, PARAMETER :: T_THRESPOS=15.+DTK ![°C]

!IF (TAIR>DTK .AND. TCAN>DTK .AND. HEI_SN<=5.) RETURN ! nothing happens

TMPC=0. ! initialize counters for cold effects
TMPC1=0.
TMPC2=0.
TMPC3=0.
TMPC4=0.

TMPW=0. ! initialize counters for warm effects
TMPW1=0.
TMPW2=0.
TMPW3=0.

! effect of air temperature
! it is assumed that, in case of T<0°C, negative effect will accumulate
! for T<-T_THRES effects are not changing
IF (TAIR<=T_THRESZER) THEN
 TMPC1=(DTK-MAX(T_THRESNEG,TAIR))*FACTOR*TIMESTEP_SEC/86400./(T_THRESZER-T_THRESNEG)
ENDIF

! effect of canopy temperature, assumed similar to the one for Tair
IF (TCAN<=T_THRESZER) THEN
 TMPC2=(DTK-MAX(T_THRESNEG,TCAN))*FACTOR*TIMESTEP_SEC/86400./(T_THRESZER-T_THRESNEG)
ENDIF

! effect of soil root temperature, assumed similar to the one for Tair
IF (TROOT<=T_THRESZER) THEN
 TMPC3=(DTK-MAX(T_THRESNEG,TROOT))*FACTOR*TIMESTEP_SEC/86400./(T_THRESZER-T_THRESNEG)
ENDIF

! effect of snow
! it is assumed that snow cover larger than 5 cm can progressively damage
! the vegetation; for HEI_SN>H_THRES effects are not changing
IF (HEI_SN>0.) THEN
 TMPC4=-MIN(0.05,HEI_SN)*FACTOR*TIMESTEP_SEC/86400./H_THRES
```

```
ENDIF

! effect of air temperature
! it is assumed that, in case of T<0°C, negative effect will accumulate
! for T<-T_THRES effects are not changing
IF (TAIR>T_THRESZER) THEN
 TMPW1=(MIN(T_THRESPOS,TAIR)-DTK)*FACTOR*TIMESTEP_SEC/86400./(T_THRESPOS-T_THRESZER)
ENDIF

! effect of canopy temperature, assumed similar to the one for Tair
IF (TCAN>T_THRESZER) THEN
 TMPW2=(MIN(T_THRESPOS,TCAN)-DTK)*FACTOR*TIMESTEP_SEC/86400./(T_THRESPOS-T_THRESZER)
ENDIF

! effect of soil root temperature, assumed similar to the one for Tair
IF (TROOT>T_THRESZER) THEN
 TMPW3=(MIN(T_THRESPOS,TROOT)-DTK)*FACTOR*TIMESTEP_SEC/86400./(T_THRESPOS-T_THRESZER)
ENDIF

TMPC=MIN(TMPC1,TMPC2,TMPC3,TMPC4)
TMPW=MAX(TMPW1,TMPW2,TMPW3)

CUMTOT=CUMTOT+TMPW+TMPC

RETURN
END

!=============================================================================
! external routines
INCLUDE '../model/eco_routines.f90'
INCLUDE '../model/calendario2000.f90'
```