**MetGrid_Handler module**

**Accessing MetGrid files from Fortran**

A Fortran module contains data structure definitions that specify defined type variables that allow the grouping of various types of data into structured data units, and sets of functions or subroutines that use these units or operate on them. The module is invoked from within a Fortran program with the `use` command thus: use MetGrid_Handler

**Module Description**

**Variable structures**

The structure most likely to be interpreted by the user, in that the internal variables will be used in a Fortran program is the climate_structure. Most of the others are specific to the internal workings of the module and so are only minimally described. This structure normally only exists during a run unit of the program.

```fortran
module MetGrid_Handler

type climate_structure
  real*4  :: lat, & ! decimal degrees latitude
             long, & ! decimal degrees longitude
             phase, & ! radians rotation angle
             confidence ! unused when constructed from MetGrid record
  real*4  :: rain(12)             ! mm per day
  real*4  :: temp(12)           ! mean temperature degrees centigrade
  real*4  :: diurn(12) ! diurnal temperature range degrees centigrade
  real*4  :: srad(12) ! solar radiation Mj m\(^{-2}\) day\(^{-1}\)
  real*4  :: raindays(12)    ! dimensionless 0 to 1
  integer*4 :: elev  ! metres above sea level
  logical*1 :: rotated  ! climate structure may be rotated or not
end type climate_structure

module routines to access MetGrid records use the loaded index arrays.
```

**Public variables**

Some variables are used within the module, but are defined as public as they may be of use in a program using the module. These include halfpi, pi and twopi as real*4, eof, year, month and day as integer*4 and month_code as an array of 12 three byte character representations of the calendar month. If the MetGrid header has been read then it is held in the variable h. Lastly, two logical*4 variables header_loaded and index_loaded indicate if the respective loading has been done.

The index arrays are defined as private and are only accessed within the module; module routines to access MetGrid records use the loaded index arrays.
**Function descriptions**

As with the variable structures, not all module functions are necessarily called by the end user. Some have to be included because they are called by end-user functions; others are there because they are of use in the administration of the files.

**Grid functions**

G01  integer*4 function grid_col (long)
G02  integer*4 function grid_row (lat, error)
G03  real*4 function grid_lat (row)
G04  real*4 function grid_long (col)

**Calendar functions**

C01  pure real*4 function month_days (month, year)
C02  pure integer*4 function rotate_month (m, phase)
C03  pure logical function leap (year)

**Input functions**

I01  type (metgrid_record) function find_met_record (lat, long, unit, error)
I02  type (metgrid_record) function nearest_met_record (lat, long, max_distance, unit)
I03  type (cli_record) function read_cli (filename)
I04  type (metgrid_hdr) function read_metgrid_header (path, version)
I05  subroutine load_metgrid_index (path, version)
I06  subroutine open_metgrid_files (path, version, unit)
I07  subroutine close_metgrid_files (unit)

**Output routines**

O01  subroutine print_climate(c)
O02  subroutine print_metgrid_record (m)
O03  subroutine write_climate(c, unit)
O04  subroutine write_cli (cli, version, path)
O05  subroutine write_pane_descriptor (p, path)

**Structure Operation functions**

B01  type (climate_structure) function make_climate_structure (m) result(c)
B02  type (climate_structure) function null_climate () result(c)
B03  type (cli_record) function make_CLI_from_climate_structure(c) result (cli)
B04  type (climate_structure) function rotate_climate (ctc, phase) result(c)
B05  type (metgrid_record) function compress_met_record(c) result (m)
B06  type (metgrid_record) function null_metgrid_record () result (m)

**Climate Rotation functions**

R01  type (climate_structure) function new_rotate(c) result(r)
R02  subroutine neg_correct (v)
R03  subroutine rotate (a, thru)
R04  subroutine decode (q, v)
R05  subroutine encode (v, q)
R06  subroutine freq1 (q,f)
R07 subroutine frqinv1(f,q)
R08 real*4 function angle(sin,cos)result(a)
R09 subroutine frrota(q,thru)

MetGrid Packing and Index tools

M01 subroutine pack12(v, n, p)
M02 subroutine unpack12 (p, n, v)
M03 type (pane_descriptor) function pane (row, col, version) result (p)
M04 integer*4 function version_index (ver) result (ind)

Examples

! PURPOSE: recover the climate data for Dolgellau,
! and print the august rainfall
!******************************************************************************

program example_1
use metgrid_handler
implicit none

type (metgrid_record) :: m

type (climate_structure) :: c

logical*4 :: error

call open_metgrid_files ('c:\worldclim_2\metgrids\',120,1)
m = find_met_record(52.737,-3.883,1,error)
if(error) then
  print *, 'coordinates must be wrong'
  stop
end if
c = make_climate_structure (m)
c = rotate_climate(c)
print *, 'Dolgellau August rainfall', c.rain(8)*month_days(8)
stop
end program example_1

------------------------------------------------------------------------------

! PURPOSE: finding the nearest MetGrid record
! when coordinates may be uncertain
!******************************************************************************

program example_2
use metgrid_handler
implicit none

type (metgrid_record) :: m

type (climate_structure) :: c

call open_metgrid_files('c:\worldclim_2\metgrids\', 120, 1)
m = nearest_met_record(52.72, -4.087, 10.0, 1)
c = make_climate_structure(m)
c = rotate_climate(c)
call print_climate(c)
! PURPOSE: To map the rainfall/temperature index for North Wales
!(this is not a recognised or useful index, but serves to illustrate a point)
!******************************************************************************

program Example_3
use MetGrid_Handler
use image_processing ! Note this module is available from P.G.Jones but
                   ! is not yet fully documented
implicit none
real*,parameter    ::  NE(2)=[53.5,-5.0],SW(2)=[52.3,-3.0] ! bounds

               type (idrisi_doc)   ::  d
               type (metgrid_record) ::  m
               type (climate_structure) ::  c

real*,allocatable ::  im(:,;)
integer*4         ::  rows(2),cols(2),row,col
logical*4         ::  error

call open_metgrid_files('c:\worldclim_2\metgrids\', 120, 1)
rows(1) = grid_row(NE(1))
rows(2) = grid_row(SW(1))
.cols(1) = grid_col(NE(2))
cols(2) = grid_col(SW(2))

allocate (im(cols(1):cols(2),rows(1):rows(2)))
im = -9999.0

      do row = rows(1),rows(2)
        do col = cols(1),cols(2)
          m = find_met_record(grid_lat(row),grid_long(col),1,error)
          if(error) cycle
          c = make_climate_structure(m)
          im(col,row) = sum(c.rain/c.temp)
        end do
      end do

      d.min_value = minval(im,im.ne.-9999.0)
      d.max_value = maxval(im)
d.display_min  = d.min_value
d.display_max  = d.max_value

open (10, file='c:\worldclim_2\data\N_Wales_example.rst', form='binary')
write (10) im
close (10)
open (10, file='c:\worldclim_2\data\N_Wales_example.rdc')
call write_idrisi_rdc(d, 10)
stop

end program Example_3