PaHM

Generated by Doxygen 1.9.2

1 PaHM

Version: 1.0.0

Over the years, various parametric wind models have been developed to estimate the surface winds within a tropical cyclone given the track of the storm. Such models can be very useful on forcing ocean and wave models in storm surge simulations, as they are lightweight and they do not require much time or computational resources to produce the wind fields on the fly for the duration of the storm. The Parametric Modeling System (*PaHM* `https://github.com/noaa-ocs-modeling/PaHM`) is developed to be used as a general atmospheric modeling system for coastal applications.

PaHM is not only an atmospheric model but rather a modeling system that contains multiple parametric models (i.e., Rankine Vortex Model, Holland Models (1980, 2010), Willoughby Model and the Generalized Asymmetric Vortex Model GAHM), and it is left to the user to activate any of these models to generate the wind fields at run time. In the case of the presence of multiple storms in the basin, *PaHM* has the capability to integrate all the storms when generating the wind fields. *PaHM* can be used either as a standalone atmospheric model, or can be coupled with ocean and wave models via NOAA's Environmental Modeling System (NEMS), a common modeling coupling framework that implements the National Unified Operational Prediction Capability (NUOPC).

1.1 Table of Contents

1.1.1 Modeling System Description

A presentation of the Parametric Hurricane Modeling System with implemented features and roadmap.

1.1.2 Installation guide

Installation and developement of *PaHM* is done through the distributed version control system Git. Even if a tarball could be sufficient, we advise to use Git system to follow *PaHM* development and merge easily to new versions. Building *PaHM* from sources requires to compile third party libraries and the use of CMake. These points are detailed below.

- System prerequisites
- Build Third party libraries
- Get PaHM on GitHub repository
- Build PaHM
- Run PaHM (different ways to execute PaHM).

1.1.3 User guide

- Project's files organization to get started with *PaHM*.
- User interface documentation to define and run *PaHM*.

1.1.4 Best pratices

Some advices around *PaHM*.

1.1.5 Developer's corner

This part of the documentation is intended for advanced developers, where he or she will find useful information on each **`Module and topic`** as well as precise descriptions and comments on subroutines, functions, variables, and types.

The detailed descriptions of Verification and validation test cases are also detailed with configurations and expected numerical results.

1.1.6 PaHM code

The complete Fortran source code and its associated documentation is visible in this section.

2 PaHM code

This part of the documentation proposes the complete Fortran code source and documentation with precise classification.

2.1 Class list

The `exhaustive list of variables, fields, types, enum and modules`.

2.2 File list

The whole `project tree with source files`.

2.3 Todo list

The `list of work` that still need to be done (and thus limitations), feel free to join the effort!

2.4 Deprecated list

#The `list of code` that is deprecated from previous version, and that you should be aware of.

3 Modeling System Description

3.1 Introduction

3.2 Purposes

3.3 Technical features list

3.3.1 Modeling features

3.3.2 Numerical features

3.3.3 System and validation features

3.3.4 Third-party libraries

3.3.5 Compilation tools

4 Modules Index

4.1 Modules List

Here is a list of all modules with brief descriptions:

5 Data Type Index

5.1 Data Types List

Here are the data types with brief descriptions:

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

7 Module Documentation

7.1 csv_module Module Reference

Data Types

- type csv_file
- type csv_string

Functions/Subroutines

- subroutine initialize_csv_file (me, quote, delimiter, enclose_strings_in_quotes, enclose_all_in_quotes, logical_↩ true_string, logical_false_string, chunk_size)

  Initialize a [[csv_file(type)]].

Variables

- integer, parameter, public csv_type_string = 1
- integer, parameter, public a
- integer, parameter, public character
- integer, parameter, public string
- integer, parameter, public cell
- integer, parameter, public csv_type_double = 2
- integer, dimension(wp), parameter, public real
- integer, parameter, public csv_type_integer = 3
- integer, parameter, public an
- integer, dimension(ip), parameter, public integer
- integer, parameter, public csv_type_logical = 4
- integer, parameter, public logical

7.1.1 Function/Subroutine Documentation

7.1.1.1 initialize_csv_file()

```
subroutine csv_module::initialize_csv_file (
    class(csv_file), intent(out) me,
    character(len=1), intent(in), optional quote,
    character(len=1), intent(in), optional delimiter,
    logical, intent(in), optional enclose_strings_in_quotes,
    logical, intent(in), optional enclose_all_in_quotes,
    character(len=1), intent(in), optional logical_true_string,
    character(len=1), intent(in), optional logical_false_string,
    integer, intent(in), optional chunk_size )
```

Initialize a [[csv_file(type)]].

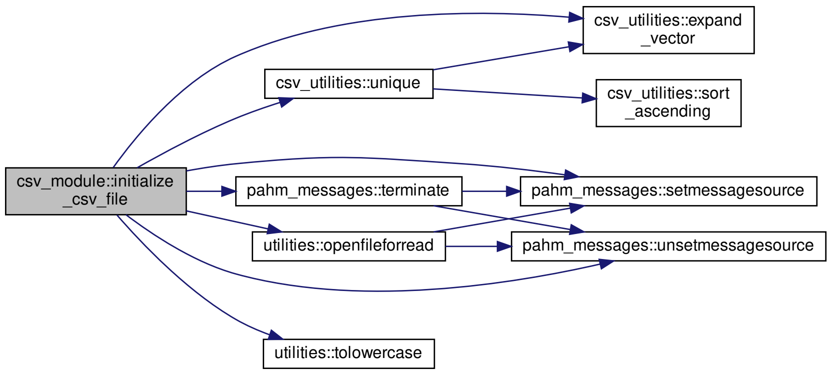
Parameters

me	The ouput csv_file structure
quote	Can only be one character (optional, default is `"`)
delimiter	Can only be one character (optional, default is `,`)
enclose_strings_in_quotes	Logical flag; if true, all string cells will be enclosed in quotes (optional, default is `T`)
enclose_all_in_quotes	Logical flag; if true, *all* cells will be enclosed in quotes (optional, default is `F`)
logical_true_string	Logical flag; when writing a logical `true` value to a CSV file, this is the string to use (optional, default is `T`)
logical_false_string	Logical flag; when writing a logical `false` value to a CSV file, this is the string to use (optional, default is `T`)
chunk_size	Factor for expanding vectors (default is 100)

Definition at line 166 of file csv_module.F90.

References csv_type_double, csv_type_integer, csv_type_logical, csv_type_string, csv_parameters::default_int_fmt, csv_parameters::default_real_fmt, pahm_messages::error, csv_utilities::expand_vector(), pahm_messages::info, logical, pahm_global::lun_btrk, utilities::openfileforread(), real, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_messages::terminate(), utilities::tolowercase(), csv_utilities::unique(), pahm_messages::unsetmessagesource(), and pahm_sizes::wp.

Here is the call graph for this function:



7.1.2 Variable Documentation

7.1.2.1 a `integer parameter public csv_module::a`

Definition at line 30 of file csv_module.F90.

7.1.2.2 an `integer, parameter, public csv_module::an`

Definition at line 32 of file csv_module.F90.

7.1.2.3 cell `integer parameter public csv_module::cell`

Definition at line 30 of file csv_module.F90.

7.1.2.4 character `integer, parameter, public csv_module::character`

Definition at line 30 of file csv_module.F90.

7.1.2.5 csv_type_double `integer, parameter, public csv_module::csv_type_double = 2`

Definition at line 31 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.6 csv_type_integer `integer, parameter, public csv_module::csv_type_integer = 3`

Definition at line 32 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.7 csv_type_logical `integer, parameter, public csv_module::csv_type_logical = 4`

Definition at line 33 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.8 csv_type_string `integer, parameter, public csv_module::csv_type_string = 1`

Definition at line 30 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.9 integer `integer, dimension(ip), parameter, public csv_module::integer`

Definition at line 32 of file csv_module.F90.

7.1.2.10 logical `integer, parameter, public csv_module::logical`

Definition at line 33 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.11 real `parwind::hollanddata_t::real`

Definition at line 31 of file csv_module.F90.

Referenced by initialize_csv_file().

7.1.2.12 string `integer, parameter, public csv_module::string`

Definition at line 30 of file csv_module.F90.

7.2 csv_parameters Module Reference

Variables

- integer(ip), parameter, public max_real_str_len = 27
- integer(ip), parameter, public maximum
- integer(ip), parameter, public string
- integer(ip), parameter, public length
- integer(ip), parameter, public of
- integer(ip), parameter, public a
- integer(ip), parameter, public real
- integer(ip), parameter, public number
- character(len= ∗), parameter, public default_real_fmt = '(E27.17E4)'
- integer(ip), parameter, public max_integer_str_len = 256

  default real number format statement (for writing real values to strings and files).
- integer(ip), parameter, public an
- integer(ip), parameter, public integer
- character(len= ∗), parameter, public default_int_fmt = '(I256)'

7.2.1 Variable Documentation

7.2.1.1 a `integer(ip), parameter, public csv_parameters::a`

Definition at line 22 of file csv_parameters.F90.

7.2.1.2 an `integer(ip), parameter, public csv_parameters::an`

Definition at line 26 of file csv_parameters.F90.

7.2.1.3 default_int_fmt `character(len=*), parameter, public csv_parameters::default_int_fmt = '(I256)'`

Definition at line 27 of file csv_parameters.F90.

Referenced by csv_module::initialize_csv_file().

7.2.1.4 default_real_fmt `character(len=*), parameter, public csv_parameters::default_real_fmt = '(E27.17E4)'`

Definition at line 23 of file csv_parameters.F90.

Referenced by csv_module::initialize_csv_file().

7.2.1.5 integer `csv_parameters::integer`

Definition at line 26 of file csv_parameters.F90.

7.2.1.6 length `integer(ip), parameter, public csv_parameters::length`

Definition at line 22 of file csv_parameters.F90.

7.2.1.7 max_integer_str_len `integer(ip), parameter, public csv_parameters::max_integer_str_len = 256`

default real number format statement (for writing real values to strings and files).

Definition at line 26 of file csv_parameters.F90.

7.2.1.8 max_real_str_len `integer(ip), parameter, public csv_parameters::max_real_str_len = 27`

Definition at line 22 of file csv_parameters.F90.

7.2.1.9 maximum `integer(ip), parameter, public csv_parameters::maximum`

Definition at line 22 of file csv_parameters.F90.

7.2.1.10 number `integer(ip), parameter, public csv_parameters::number`

Definition at line 22 of file csv_parameters.F90.

7.2.1.11 of `integer(ip), parameter, public csv_parameters::of`

Definition at line 22 of file csv_parameters.F90.

7.2.1.12 real `integer(ip), parameter, public csv_parameters::real`

Definition at line 22 of file csv_parameters.F90.

7.2.1.13 string `integer(ip), parameter, public csv_parameters::string`

Definition at line 22 of file csv_parameters.F90.

7.3 csv_utilities Module Reference

Functions/Subroutines

- pure subroutine, public expand_vector (vec, n, chunk_size, val, finished)

  Add elements to the integer vector in chunks.
- integer function, dimension(:), allocatable, public unique (vec, chunk_size)

  Returns only the unique elements of the vector.
- subroutine, public sort_ascending (ivec)

  Sorts an integer array `ivec` in increasing order. Uses a basic recursive quicksort (with insertion sort for partitions with $\leq$ 20 elements).

7.3.1 Function/Subroutine Documentation

7.3.1.1 expand_vector()

```
pure subroutine, public csv_utilities::expand_vector (
        integer, dimension(:), intent(inout), allocatable vec,
        integer, intent(inout) n,
        integer, intent(in) chunk_size,
        integer, intent(in), optional val,
        logical, intent(in), optional finished )
```

Add elements to the integer vector in chunks.

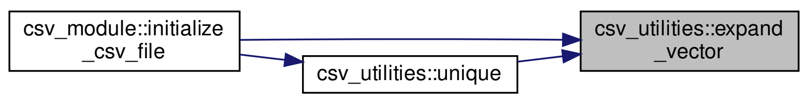
Parameters

vec	The input integer vector (input/output)
n	Counter for last element added to `vec`; must be initialized to `size(vec)` (or 0 if not allocated) before first call (input/output)
chunk_size	Allocate `vec` in blocks of this size (>0)
val	The value to add to `vec` (optional)
finished	Set to true to return `vec` as its correct size (`n`) (optional)

Definition at line 54 of file csv_utilities.F90.

Referenced by csv_module::initialize_csv_file(), and unique().

Here is the caller graph for this function:



7.3.1.2 sort_ascending() `subroutine, public csv_utilities::sort_ascending ( integer, dimension(:), intent(inout) ivec )`

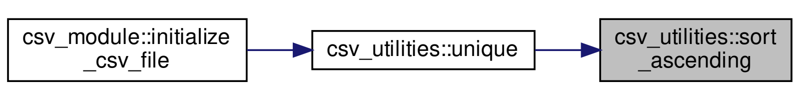
Sorts an integer array `ivec` in increasing order. Uses a basic recursive quicksort (with insertion sort for partitions with $\leq$ 20 elements).

Definition at line 142 of file csv_utilities.F90.

References pahm_sizes::ip.

Referenced by unique().

Here is the caller graph for this function:



7.3.1.3 unique() `integer function, dimension(:), allocatable, public csv_utilities::unique ( integer, dimension(size(vec)), intent(in) vec, integer, intent(in) chunk_size )`

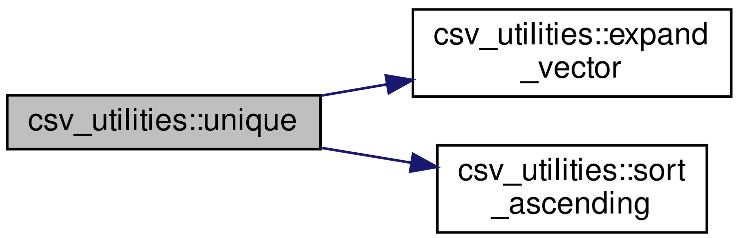
Returns only the unique elements of the vector.

Definition at line 103 of file csv_utilities.F90.

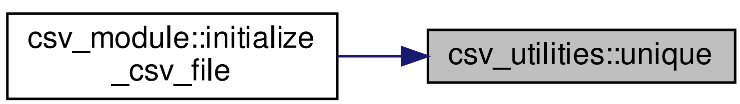
References expand_vector(), and sort_ascending().

Referenced by csv_module::initialize_csv_file().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4 pahm_drivermod Module Reference

Functions/Subroutines

- subroutine getprogramcmdlargs ()

  Prints on the screen the help system of the PaHM program.
- subroutine pahm_init ()

  Subroutine to initialize a PaHM run.
- subroutine pahm_run (nTimeSTP)

  Subroutine to run PaHM (timestepping).
- subroutine pahm_finalize ()

  Subroutine to finalize a PaHM run.

Variables

- integer, save cnttimebegin
- integer, save cnttimeend

7.4.1 Function/Subroutine Documentation

7.4.1.1 getprogramcmdlargs() `subroutine pahm_drivermod::getprogramcmdlargs`

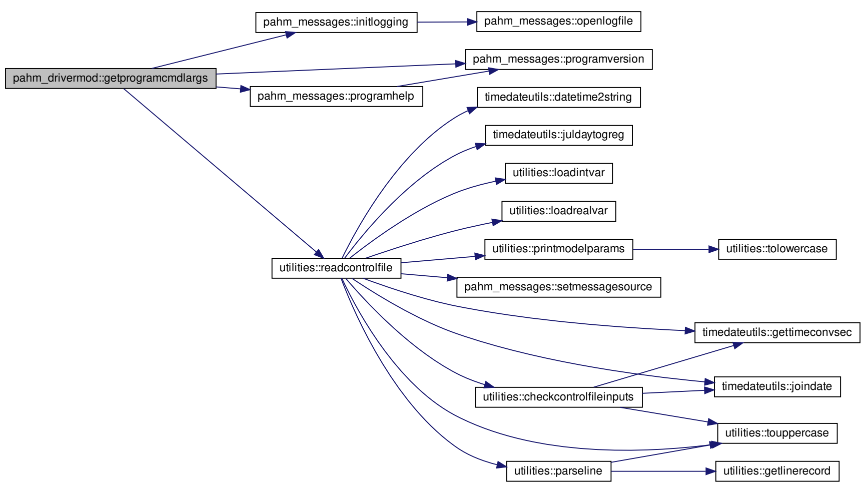
Prints on the screen the help system of the PaHM program.

Definition at line 40 of file driver_mod.F90.

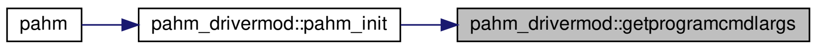
References pahm_global::controlfilename, pahm_messages::initlogging(), pahm_messages::programhelp(), pahm_messages::programve
and utilities::readcontrolfile().

Referenced by pahm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.1.2 pahm_finalize() `subroutine pahm_drivermod::pahm_finalize`

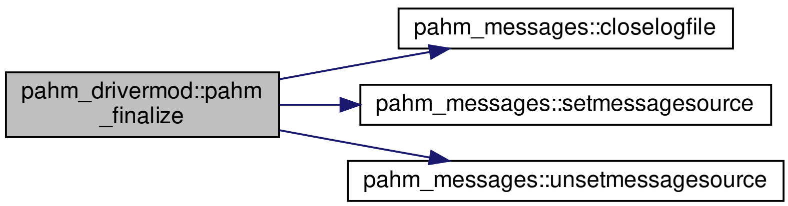
Subroutine to finalize a PaHM run.

Definition at line 194 of file driver_mod.F90.

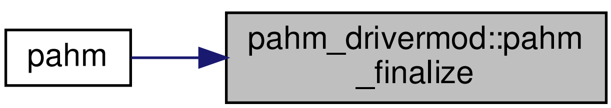
References pahm_messages::closelogfile(), pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by pahm().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.1.3 pahm_init() `subroutine pahm_drivermod::pahm_init`

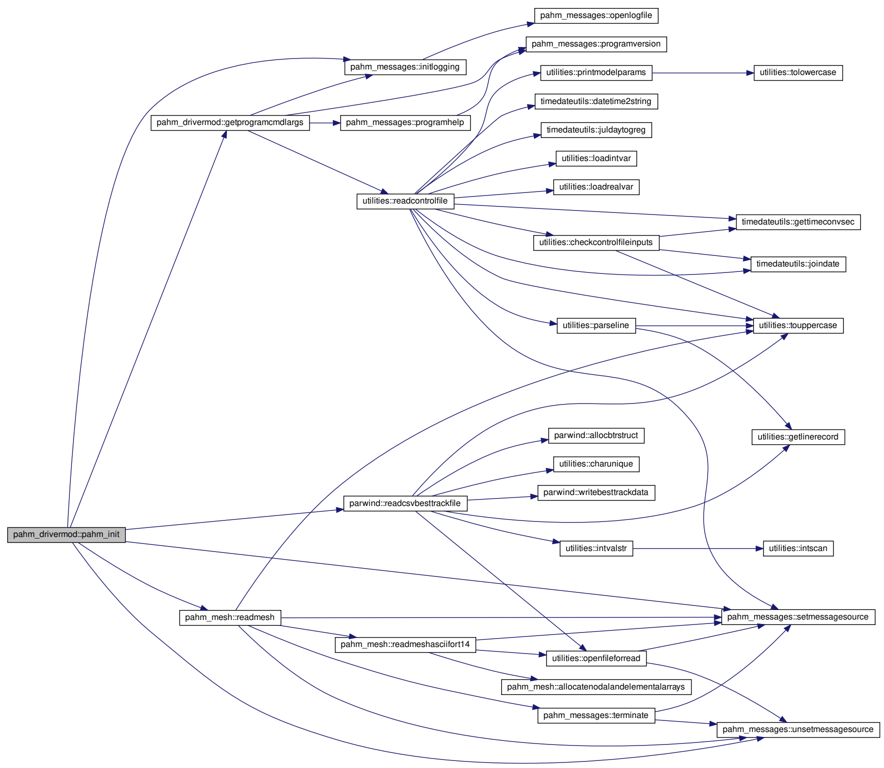
Subroutine to initialize a PaHM run.

Definition at line 94 of file driver_mod.F90.

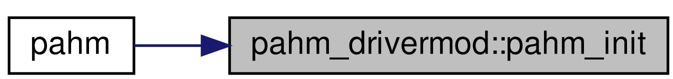
References cnttimebegin, cnttimeend, getprogramcmdlargs(), pahm_messages::initlogging(), pahm_global::noutdt, parwind::readcsvbesttrackfile(), pahm_mesh::readmesh(), pahm_messages::setmessagesource(), and pahm_messages::unsetmessages

Referenced by pahm().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.1.4 pahm_run()

```
subroutine pahm_drivermod::pahm_run (
    integer, intent(in), optional nTimeSTP )
```

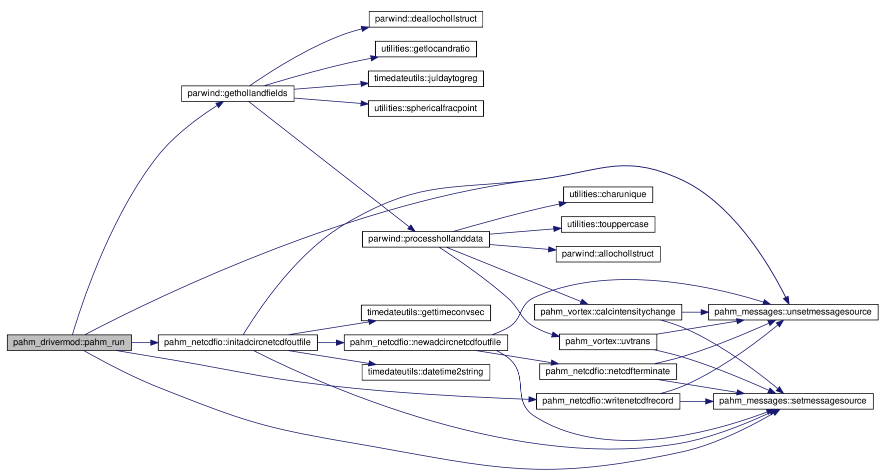
Subroutine to run PaHM (timestepping).

Definition at line 136 of file driver_mod.F90.

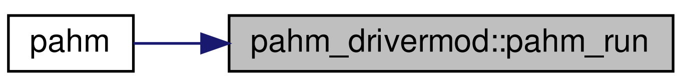
References cnttimebegin, cnttimeend, pahm_messages::error, parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), pahm_global::modeltype, pahm_global::outfilename, pahm_global::outfilenamespecified, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_global::times, pahm_messages::unsetmessagesource(), pahm_global::wpress, pahm_netcdfio::writenetcdfrecord(), pahm_global::wvelx, and pahm_global::wvely.

Referenced by pahm().

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.2 Variable Documentation

7.4.2.1 cnttimebegin `integer, save pahm_drivermod::cnttimebegin`

Definition at line 23 of file driver_mod.F90.

Referenced by pahm_init(), and pahm_run().

7.4.2.2 cnttimeend `integer, save pahm_drivermod::cnttimeend`

Definition at line 23 of file driver_mod.F90.

Referenced by pahm_init(), and pahm_run().

7.5 pahm_global Module Reference

Functions/Subroutines

- real(sz) function airdensity (atmT, atmP, relHum)
  This function calculates the density of the moist air.

Variables

- integer, parameter lun_screen = 6
- integer, parameter lun_ctrl = 10
- integer, parameter lun_inp = 14
- integer, parameter lun_inp1 = 15
- integer, parameter lun_log = 35
- integer, parameter lun_btrk = 22
- integer, parameter lun_btrk1 = 23
- integer, parameter lun_out = 25
- integer, parameter lun_out1 = 26
- real(sz), parameter defv_gravity = 9.80665_SZ
- real(sz), parameter defv_atmpress = 1013.25_SZ
- real(sz), parameter defv_rhoair = 1.1478_SZ
- real(sz), parameter defv_rhowater = 1000.0000
- real(sz), parameter one2ten = 0.8928_SZ
- real(sz), parameter ten2one = 1.0_SZ / 0.8928_SZ
- real(sz), parameter pi = 3.141592653589793_SZ
- real(sz), parameter deg2rad = PI / 180.0_SZ
- real(sz), parameter rad2deg = 180.0_SZ / PI
- real(sz), parameter basee = 2.718281828459045_SZ
- real(sz), parameter rearth = 6378206.4_SZ
- real(sz), parameter nm2m = 1852.0_SZ
- real(sz), parameter m2nm = 1.0_SZ / NM2M
- real(sz), parameter kt2ms = NM2M / 3600.0_SZ
- real(sz), parameter ms2kt = 1.0_SZ / KT2MS
- real(sz), parameter omega = 2.0_SZ ∗ PI / 86164.2_SZ
- real(sz), parameter mb2pa = 100.0_SZ
- real(sz), parameter mb2kpa = 0.1_SZ
- character(len=fnamelen) logfilename = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) controlfilename = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_control.in'
- logical meshfilenamespecified = .FALSE.
- character(len=fnamelen) meshfilename = BLANK
- character(len=64) meshfiletype = BLANK
- character(len=64) meshfileform = BLANK
- logical besttrackfilenamespecified = .FALSE.

- integer nbtrfiles = IMISSV
- character(len=fnamelen), dimension(:), allocatable besttrackfilename
- character(len=512) title = BLANK
- real(sz) gravity = DEFV_GRAVITY
- real(sz) rhowater = DEFV_RHOWATER
- real(sz) rhoair = DEFV_RHOAIR
- real(sz) backgroundatmpress = DEFV_ATMPRESS
- real(sz), parameter defv_bladjustfac = 0.9_SZ
- real(sz) windreduction = DEFV_BLADJUSTFAC
- real(sz) bladjustfac = DEFV_BLADJUSTFAC
- character(len=64) refdatetime = BLANK
- integer refdate = IMISSV
- integer reftime = IMISSV
- integer refyear = IMISSV
- integer refmonth = 0
- integer refday = 0
- integer refhour = 0
- integer refmin = 0
- integer refsec = 0
- logical refdatespecified = .FALSE.
- character(len=64) begdatetime = BLANK
- integer begdate = IMISSV
- integer begtime = IMISSV
- integer begyear = IMISSV
- integer begmonth = 0
- integer begday = 0
- integer beghour = 0
- integer begmin = 0
- integer begsec = 0
- logical begdatespecified = .FALSE.
- character(len=64) enddatetime = BLANK
- integer enddate = IMISSV
- integer endtime = IMISSV
- integer endyear = IMISSV
- integer endmonth = 0
- integer endday = 0
- integer endhour = 0
- integer endmin = 0
- integer endsec = 0
- logical enddatespecified = .FALSE.
- real(sz) begsimtime = RMISSV
- real(sz) endsimtime = RMISSV
- logical begsimspecified = .FALSE.
- logical endsimspecified = .FALSE.
- character(len=1) unittime = 'S'
- real(sz) outdt = RMISSV
- integer noutdt = IMISSV
- real(sz) mdoutdt = RMISSV
- real(sz) mdbegsimtime = RMISSV
- real(sz) mdendsimtime = RMISSV
- logical outfilenamespecified = .FALSE.

- character(len=fnamelen) outfilename = BLANK
- integer ncshuffle = 0
- integer ncdeflate = 0
- integer ncdlevel = 0
- character(len=20), parameter def_ncnam_pres = 'P'
- character(len=20), parameter def_ncnam_wndx = 'uwnd'
- character(len=20), parameter def_ncnam_wndy = 'vwnd'
- character(len=20) ncvarnam_pres = DEF_NCNAM_PRES
- character(len=20) ncvarnam_wndx = DEF_NCNAM_WNDX
- character(len=20) ncvarnam_wndy = DEF_NCNAM_WNDY
- integer modeltype = IMISSV
- logical writeparams = .FALSE.
- real(sz), dimension(:), allocatable wvelx
- real(sz), dimension(:), allocatable wvely
- real(sz), dimension(:), allocatable wpress
- real(sz), dimension(:), allocatable times

7.5.1 Function/Subroutine Documentation

7.5.1.1 airdensity()

```
real(sz) function pahm_global::airdensity (
        real(sz), intent(in) atmT,
        real(sz), intent(in) atmP,
        real(sz), intent(in) relHum )
```

This function calculates the density of the moist air.

See also

https://en.wikipedia.org/wiki/Density_of_air

Parameters

atmT	Air temperature (^0C)
atmP	Atmospheric pressure ($mbar$)
relHum	Relative humidity ($0-100$)

Returns

myValOut: The density of moist air (kg/m^3)

Definition at line 250 of file global.F90.

7.5.2 Variable Documentation

7.5.2.1 backgroundatmpress `real(sz) pahm_global::backgroundatmpress = DEFV_ATMPRESS`

Definition at line 117 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.2 basee `real(sz), parameter pahm_global::basee = 2.718281828459045_SZ`

Definition at line 78 of file global.F90.

Referenced by parwind::gethollandfields().

7.5.2.3 begdate `integer pahm_global::begdate = IMISSV`

Definition at line 142 of file global.F90.

7.5.2.4 begdatespecified `logical pahm_global::begdatespecified = .FALSE.`

Definition at line 150 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.5 begdatetime `character(len=64) pahm_global::begdatetime = BLANK`

Definition at line 141 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.6 begday `integer pahm_global::begday = 0`

Definition at line 146 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.7 beghour `integer pahm_global::beghour = 0`

Definition at line 147 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.8 begmin `integer pahm_global::begmin = 0`

Definition at line 148 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.9 begmonth `integer pahm_global::begmonth = 0`

Definition at line 145 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.10 begsec `integer pahm_global::begsec = 0`

Definition at line 149 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.11 begsimspecified `logical pahm_global::begsimspecified = .FALSE.`

Definition at line 167 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.12 begsimtime `real(sz) pahm_global::begsimtime = RMISSV`

Definition at line 165 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.13 begtime `integer pahm_global::begtime = IMISSV`

Definition at line 143 of file global.F90.

7.5.2.14 begyear `integer pahm_global::begyear = IMISSV`

Definition at line 144 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.15 besttrackfilename `character(len=fnamelen), dimension(:), allocatable pahm_global::besttrackfilename`

Definition at line 108 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), utilities::printmodelparams(), parwind::readbesttrackfile(), and parwind::readcsvbesttrackfile().

7.5.2.16 besttrackfilenamespecified `logical pahm_global::besttrackfilenamespecified = .FALSE.`

Definition at line 106 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.17 bladjustfac `real(sz) pahm_global::bladjustfac = DEFV_BLADJUSTFAC`

Definition at line 122 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.18 controlfilename `character(fnamelen) pahm_global::controlfilename = TRIM(ADJUSTL(PROG_↩ NAME_LOW)) // '_control.in'`

Definition at line 99 of file global.F90.

Referenced by pahm_drivermod::getprogramcmdlargs().

7.5.2.19 def_ncnam_pres `character(len=20), parameter pahm_global::def_ncnam_pres = 'P'`

Definition at line 190 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.20 def_ncnam_wndx `character(len=20), parameter pahm_global::def_ncnam_wndx = 'uwnd'`

Definition at line 190 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.21 def_ncnam_wndy `character(len=20), parameter pahm_global::def_ncnam_wndy = 'vwnd'`

Definition at line 190 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.22 defv_atmpress `real(sz), parameter pahm_global::defv_atmpress = 1013.25_SZ`

Definition at line 43 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.23 defv_bladjustfac `real(sz), parameter pahm_global::defv_bladjustfac = 0.9_SZ`

Definition at line 120 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.24 defv_gravity `real(sz), parameter pahm_global::defv_gravity = 9.80665_SZ`

Definition at line 42 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.25 defv_rhoair `real(sz), parameter pahm_global::defv_rhoair = 1.1478_SZ`

Definition at line 45 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.26 defv_rhowater `real(sz), parameter pahm_global::defv_rhowater = 1000.0000`

Definition at line 49 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.27 deg2rad `real(sz), parameter pahm_global::deg2rad = PI / 180.0_SZ`

Definition at line 76 of file global.F90.

Referenced by pahm_vortex::calcintensitychange(), utilities::cpptogeo::cpptogeo_1d(), utilities::cpptogeo::cpptogeo_scalar(), utilities::geotocpp::geotocpp_1d(), utilities::geotocpp::geotocpp_scalar(), parwind::gethollandfields(), pahm_vortex::newvortex(), pahm_vortex::newvortexfull(), pahm_vortex::rotate(), pahm_vortex::setvortex(), utilities::sphericaldistance::sphericaldistance_1d(), utilities::sphericaldistance::sphericaldistance_2d(), utilities::sphericaldistance::sphericaldistance_scalar(), utilities::sphericaldistanceharv(), utilities::sphericalfracpoint(), pahm_vortex::uvp(), pahm_vortex::uvpr(), pahm_vortex::uvtrans(), and pahm_vortex::uvtranspoint().

7.5.2.28 enddate `integer pahm_global::enddate = IMISSV`

Definition at line 154 of file global.F90.

7.5.2.29 enddatespecified `logical pahm_global::enddatespecified = .FALSE.`

Definition at line 162 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.30 enddatetime `character(len=64) pahm_global::enddatetime = BLANK`

Definition at line 153 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.31 endday `integer pahm_global::endday = 0`

Definition at line 158 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.32 endhour `integer pahm_global::endhour = 0`

Definition at line 159 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.33 endmin `integer pahm_global::endmin = 0`

Definition at line 160 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.34 endmonth `integer pahm_global::endmonth = 0`

Definition at line 157 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.35 endsec `integer pahm_global::endsec = 0`

Definition at line 161 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.36 endsimspecified `logical pahm_global::endsimspecified = .FALSE.`

Definition at line 168 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.37 endsimtime `real(sz) pahm_global::endsimtime = RMISSV`

Definition at line 166 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.38 endtime `integer pahm_global::endtime = IMISSV`

Definition at line 155 of file global.F90.

7.5.2.39 endyear `integer pahm_global::endyear = IMISSV`

Definition at line 156 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.40 gravity `real(sz) pahm_global::gravity = DEFV_GRAVITY`

Definition at line 114 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.41 kt2ms `real(sz), parameter pahm_global::kt2ms = NM2M / 3600.0_SZ`

Definition at line 83 of file global.F90.

Referenced by pahm_vortex::calcrmaxesfull(), pahm_vortex::newvortex(), pahm_vortex::newvortexfull(), parwind::processhollanddata(), pahm_vortex::uvp(), pahm_vortex::uvpr(), pahm_vortex::vhnocori(), pahm_vortex::vhwithcori(), and pahm_vortex::vhwithcorifull().

7.5.2.42 logfilename `character(len=fnamelen) pahm_global::logfilename = TRIM(ADJUSTL(PROG_NAME_↩ LOW)) // '_model.log'`

Definition at line 96 of file global.F90.

Referenced by pahm_messages::openlogfile().

7.5.2.43 lun_btrk `integer, parameter pahm_global::lun_btrk = 22`

Definition at line 30 of file global.F90.

Referenced by parwind::gethollandfields(), csv_module::initialize_csv_file(), parwind::readbesttrackfile(), and parwind::writebesttrackdata().

7.5.2.44 lun_btrk1 `integer, parameter pahm_global::lun_btrk1 = 23`

Definition at line 31 of file global.F90.

Referenced by parwind::gethollandfields(), parwind::readbesttrackfile(), and parwind::writebesttrackdata().

7.5.2.45 lun_ctrl `integer, parameter pahm_global::lun_ctrl = 10`

Definition at line 26 of file global.F90.

Referenced by utilities::readcontrolfile().

7.5.2.46 lun_inp `integer, parameter pahm_global::lun_inp = 14`

Definition at line 27 of file global.F90.

Referenced by pahm_mesh::readmeshasciifort14().

7.5.2.47 lun_inp1 `integer, parameter pahm_global::lun_inp1 = 15`

Definition at line 28 of file global.F90.

7.5.2.48 lun_log `integer, parameter pahm_global::lun_log = 35`

Definition at line 29 of file global.F90.

Referenced by pahm_messages::closelogfile(), pahm_messages::logmessage::logmessage_1(), pahm_messages::logmessage::logmess
and pahm_messages::openlogfile().

7.5.2.49 lun_out `integer, parameter pahm_global::lun_out = 25`

Definition at line 32 of file global.F90.

7.5.2.50 lun_out1 `integer, parameter pahm_global::lun_out1 = 26`

Definition at line 33 of file global.F90.

7.5.2.51 lun_screen `integer, parameter pahm_global::lun_screen = 6`

Definition at line 25 of file global.F90.

Referenced by pahm_messages::programhelp(), pahm_messages::programversion(), utilities::readcontrolfile(), pahm_messages::screenmessage::screenmessage_1(), and pahm_messages::screenmessage::screenmessage_2().

7.5.2.52 m2nm `real(sz), parameter pahm_global::m2nm = 1.0_SZ / NM2M`

Definition at line 82 of file global.F90.

Referenced by pahm_vortex::uvp().

7.5.2.53 mb2kpa `real(sz), parameter pahm_global::mb2kpa = 0.1_SZ`

Definition at line 87 of file global.F90.

Referenced by parwind::gethollandfields().

7.5.2.54 mb2pa `real(sz), parameter pahm_global::mb2pa = 100.0_SZ`

Definition at line 86 of file global.F90.

Referenced by pahm_vortex::calcrmaxesfull(), parwind::gethollandfields(), pahm_vortex::newvortex(), pahm_vortex::newvortexfull(), pahm_vortex::uvp(), and pahm_vortex::uvpr().

7.5.2.55 mdbegsimtime `real(sz) pahm_global::mdbegsimtime = RMISSV`

Definition at line 178 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.56 mdendsimtime `real(sz) pahm_global::mdendsimtime = RMISSV`

Definition at line 179 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.57 mdoutdt `real(sz) pahm_global::mdoutdt = RMISSV`

Definition at line 177 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.58 meshfileform `character(len=64) pahm_global::meshfileform = BLANK`

Definition at line 104 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), utilities::printmodelparams(), and pahm_mesh::readmesh().

7.5.2.59 meshfilename `character(len=fnamelen) pahm_global::meshfilename = BLANK`

Definition at line 102 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), utilities::printmodelparams(), pahm_mesh::readmesh(), and pahm_mesh::readmeshasciifort14().

7.5.2.60 meshfilenamespecified `logical pahm_global::meshfilenamespecified = .FALSE.`

Definition at line 101 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and pahm_mesh::readmesh().

7.5.2.61 meshfiletype `character(len=64) pahm_global::meshfiletype = BLANK`

Definition at line 103 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), utilities::printmodelparams(), and pahm_mesh::readmesh().

7.5.2.62 modeltype `integer pahm_global::modeltype = IMISSV`

Definition at line 198 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_drivermod::pahm_run(), and utilities::printmodelparams().

7.5.2.63 ms2kt `real(sz), parameter pahm_global::ms2kt = 1.0_SZ / KT2MS`

Definition at line 84 of file global.F90.

Referenced by pahm_vortex::vhnocori(), pahm_vortex::vhwithcori(), and pahm_vortex::vhwithcorifull().

7.5.2.64 nbtrfiles `integer pahm_global::nbtrfiles = IMISSV`

Definition at line 107 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), utilities::printmodelparams(), parwind::processhollanddata(), parwind::readbesttrackfile(), and parwind::readcsvbesttrackfile().

7.5.2.65 ncdeflate `integer pahm_global::ncdeflate = 0`

Definition at line 184 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.66 ncdlevel `integer pahm_global::ncdlevel = 0`

Definition at line 185 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.67 ncshuffle `integer pahm_global::ncshuffle = 0`

Definition at line 183 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.68 ncvarnam_pres `character(len=20) pahm_global::ncvarnam_pres = DEF_NCNAM_PRES`

Definition at line 194 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.69 ncvarnam_wndx `character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAM_WNDX`

Definition at line 194 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.70 ncvarnam_wndy `character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAM_WNDY`

Definition at line 194 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.71 nm2m `real(sz), parameter pahm_global::nm2m = 1852.0_SZ`

Definition at line 81 of file global.F90.

Referenced by pahm_vortex::calcrmaxesfull(), parwind::processhollanddata(), pahm_vortex::uvp(), pahm_vortex::uvpr(), pahm_vortex::vhwithcori(), and pahm_vortex::vhwithcorifull().

7.5.2.72 noutdt `integer pahm_global::noutdt = IMISSV`

Definition at line 176 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), pahm_drivermod::pahm_init(), and utilities::printmodelparams().

7.5.2.73 omega `real(sz), parameter pahm_global::omega = 2.0_SZ * PI / 86164.2_SZ`

Definition at line 85 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_vortex::newvortex(), pahm_vortex::newvortexfull(), and pahm_vortex::setvortex().

7.5.2.74 one2ten `real(sz), parameter pahm_global::one2ten = 0.8928_SZ`

Definition at line 72 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_vortex::uvp(), and pahm_vortex::uvpr().

7.5.2.75 outdt `real(sz) pahm_global::outdt = RMISSV`

Definition at line 175 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.76 outfilename `character(len=fnamelen) pahm_global::outfilename = BLANK`

Definition at line 182 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_drivermod::pahm_run(), and utilities::printmodelparams().

7.5.2.77 outfilenamespecified `logical pahm_global::outfilenamespecified = .FALSE.`

Definition at line 181 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and pahm_drivermod::pahm_run().

7.5.2.78 pi `real(sz), parameter pahm_global::pi = 3.141592653589793_SZ`

Definition at line 75 of file global.F90.

7.5.2.79 rad2deg `real(sz), parameter pahm_global::rad2deg = 180.0_SZ / PI`

Definition at line 77 of file global.F90.

Referenced by parwind::gethollandfields(), utilities::sphericalfracpoint(), and pahm_vortex::uvp().

7.5.2.80 rearth `real(sz), parameter pahm_global::rearth = 6378206.4_SZ`

Definition at line 80 of file global.F90.

Referenced by utilities::cpptogeo::cpptogeo_1d(), utilities::cpptogeo::cpptogeo_scalar(), utilities::geotocpp::geotocpp_1d(), utilities::geotocpp::geotocpp_scalar(), pahm_netcdfio::initadcircnetcdfoutfile(), utilities::sphericaldistance::sphericaldistance_1d(), utilities::sphericaldistance::sphericaldistance_2d(), utilities::sphericaldistance::sphericaldistance_scalar(), utilities::sphericaldistanceharv(), utilities::sphericalfracpoint(), and pahm_vortex::uvp().

7.5.2.81 refdate `integer pahm_global::refdate = IMISSV`

Definition at line 130 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.82 refdatespecified `logical pahm_global::refdatespecified = .FALSE.`

Definition at line 138 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.83 refdatetime `character(len=64) pahm_global::refdatetime = BLANK`

Definition at line 129 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), and utilities::printmodelparams().

7.5.2.84 refday `integer pahm_global::refday = 0`

Definition at line 134 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.85 refhour `integer pahm_global::refhour = 0`

Definition at line 135 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.86 refmin `integer pahm_global::refmin = 0`

Definition at line 136 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.87 refmonth `integer pahm_global::refmonth = 0`

Definition at line 133 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.88 refsec `integer pahm_global::refsec = 0`

Definition at line 137 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.89 reftime `integer pahm_global::reftime = IMISSV`

Definition at line 131 of file global.F90.

Referenced by utilities::checkcontrolfileinputs().

7.5.2.90 refyear `integer pahm_global::refyear = IMISSV`

Definition at line 132 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), utilities::printmodelparams(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.5.2.91 rhoair `real(sz) pahm_global::rhoair = DEFV_RHOAIR`

Definition at line 116 of file global.F90.

Referenced by pahm_vortex::calcrmaxesfull(), utilities::checkcontrolfileinputs(), parwind::gethollandfields(), pahm_vortex::newvortex(), pahm_vortex::newvortexfull(), and utilities::printmodelparams().

7.5.2.92 rhowater `real(sz) pahm_global::rhowater = DEFV_RHOWATER`

Definition at line 115 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), parwind::gethollandfields(), and utilities::printmodelparams().

7.5.2.93 ten2one `real(sz), parameter pahm_global::ten2one = 1.0_SZ / 0.8928_SZ`

Definition at line 73 of file global.F90.

7.5.2.94 times `real(sz), dimension(:), allocatable pahm_global::times`

Definition at line 217 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and pahm_drivermod::pahm_run().

7.5.2.95 title `character(len=512) pahm_global::title = BLANK`

Definition at line 112 of file global.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.96 unittime `character(len=1) pahm_global::unittime = 'S'`

Definition at line 170 of file global.F90.

Referenced by utilities::checkcontrolfileinputs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::printmodelparams().

7.5.2.97 windreduction `real(sz) pahm_global::windreduction = DEFV_BLADJUSTFAC`

Definition at line 121 of file global.F90.

Referenced by pahm_vortex::uvp(), and pahm_vortex::uvpr().

7.5.2.98 wpress `real(sz), dimension(:), allocatable pahm_global::wpress`

Definition at line 216 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), pahm_drivermod::pahm_run(), and pahm_netcdfio::writenetcdfrecord().

7.5.2.99 writeparams `logical pahm_global::writeparams = .FALSE.`

Definition at line 205 of file global.F90.

Referenced by utilities::printmodelparams().

7.5.2.100 wvelx `real(sz), dimension(:), allocatable pahm_global::wvelx`

Definition at line 216 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), pahm_drivermod::pahm_run(), and pahm_netcdfio::writenetcdfrecord().

7.5.2.101 wvely `real(sz), dimension(:), allocatable pahm_global::wvely`

Definition at line 216 of file global.F90.

Referenced by parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), pahm_drivermod::pahm_run(), and pahm_netcdfio::writenetcdfrecord().

7.6 pahm_mesh Module Reference

Functions/Subroutines

- subroutine readmesh ()

  Reads an input mesh file for the specified supported model type.
- subroutine readmeshasciifort14 ()

  Reads the ADCIRC fort.14 mesh file.
- subroutine allocatenodalandelementalarrays ()

  Allocates memory to mesh arrays.

Variables

- character(len=80) agrid
- integer np = IMISSV
- integer ne = IMISSV
- integer ics
- real(sz), dimension(:), allocatable dp
- integer, dimension(:), allocatable nfn
- integer, dimension(:, :), allocatable nm
- real(sz), dimension(:), allocatable slam
- real(sz), dimension(:), allocatable sfea
- real(sz), dimension(:), allocatable xcslam
- real(sz), dimension(:), allocatable ycsfea
- real(sz) slam0 = RMISSV
- real(sz) sfea0 = RMISSV
- integer, parameter maxfacenodes = 5
- logical ismeshok = .FALSE.

7.6.1 Function/Subroutine Documentation

7.6.1.1 allocatenodalandelementalarrays() `subroutine pahm_mesh::allocatenodalandelementalarrays`

Allocates memory to mesh arrays.

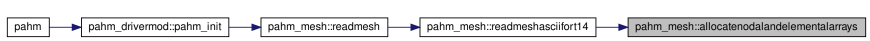
Mesh related memory allocation for any array that is dimensioned by the number of nodes in the mesh or the number of elements in the mesh.

Definition at line 301 of file mesh.F90.

References dp, maxfacenodes, ne, nfn, nm, np, sfea, slam, xcslam, and ycsfea.

Referenced by readmeshasciifort14().

Here is the caller graph for this function:



7.6.1.2 readmesh() subroutine pahm_mesh::readmesh

Reads an input mesh file for the specified supported model type.

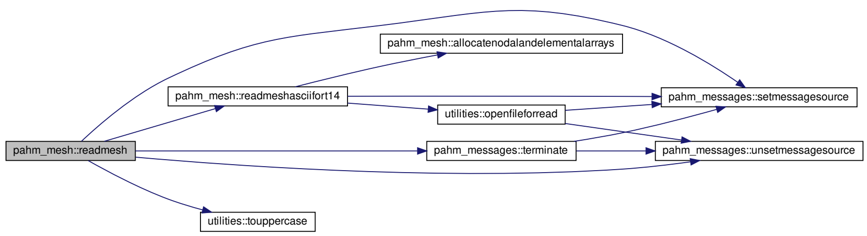
Read the mesh file for the specified model type (meshFileType) and in ASCII or NetCDF format (if applicable).

Definition at line 69 of file mesh.F90.

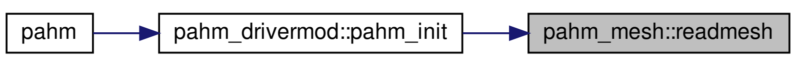
References pahm_messages::error, pahm_global::meshfileform, pahm_global::meshfilename, pahm_global::meshfilenamespecified, pahm_global::meshfiletype, readmeshasciifort14(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_messages::terminate(), utilities::touppercase(), and pahm_messages::unsetmessagesource().

Referenced by pahm_drivermod::pahm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.1.3 readmeshasciifort14() subroutine pahm_mesh::readmeshasciifort14

Reads the ADCIRC fort.14 mesh file.

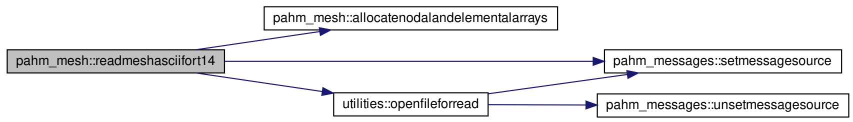
Reads the ADCIRC fort.14 mesh file and sets all mesh variables and arrays.

Definition at line 170 of file mesh.F90.

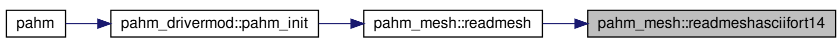
References agrid, allocatenodalandelementalarrays(), pahm_messages::info, ismeshok, pahm_global::lun_inp, maxfacenodes, pahm_global::meshfilename, ne, np, utilities::openfileforread(), pahm_messages::setmessagesource(), sfea, sfea0, slam, slam0, xcslam, and ycsfea.

Referenced by readmesh().

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.2 Variable Documentation

7.6.2.1 agrid `character(len=80) pahm_mesh::agrid`

Definition at line 32 of file mesh.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.2 dp `real(sz), dimension(:), allocatable pahm_mesh::dp`

Definition at line 36 of file mesh.F90.

Referenced by allocatenodalandelementalarrays().

7.6.2.3 ics `integer pahm_mesh::ics`

Definition at line 35 of file mesh.F90.

7.6.2.4 ismeshok `logical pahm_mesh::ismeshok = .FALSE.`

Definition at line 51 of file mesh.F90.

Referenced by parwind::gethollandfields(), and readmeshasciifort14().

7.6.2.5 maxfacenodes `integer, parameter pahm_mesh::maxfacenodes = 5`

Definition at line 48 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), and readmeshasciifort14().

7.6.2.6 ne `integer pahm_mesh::ne = IMISSV`

Definition at line 34 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.7 nfn `integer, dimension(:), allocatable pahm_mesh::nfn`

Definition at line 37 of file mesh.F90.

Referenced by allocatenodalandelementalarrays().

7.6.2.8 nm `integer, dimension(:, :), allocatable pahm_mesh::nm`

Definition at line 38 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), and pahm_netcdfio::initadcircnetcdfoutfile().

7.6.2.9 np `integer pahm_mesh::np = IMISSV`

Definition at line 33 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.10 sfea `real(sz), dimension(:), allocatable pahm_mesh::sfea`

Definition at line 40 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.11 sfea0 `real(sz) pahm_mesh::sfea0 = RMISSV`

Definition at line 45 of file mesh.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.12 slam `real(sz), dimension(:), allocatable pahm_mesh::slam`

Definition at line 39 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.13 slam0 `real(sz) pahm_mesh::slam0 = RMISSV`

Definition at line 44 of file mesh.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.14 xcslam `real(sz), dimension(:), allocatable pahm_mesh::xcslam`

Definition at line 41 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.6.2.15 ycsfea `real(sz), dimension(:), allocatable pahm_mesh::ycsfea`

Definition at line 42 of file mesh.F90.

Referenced by allocatenodalandelementalarrays(), parwind::gethollandfields(), pahm_netcdfio::initadcircnetcdfoutfile(), and readmeshasciifort14().

7.7 pahm_messages Module Reference

Data Types

- interface allmessage
- interface logmessage
- interface screenmessage

Functions/Subroutines

- subroutine initlogging ()

  Initializes logging levels.
- subroutine openlogfile ()

  Opens the log file for writting.
- subroutine closelogfile ()

  Closes an opened log file.
- subroutine screenmessage_1 (message)

  General purpose subroutine to write a message to the screen.
- subroutine screenmessage_2 (level, message)
- subroutine logmessage_1 (message)

  General purpose subroutine to write a message to the log file.
- subroutine logmessage_2 (level, message)
- subroutine allmessage_1 (message)

  General purpose subroutine to write a message to both the screen and the log file.
- subroutine allmessage_2 (level, message)
- subroutine setmessagesource (source)

  Sets the name of the subroutine that is writing log and/or screen messages.
- subroutine unsetmessagesource ()

  Removes the name of the subroutine that is no longer active.
- subroutine programversion ()

  Prints on the screen the versioning information of the program.
- subroutine programhelp ()

  Prints on the screen the help system of the program.
- subroutine terminate ()

  Terminates the calling program when a fatal error is encountered.

Variables

- integer nscreen = 1
- integer, parameter debug = -1
- integer, parameter echo = 0
- integer, parameter info = 1
- integer, parameter warning = 2
- integer, parameter error = 3
- character(len=10), dimension(5) loglevelnames
- character(len=50), dimension(100) messagesources
- character(len=1024) scratchmessage
- character(len=1024) scratchformat
- integer sourcenumber
- logical logfileopened = .FALSE.
- logical loginitcalled = .FALSE.

7.7.1 Function/Subroutine Documentation

7.7.1.1 allmessage_1()

```
subroutine pahm_messages::allmessage_1 (
        character(len=*), intent(in) message )
```

General purpose subroutine to write a message to both the screen and the log file.

Definition at line 309 of file messages.F90.

7.7.1.2 allmessage_2()

```
subroutine pahm_messages::allmessage_2 (
        integer, intent(in) level,
        character(len=*), intent(in) message )
```

Definition at line 321 of file messages.F90.

7.7.1.3 closelogfile()

```
subroutine pahm_messages::closelogfile
```

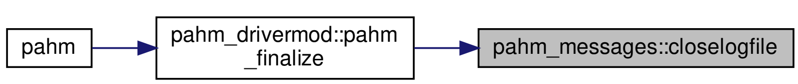
Closes an opened log file.

Definition at line 148 of file messages.F90.

References logfileopened, and pahm_global::lun_log.

Referenced by pahm_drivermod::pahm_finalize().

Here is the caller graph for this function:



7.7.1.4 initlogging() `subroutine pahm_messages::initlogging`

Initializes logging levels.

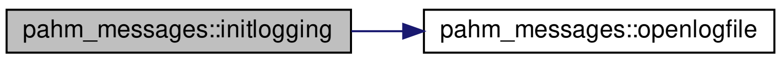
Initialize the names for the logging levels and the counter for the current subroutine.

Definition at line 81 of file messages.F90.

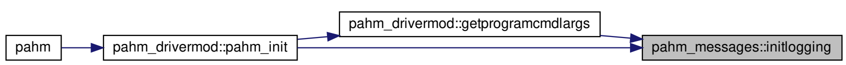
References loginitcalled, loglevelnames, openlogfile(), and sourcenumber.

Referenced by pahm_drivermod::getprogramcmdlargs(), and pahm_drivermod::pahm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.1.5 logmessage_1() `subroutine pahm_messages::logmessage_1 ( character(len=*), intent(in) message )`

General purpose subroutine to write a message to the log file.

This subroutine assumes that the global variable "caller" has been set to the name of the subroutine calling it. Therefore, the SetMessageSource subroutine must be called at the beginning of the subroutine that calls this one, and Unset↩ MessageSource must be called at the end.

Definition at line 245 of file messages.F90.

7.7.1.6 logmessage_2() `subroutine pahm_messages::logmessage_2 ( integer, intent(in) level, character(len=*), intent(in) message )`

Definition at line 269 of file messages.F90.

7.7.1.7 openlogfile() `subroutine pahm_messages::openlogfile`

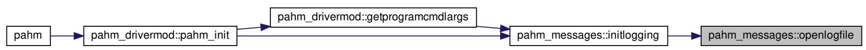
Opens the log file for writting.

Definition at line 113 of file messages.F90.

References error, pahm_global::logfilename, logfileopened, pahm_global::lun_log, and scratchmessage.

Referenced by initlogging().

Here is the caller graph for this function:



7.7.1.8 programhelp() `subroutine pahm_messages::programhelp`

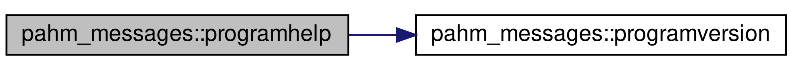
Prints on the screen the help system of the program.

Definition at line 427 of file messages.F90.

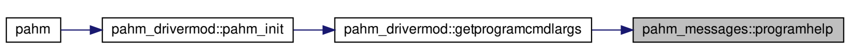
References pahm_global::lun_screen, and programversion().

Referenced by pahm_drivermod::getprogramcmdlargs().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.1.9 programversion() `subroutine pahm_messages::programversion`

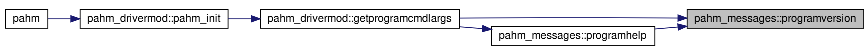
Prints on the screen the versioning information of the program.

Definition at line 397 of file messages.F90.

References pahm_global::lun_screen.

Referenced by pahm_drivermod::getprogramcmdlargs(), and programhelp().

Here is the caller graph for this function:



7.7.1.10 screenmessage_1() `subroutine pahm_messages::screenmessage_1 ( character(len=*), intent(in) message )`

General purpose subroutine to write a message to the screen.

General purpose subroutine to write a message to the screen with a certain "logging level", and subject to the user's selection of where to write screen output.

This subroutine assumes that the global variable "caller" has been set to the name of the subroutine calling it. Therefore, the SetMessageSource subroutine must be called at the beginning of the subroutine that calls this one, and Unset←MessageSource must be called at the end.

Definition at line 177 of file messages.F90.

7.7.1.11 screenmessage_2() `subroutine pahm_messages::screenmessage_2 ( integer, intent(in) level, character(len=*), intent(in) message )`

Definition at line 201 of file messages.F90.

7.7.1.12 setmessagesource() `subroutine pahm_messages::setmessagesource ( character(len=*), intent(in) source )`

Sets the name of the subroutine that is writing log and/or screen messages.

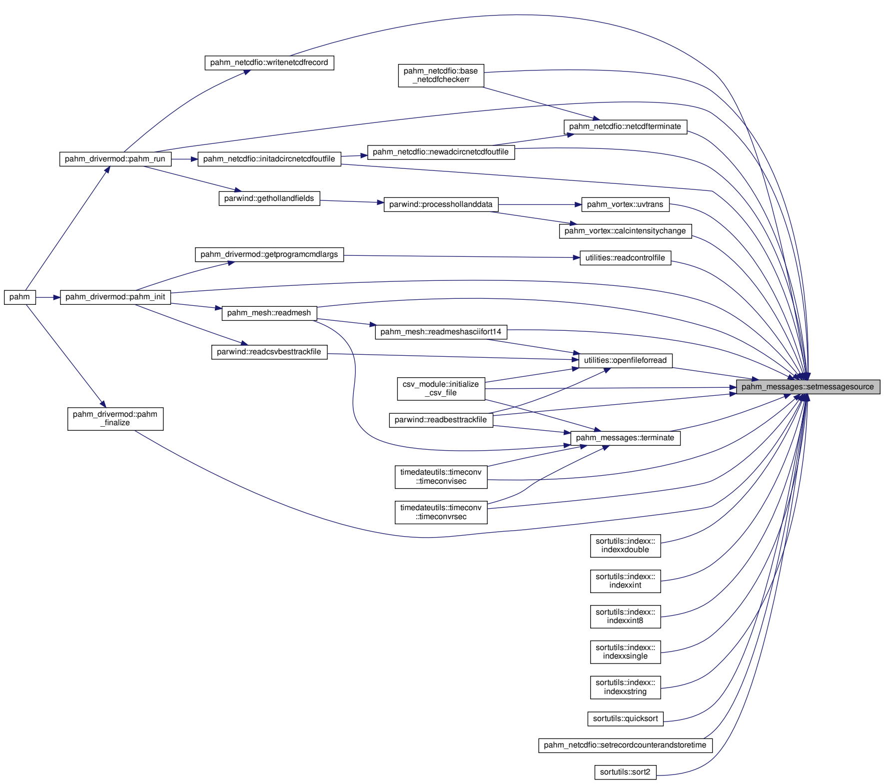
Sets the name of the subroutine that is writing log and/or screen messages. Must use at the start of any subroutine that calls ScreenMessage, LogMessage, or AllMessage.

Definition at line 349 of file messages.F90.

References messagesources, and sourcenumber.

Referenced by pahm_netcdfio::base_netcdfcheckerr(), pahm_vortex::calcintensitychange(), sortutils::indexx::indexxdouble(), sortutils::indexx::indexxint(), sortutils::indexx::indexxint8(), sortutils::indexx::indexxsingle(), sortutils::indexx::indexxstring(), pahm_netcdfio::initadcircnetcdfoutfile(), csv_module::initialize_csv_file(), pahm_netcdfio::netcdfterminate(), pahm_netcdfio::newadcircnetc utilities::openfileforread(), pahm_drivermod::pahm_finalize(), pahm_drivermod::pahm_init(), pahm_drivermod::pahm_run(), sortutils::quicksort(), parwind::readbesttrackfile(), utilities::readcontrolfile(), pahm_mesh::readmesh(), pahm_mesh::readmeshasciifort14(), pahm_netcdfio::setrecordcounterandstoretime(), sortutils::sort2(), terminate(), timedateutils::timeconv::timeconvisec(), timedateutils::timeconv::timeconvrsec(), pahm_vortex::uvtrans(), and pahm_netcdfio::writenetcdfrecord().

Here is the caller graph for this function:



7.7.1.13 terminate() `subroutine pahm_messages::terminate`

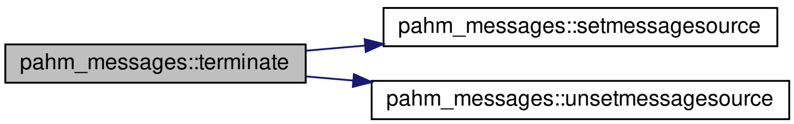
Terminates the calling program when a fatal error is encountered.

Definition at line 452 of file messages.F90.

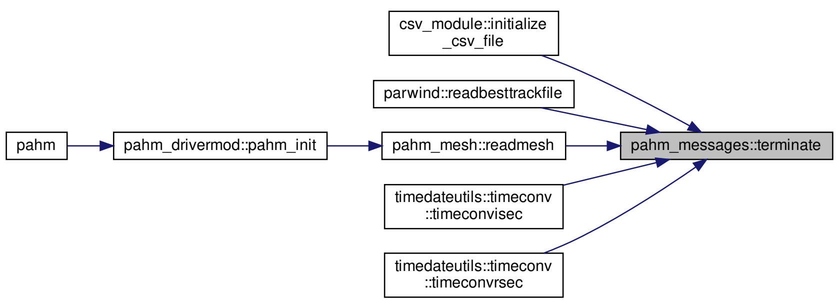
References error, setmessagesource(), and unsetmessagesource().

Referenced by csv_module::initialize_csv_file(), parwind::readbesttrackfile(), pahm_mesh::readmesh(), timedateutils::timeconv::timeconvis
and timedateutils::timeconv::timeconvrsec().

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.1.14 unsetmessagesource() `subroutine pahm_messages::unsetmessagesource`

Removes the name of the subroutine that is no longer active.

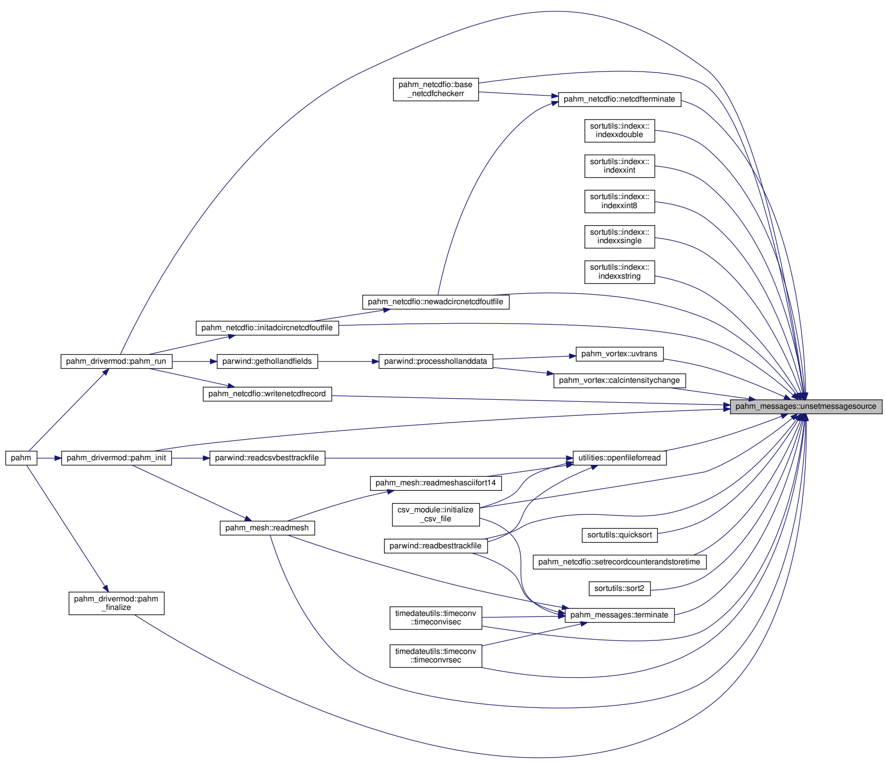
Removes the name of the subroutine that is no longer writing log and/or screen messages. Must use at the end of any subroutine that calls ScreenMessage, LogMessage, or AllMessage.

Definition at line 376 of file messages.F90.

References sourcenumber.

Referenced by pahm_netcdfio::base_netcdfcheckerr(), pahm_vortex::calcintensitychange(), sortutils::indexx::indexxdouble(), sortutils::indexx::indexxint(), sortutils::indexx::indexxint8(), sortutils::indexx::indexxsingle(), sortutils::indexx::indexxstring(), pahm_netcdfio::initadcircnetcdfoutfile(), csv_module::initialize_csv_file(), pahm_netcdfio::netcdfterminate(), pahm_netcdfio::newadcircnetc utilities::openfileforread(), pahm_drivermod::pahm_finalize(), pahm_drivermod::pahm_init(), pahm_drivermod::pahm_run(), sortutils::quicksort(), parwind::readbesttrackfile(), pahm_mesh::readmesh(), pahm_netcdfio::setrecordcounterandstoretime(), sortutils::sort2(), terminate(), timedateutils::timeconv::timeconvisec(), timedateutils::timeconv::timeconvrsec(), pahm_vortex::uvtrans(), and pahm_netcdfio::writenetcdfrecord().

Here is the caller graph for this function:



7.7.2 Variable Documentation

7.7.2.1 debug `integer, parameter pahm_messages::debug = -1`

Definition at line 30 of file messages.F90.

7.7.2.2 echo `integer, parameter pahm_messages::echo = 0`

Definition at line 31 of file messages.F90.

7.7.2.3 error `integer, parameter pahm_messages::error = 3`

Definition at line 34 of file messages.F90.

Referenced by pahm_netcdfio::base_netcdfcheckerr(), pahm_vortex::calcintensitychange(), sortutils::indexx::indexxdouble(), sortutils::indexx::indexxint(), sortutils::indexx::indexxint8(), sortutils::indexx::indexxsingle(), sortutils::indexx::indexxstring(), csv_module::initialize_csv_file(), pahm_netcdfio::newadcircnetcdfoutfile(), utilities::openfileforread(), openlogfile(), pahm_drivermod::pahm_run(), sortutils::quicksort(), parwind::readbesttrackfile(), pahm_mesh::readmesh(), sortutils::sort2(), terminate(), timedateutils::timeconv::timeconvisec(), timedateutils::timeconv::timeconvrsec(), and pahm_vortex::uvtrans().

7.7.2.4 info `integer, parameter pahm_messages::info = 1`

Definition at line 32 of file messages.F90.

Referenced by pahm_netcdfio::base_netcdfcheckerr(), csv_module::initialize_csv_file(), pahm_netcdfio::netcdfterminate(), pahm_netcdfio::newadcircnetcdfoutfile(), utilities::openfileforread(), parwind::readbesttrackfile(), pahm_mesh::readmeshasciifort14(), and pahm_netcdfio::setrecordcounterandstoretime().

7.7.2.5 logfileopened `logical pahm_messages::logfileopened = .FALSE.`

Definition at line 43 of file messages.F90.

Referenced by closelogfile(), pahm_messages::logmessage::logmessage_1(), pahm_messages::logmessage::logmessage_2(), and openlogfile().

7.7.2.6 loginitcalled `logical pahm_messages::loginitcalled = .FALSE.`

Definition at line 44 of file messages.F90.

Referenced by initlogging(), pahm_messages::logmessage::logmessage_1(), pahm_messages::logmessage::logmessage_2(), pahm_messages::screenmessage::screenmessage_1(), and pahm_messages::screenmessage::screenmessage_2().

7.7.2.7 loglevelnames `character(len=10), dimension(5) pahm_messages::loglevelnames`

Definition at line 36 of file messages.F90.

Referenced by initlogging(), pahm_messages::logmessage::logmessage_2(), and pahm_messages::screenmessage::screenmessage_2(

7.7.2.8 messagesources `character(len=50), dimension(100) pahm_messages::messagesources`

Definition at line 37 of file messages.F90.

Referenced by pahm_messages::logmessage::logmessage_1(), pahm_messages::logmessage::logmessage_2(), pahm_messages::screenmessage::screenmessage_1(), pahm_messages::screenmessage::screenmessage_2(), and setmessagesource().

7.7.2.9 nscreen `integer pahm_messages::nscreen = 1`

Definition at line 27 of file messages.F90.

Referenced by pahm_messages::screenmessage::screenmessage_1(), and pahm_messages::screenmessage::screenmessage_2().

7.7.2.10 scratchformat `character(len=1024) pahm_messages::scratchformat`

Definition at line 39 of file messages.F90.

Referenced by pahm_netcdfio::setrecordcounterandstoretime().

7.7.2.11 scratchmessage `character(len=1024) pahm_messages::scratchmessage`

Definition at line 38 of file messages.F90.

Referenced by pahm_vortex::calcintensitychange(), sortutils::indexx::indexxdouble(), sortutils::indexx::indexxint(), sortutils::indexx::indexxint8(), sortutils::indexx::indexxsingle(), sortutils::indexx::indexxstring(), csv_module::initialize_csv_file(), pahm_netcdfio::newadcircnetcdfoutfile(), utilities::openfileforread(), openlogfile(), pahm_drivermod::pahm_run(), sortutils::quicksort(), parwind::readbesttrackfile(), pahm_mesh::readmesh(), pahm_netcdfio::setrecordcounterandstoretime(), sortutils::sort2(), timedateutils::timeconv::timeconvisec(), timedateutils::timeconv::timeconvrsec(), and pahm_vortex::uvtrans().

7.7.2.12 sourcenumber `integer pahm_messages::sourcenumber`

Definition at line 40 of file messages.F90.

Referenced by initlogging(), pahm_messages::logmessage::logmessage_1(), pahm_messages::logmessage::logmessage_2(), pahm_messages::screenmessage::screenmessage_1(), pahm_messages::screenmessage::screenmessage_2(), setmessagesource(), and unsetmessagesource().

7.7.2.13 warning `integer, parameter pahm_messages::warning = 2`

Definition at line 33 of file messages.F90.

7.8 pahm_netcdfio Module Reference

Data Types

- type filedata_t
- type timedata_t

Functions/Subroutines

- subroutine initadcircnetcdfoutfile (adcircOutFile)

  Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.
- subroutine newadcircnetcdfoutfile (ncID, adcircOutFile)

  Creates a new NetCDF data file and puts it in define mode.
- subroutine base_netcdfcheckerr (ierr, file, line)

  Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file.
- subroutine netcdfterminate ()

  Terminates the program on NetCDF error.
- subroutine writenetcdfrecord (adcircOutFile, timeLoc)

  Writes data to the NetCDF file.
- subroutine setrecordcounterandstoretime (ncID, f, t)

  Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

Variables

- type(timedata_t), save mytime

7.8.1 Detailed Description

Author

PanagiotisVelissariou `panagiotis.velissariou@noaa.gov`

7.8.2 Function/Subroutine Documentation

7.8.2.1 base_netcdfcheckerr()

```
subroutine pahm_netcdfio::base_netcdfcheckerr (
        integer, intent(in) ierr,
        character(len=*), intent(in) file,
        integer, intent(in) line )
```

Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file.

Checks the return value from netCDF calls.

On Input:

ierr The error from a NetCDF library call.

Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file and then terminates the program.

Parameters

ierr	The error status from a NetCDF library call
file	The name of the file the error occured
line	The line number of the file the error occured

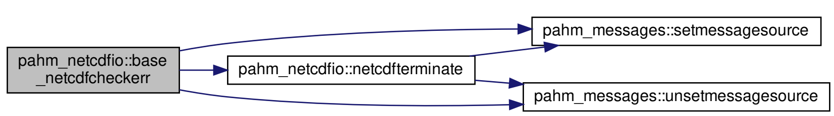
Returns

adcircOutFile: The renamed input file ncID: The id of the newly created file

Definition at line 697 of file netcdfio-nems.F90.

References pahm_messages::error, pahm_messages::info, netcdfterminate(), pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



7.8.2.2 initadcircnetcdfoutfile()

```
subroutine pahm_netcdfio::initadcircnetcdfoutfile (
        character(len=*), intent(inout) adcircOutFile )
```

Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.

Initializes a new NetCDF data file and puts it in define mode.

On input: adcircOutFile The name of the file to be initialized. The file is first created by calling NewAdcircNetCDFOutFile

On output: adcircOutFile The renamed input file

Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.

Parameters

adcircOutFile	The name of the file to be initialized. The file is first created by calling NewAdcircNetCDFOutFile.

Returns

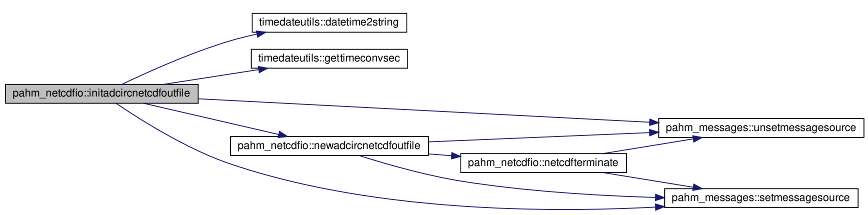
adcircOutFile: The renamed input file.

Definition at line 114 of file netcdfio-nems.F90.

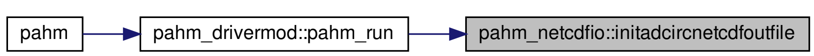
References pahm_mesh::agrid, timedateutils::datetime2string(), timedateutils::gettimeconvsec(), pahm_sizes::imissv, pahm_global::ncdeflate, pahm_global::ncdlevel, pahm_global::ncshuffle, pahm_global::ncvarnam_pres, pahm_global::ncvarnam_wndx, pahm_global::ncvarnam_wndy, pahm_mesh::ne, newadcircnetcdfoutfile(), pahm_mesh::nm, pahm_mesh::np, pahm_global::rearth, pahm_global::refday, pahm_global::refhour, pahm_global::refmin, pahm_global::refmonth, pahm_global::refsec, pahm_global::refyear, pahm_sizes::rmissv, pahm_messages::setmessagesource(), pahm_mesh::sfea, pahm_mesh::sfea0, pahm_mesh::slam, pahm_mesh::slam0, pahm_global::times, pahm_global::title, pahm_global::unittime, pahm_messages::unsetmessagesource(), pahm_global::wpress, pahm_global::wvelx, pahm_global::wvely, pahm_mesh::xcslam, and pahm_mesh::ycsfea.

Referenced by pahm_drivermod::pahm_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.2.3 netcdfterminate() `subroutine pahm_netcdfio::netcdfterminate`

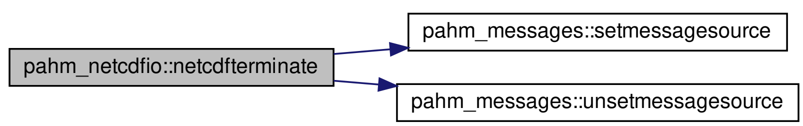
Terminates the program on NetCDF error.

Definition at line 727 of file netcdfio-nems.F90.

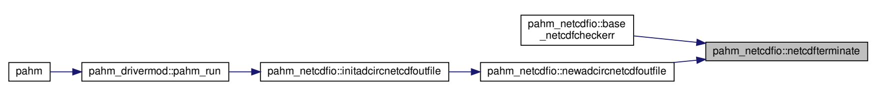
References pahm_messages::info, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by base_netcdfcheckerr(), and newadcircnetcdfoutfile().

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.2.4 newadcircnetcdfoutfile() `subroutine pahm_netcdfio::newadcircnetcdfoutfile (`

```
integer, intent(out) ncID,
character(len=*), intent(inout) adcircOutFile )
```

Creates a new NetCDF data file and puts it in define mode.

On input: adcircOutFile The name of the file to be created; the file extension is replaced by .nc or .nc4. If a file with the same name exists, it is renamed to: adcircOutFile.ext-YYYYMMDDhhmmss

On output: adcircOutFile The renamed input file ncID The id of the newly created file

Creates a new NetCDF data file and puts it in define mode. The file extension is replaced by .nc or .nc4. If a file with the same name exists, it is renamed to: adcircOutFile.ext-YYYYMMDDhhmmss

Parameters

ncID	The NetCDF ID of the file to be created (output)
adcircOutFile	The name of the file to be created (input/output)

Returns

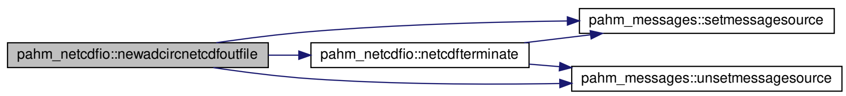
adcircOutFile: The renamed input file ncID: The id of the newly created file

Definition at line 607 of file netcdfio-nems.F90.

References pahm_messages::error, pahm_messages::info, netcdfterminate(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by initadcircnetcdfoutfile().

Here is the call graph for this function:



Here is the caller graph for this function:

7.8.2.5 setrecordcounterandstoretime()

```
subroutine pahm_netcdfio::setrecordcounterandstoretime (
        integer, intent(in) ncID,
        type(filedata_t), intent(inout) f,
        type(timedata_t), intent(inout) t )
```

Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

Sets the record counter.

On input: ncID The ID of the NetCDF file f The file structure t The time structure

On output: f The updated file structure t The updated time structure

Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

Parameters

ncID	The ID of the NetCDF file
f	The file structure
t	The time structure

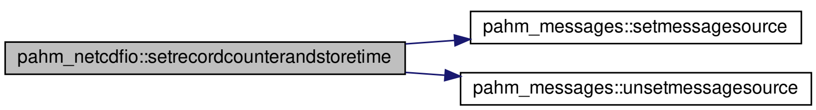
Returns

f: The updated file structure t: The updated time structure

Definition at line 826 of file netcdfio-nems.F90.

References pahm_messages::info, pahm_messages::scratchformat, pahm_messages::scratchmessage, pahm_messages::setmessages
and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



7.8.2.6 writenetcdfrecord()

```
subroutine pahm_netcdfio::writenetcdfrecord (
    character(len=*), intent(in) adcircOutFile,
    integer timeLoc )
```

Writes data to the NetCDF file.

On input: adcircOutFile The name of the file to be created; the file extension is replaced by .nc or .nc4. If a file with the same name exists, it is renamed to: adcircOutFile.ext-YYYYMMDDhhmmss

On output: adcircOutFile The renamed input file ncID The id of the newly created file

This subroutine is called repeatedly to write the 2D field records in the NetCDF file.

Parameters

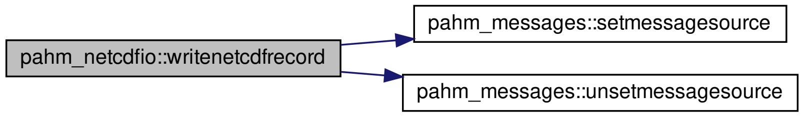
adcircOutFile	The name of the NetCDF file
timeLoc	The time record to write

Definition at line 763 of file netcdfio-nems.F90.

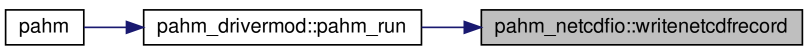
References mytime, pahm_messages::setmessagesource(), pahm_messages::unsetmessagesource(), pahm_global::wpress, pahm_global::wvelx, and pahm_global::wvely.

Referenced by pahm_drivermod::pahm_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3 Variable Documentation

7.8.3.1 mytime `type(timedata_t), save pahm_netcdfio::mytime`

Definition at line 82 of file netcdfio-nems.F90.

Referenced by writenetcdfrecord().

7.9 pahm_sizes Module Reference

Data Types

- interface comparereals
- interface fixnearwholereal

Functions/Subroutines

- integer function comparedoublereals (rVal1, rVal2, eps)

    Compares two double precision numbers.
- integer function comparesinglereals (rVal1, rVal2, eps)

    Compares two single precision numbers.
- real(hp) function fixnearwholedoublereal (rVal, eps)

    Rounds a double precision real number to its nearest whole number.
- real(sp) function fixnearwholesinglereal (rVal, eps)

    Rounds a single precision real number to its nearest whole number.

Variables

- integer, parameter sp = SELECTED_REAL_KIND(6, 37)
- integer, parameter hp = SELECTED_REAL_KIND(15, 307)
- integer, parameter int16 = SELECTED_INT_KIND(38)
- integer, parameter int8 = SELECTED_INT_KIND(18)
- integer, parameter int4 = SELECTED_INT_KIND(9)
- integer, parameter int2 = SELECTED_INT_KIND(4)
- integer, parameter int1 = SELECTED_INT_KIND(2)
- integer, parameter long = INT8
- integer, parameter llong = INT16
- integer, parameter wp = HP
- integer, parameter ip = INT8
- integer, parameter sz = HP
- integer, parameter nbyte = 8
- real(sz), parameter rmissv = -999999.0_SZ
- integer, parameter imissv = -999999
- character(len=1), parameter blank = ' '
- integer, parameter fnamelen = 1024

7.9.1 Function/Subroutine Documentation

7.9.1.1 comparedoublereals()

```
integer function pahm_sizes::comparedoublereals (
        real(hp), intent(in) rVal1,
        real(hp), intent(in) rVal2,
        real(hp), intent(in), optional eps )
```

Compares two double precision numbers.

Allow users to define the value of eps. If not, eps equals to the default machine eps.

Parameters

rVal1	The first value (double precision number) in the comparison
rVal2	The second value (double precision number) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut

```
-1 (if rVal1 < rVal2)
 0 (if rVal1 = rVal2)
+1 (if rVal1 > rVal2)
```

Note

The code was adopted from the D-Flow FM source (...src/precision_basics.F90)

Definition at line 101 of file sizes.F90.

7.9.1.2 comparesinglereals()

```
integer function pahm_sizes::comparesinglereals (
        real(sp), intent(in) rVal1,
        real(sp), intent(in) rVal2,
        real(sp), intent(in), optional eps )
```

Compares two single precision numbers.

Allow users to define the value of eps. If not, eps equals to the default machine eps.

Parameters

rVal1	The first value (single precision number) in the comparison
rVal2	The second value (single precision number) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut

```
-1 (if rVal1 < rVal2)
 0 (if rVal1 = rVal2)
+1 (if rVal1 > rVal2)
```

Note

The code was adopted from the D-Flow FM source (...src/precision_basics.F90)

Definition at line 168 of file sizes.F90.

7.9.1.3 fixnearwholedoublereal()

real(hp) function pahm_sizes::fixnearwholedoublereal (
    real(hp), intent(in) *rVal*,
    real(hp), intent(in), optional *eps*)

Rounds a double precision real number to its nearest whole number.

Rounds a double precision real number to its nearest whole number. If the real number is very close (within a tolerance) to its nearest whole number then it is set equal to its nearest whole number. Allow users to define the value of the tolerance "eps". If not, then eps equals to the default machine eps.

Parameters

rVal	The real number value (double precision) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut : Either **rVal** or its nearest integer **iVar** converted to double

```
rVal (if abs(rVal - iVal) >  eps
iVal (if abs(rVal - iVal) <= eps
```

Definition at line 235 of file sizes.F90.

7.9.1.4 fixnearwholesinglereal()

real(sp) function pahm_sizes::fixnearwholesinglereal (
    real(sp), intent(in) *rVal*,
    real(sp), intent(in), optional *eps*)

Rounds a single precision real number to its nearest whole number.

Rounds a single precision real number to its nearest whole number. If the real number is very close (within a tolerance) to its nearest whole number then it is set equal to its nearest whole number. Allow users to define the value of the tolerance "eps". If not, then eps equals to the default machine eps.

Parameters

rVal	The real number value (single precision) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut : Either **rVal** or its nearest integer **iVar** converted to real

```
rVal (if abs(rVal - iVal) >  eps
iVal (if abs(rVal - iVal) <= eps
```

Definition at line 291 of file sizes.F90.

7.9.2 Variable Documentation

7.9.2.1 blank `character(len=1), parameter pahm_sizes::blank = ' '`

Definition at line 66 of file sizes.F90.

Referenced by utilities::readcontrolfile().

7.9.2.2 fnamelen `integer, parameter pahm_sizes::fnamelen = 1024`

Definition at line 69 of file sizes.F90.

7.9.2.3 hp `integer, parameter pahm_sizes::hp = SELECTED_REAL_KIND(15, 307)`

Definition at line 35 of file sizes.F90.

Referenced by timedateutils::gregtojulday::gregtojuldayisec().

7.9.2.4 imissv `integer, parameter pahm_sizes::imissv = -999999`

Definition at line 64 of file sizes.F90.

Referenced by timedateutils::dayofyear(), pahm_netcdfio::initadcircnetcdfoutfile(), timedateutils::monthdays(), and utilities::readcontrolfile().

7.9.2.5 int1 `integer, parameter pahm_sizes::int1 = SELECTED_INT_KIND( 2)`

Definition at line 42 of file sizes.F90.

7.9.2.6 int16 `integer, parameter pahm_sizes::int16 = SELECTED_INT_KIND(38)`

Definition at line 38 of file sizes.F90.

7.9.2.7 int2 `integer, parameter pahm_sizes::int2 = SELECTED_INT_KIND( 4)`

Definition at line 41 of file sizes.F90.

7.9.2.8 int4 `integer, parameter pahm_sizes::int4 = SELECTED_INT_KIND( 9)`

Definition at line 40 of file sizes.F90.

7.9.2.9 int8 `integer, parameter pahm_sizes::int8 = SELECTED_INT_KIND(18)`

Definition at line 39 of file sizes.F90.

7.9.2.10 ip `integer, parameter pahm_sizes::ip = INT8`

Definition at line 47 of file sizes.F90.

Referenced by csv_utilities::sort_ascending().

7.9.2.11 llong `integer, parameter pahm_sizes::llong = INT16`

Definition at line 44 of file sizes.F90.

7.9.2.12 long `integer, parameter pahm_sizes::long = INT8`

Definition at line 43 of file sizes.F90.

7.9.2.13 nbyte `integer, parameter pahm_sizes::nbyte = 8`

Definition at line 57 of file sizes.F90.

7.9.2.14 rmissv `real(sz), parameter pahm_sizes::rmissv = -999999.0_SZ`

Definition at line 63 of file sizes.F90.

Referenced by timedateutils::dayofyear(), timedateutils::gregtojulday::gregtojuldayisec(), pahm_netcdfio::initadcircnetcdfoutfile(), timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

7.9.2.15 sp `integer, parameter pahm_sizes::sp = SELECTED_REAL_KIND(6, 37)`

Definition at line 34 of file sizes.F90.

7.9.2.16 sz `integer, parameter pahm_sizes::sz = HP`

Definition at line 56 of file sizes.F90.

7.9.2.17 wp `integer, parameter pahm_sizes::wp = HP`

Definition at line 46 of file sizes.F90.

Referenced by csv_module::initialize_csv_file().

7.10 pahm_vortex Module Reference

Functions/Subroutines

- subroutine calcintensitychange (var, times, calcInt, status, order)

  This subroutine calculates the intensity time change of a variable using second order mumerical accuracy and uneven spacing.
- subroutine uvtrans (lat, lon, times, u, v, status, order)

  This subroutine calculates the translational velocity of a moving hurricane using second order mumerical accuracy and uneven spacing.
- subroutine uvtranspoint (lat1, lon1, lat2, lon2, time1, time2, u, v)

  This subroutine calculates the translational velocity of a moving hurricane.
- subroutine newvortex (pinf, p0, lat, lon, vm)

  Create a new Vortex object.
- subroutine newvortexfull (pinf, p0, lat, lon, vm)

  A new vortex is created for the full gradient wind balance.
- subroutine setvortex (pinf, p0, lat, lon)

  Set basic parameter for a new Vortex object.
- subroutine setrmaxes (rMaxW)
- subroutine getrmaxes (rMaxW)

- subroutine calcrmaxes ()

  Calculate the radius of maximum winds for all storm quadrants.
- subroutine calcrmaxesfull ()

  Calculate the radius of maximum winds for all storm quadrants. Solving the full gradient wind equation without the assumption of cyclostrohpic balance.
- subroutine fitrmaxes ()

  Calculates the coefficients that fit the given radius of maximum winds for all storm quadrants.
- subroutine fitrmaxes4 ()
- subroutine setvmaxesbl (vMaxW)
- subroutine getvmaxesbl (vMaxW)
- subroutine setusevmaxesbl (u)
- subroutine setshapeparameter (param)
- real(sz) function getshapeparameter ()
- real(sz) function, dimension(4) getshapeparameters ()
- real(sz) function, dimension(4) getphifactors ()
- subroutine setisotachradii (ir)
- subroutine setisotachwindspeeds (vrQ)
- subroutine setusequadrantvr (u)
- logical function getusequadrantvr ()
- real(sz) function spinterp (angle, dist, opt)

  Spatial Interpolation function based on angle and r.
- real(sz) function interpr (quadVal, quadSel, quadDis)
- real(sz) function rmw (angle)

  Calculate the radius of maximum winds.
- subroutine uvp (lat, lon, uTrans, vTrans, u, v, p)

  Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.
- subroutine uvpr (iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, uTrans, vTrans, geof, u, v, p)

  Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.
- real(sz) function fang (r, rmx)

  Compute a wind angle to parameterize frictional inflow across isobars.
- subroutine rotate (x, y, angle, whichWay, xr, yr)

  Rotate a 2D vector (x, y) by an angle.
- real(sz) function getlatestrmax ()
- real(sz) function getlatestangle ()
- real(sz) function vhwithcorifull (testRMax)

  External function f(x) = 0 for which a root is sought using Brent's root-finding method.
- real(sz) function vhwithcori (testRMax)

  External function f(x) = 0 for which a root is sought using Brent's root-finding method.
- real(sz) function vhnocori (testRMax)
- real(sz) function findroot (func, x1, x2, dx, a, b)

  Use brute-force marching to find a root the interval [x1,x2].

Variables

- integer, parameter nquads = 4
- integer, parameter npoints = NQUADS + 2
- real(sz), dimension(npoints) rmaxes
- real(sz), dimension(npoints, 4) rmaxes4
- real(sz) pn
- real(sz) pc
- real(sz) clat
- real(sz) clon
- real(sz) vmax
- real(sz) b
- real(sz) corio
- real(sz) vr
- real(sz) phi
- real(sz), dimension(npoints) phis
- real(sz), dimension(npoints, 4) phis4
- real(sz), dimension(npoints) bs
- real(sz), dimension(npoints, 4) bs4
- real(sz), dimension(npoints) vmbl
- real(sz), dimension(npoints, 4) vmbl4
- integer, dimension(npoints, 4) quadflag4
- real(sz), dimension(npoints, 4) quadir4
- real(sz), dimension(nquads) vrquadrant
- real(sz), dimension(nquads) radius
- integer quad
- real(sz) latestrmax
- real(sz) latestangle
- logical usequadrantvr
- logical usevmaxesbl

7.10.1 Function/Subroutine Documentation

7.10.1.1 calcintensitychange()

```
subroutine pahm_vortex::calcintensitychange (
    real(sz), dimension(:), intent(in) var,
    real(sz), dimension(:), intent(in) times,
    real(sz), dimension(:), intent(out) calcInt,
    integer, intent(out) status,
    integer, intent(in), optional order )
```

This subroutine calculates the intensity time change of a variable using second order mumerical accuracy and uneven spacing.

On input: var The input variable (vector) times Time values (vector) at the center locations order The accuracy order required for the calculations (1, 2) <= 1: first order approximation for finite differences >= 2: second order approximation for finite differences

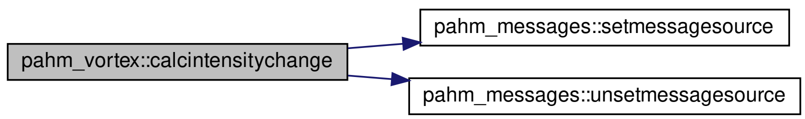
On output: calcInt the calculated intensity change (df/dt) status error status (0 means no error)

Definition at line 79 of file vortex.F90.

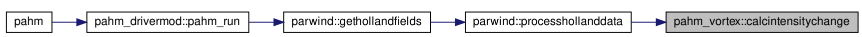
References pahm_global::deg2rad, pahm_messages::error, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by parwind::processhollanddata().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.2 calcrmaxes() `subroutine pahm_vortex::calcrmaxes`

Calculate the radius of maximum winds for all storm quadrants.

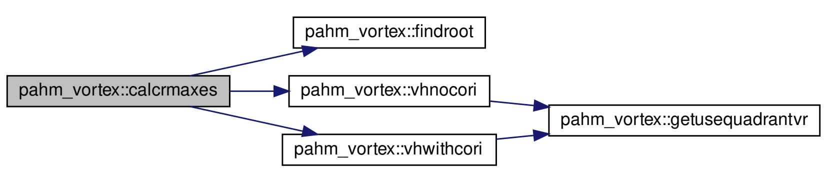
On input: none

On output: rMax radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity

Definition at line 666 of file vortex.F90.

References b, bs, findroot(), nquads, quad, radius, rmaxes, vhnocori(), vhwithcori(), vmax, and vmbl.

Here is the call graph for this function:



7.10.1.3 calcrmaxesfull() `subroutine pahm_vortex::calcrmaxesfull`

Calculate the radius of maximum winds for all storm quadrants. Solving the full gradient wind equation without the assumption of cyclostrohpic balance.

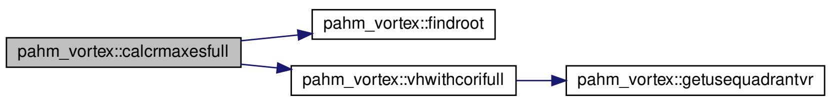
On input: none

On output: rMax radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity

Definition at line 743 of file vortex.F90.

References b, bs, corio, findroot(), pahm_global::kt2ms, pahm_global::mb2pa, pahm_global::nm2m, nquads, pc, phi, phis, pn, quad, radius, pahm_global::rhoair, rmaxes, vhwithcorifull(), vmax, and vmbl.

Here is the call graph for this function:



7.10.1.4 fang() `real(sz) function pahm_vortex::fang (`

```
real(sz), intent(in) r,
real(sz), intent(in) rmx )
```

Compute a wind angle to parameterize frictional inflow across isobars.

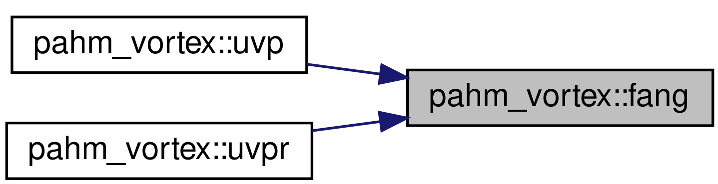
On input: r distance from center of storm rmx radius of maximum winds

On output: FAng frictional inflow angle (degrees)

Definition at line 1610 of file vortex.F90.

Referenced by uvp(), and uvpr().

Here is the caller graph for this function:



7.10.1.5 findroot()

```
real(sz) function pahm_vortex::findroot (
        real(sz), external func,
        real(sz), intent(in) x1,
        real(sz), intent(in) x2,
        real(sz), intent(in) dx,
        real(sz), intent(out) a,
        real(sz), intent(out) b )
```

Use brute-force marching to find a root the interval [x1,x2].

On input: func function f(x)=0 for which root is sough x1 left side of interval x2 right side of interval dx x increment for march

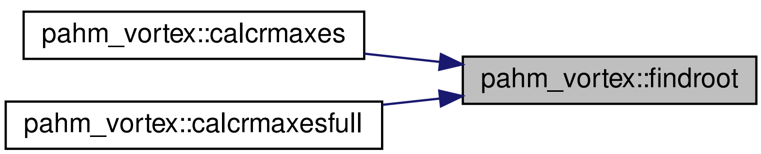
On output: a left side of interval that brackets the root b right side of interval that brackets the root FindRoot root returned

Definition at line 1845 of file vortex.F90.

References b.

Referenced by calcrmaxes(), and calcrmaxesfull().

Here is the caller graph for this function:



7.10.1.6 fitrmaxes()

```
subroutine pahm_vortex::fitrmaxes
```

Calculates the coefficients that fit the given radius of maximum winds for all storm quadrants.

On input: rMax in all 4 quadrants plus 2 extra values to tie down circular periodicity

On output: rMax radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity

Definition at line 880 of file vortex.F90.

References rmaxes.

7.10.1.7 fitrmaxes4() `subroutine pahm_vortex::fitrmaxes4`

Definition at line 895 of file vortex.F90.

References bs4, phis4, quadflag4, quadir4, rmaxes4, and vmbl4.

7.10.1.8 getlatestangle() `real(sz) function pahm_vortex::getlatestangle`

Definition at line 1691 of file vortex.F90.

References latestangle.

7.10.1.9 getlatestrmax() `real(sz) function pahm_vortex::getlatestrmax`

Definition at line 1678 of file vortex.F90.

References latestrmax.

7.10.1.10 getphifactors() `real(sz) function, dimension(4) pahm_vortex::getphifactors`

Definition at line 1027 of file vortex.F90.

References phis.

7.10.1.11 getrmaxes() `subroutine pahm_vortex::getrmaxes (`
`real(sz), dimension(4), intent(out)` *`rMaxW`* `)`

Definition at line 636 of file vortex.F90.

References rmaxes.

7.10.1.12 getshapeparameter() `real(sz) function pahm_vortex::getshapeparameter`

Definition at line 991 of file vortex.F90.

References b.

7.10.1.13 getshapeparameters() `real(sz) function, dimension(4) pahm_vortex::getshapeparameters`

Definition at line 1006 of file vortex.F90.

References bs.

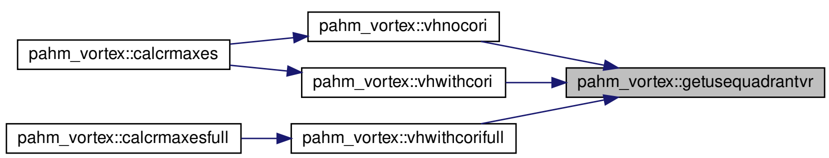
7.10.1.14 getusequadrantvr() `logical function pahm_vortex::getusequadrantvr`

Definition at line 1093 of file vortex.F90.

References usequadrantvr.

Referenced by vhnocori(), vhwithcori(), and vhwithcorifull().

Here is the caller graph for this function:



7.10.1.15 getvmaxesbl() `subroutine pahm_vortex::getvmaxesbl ( real(sz), dimension(4), intent(out) *vMaxW* )`

Definition at line 944 of file vortex.F90.

References vmbl.

7.10.1.16 interpr()

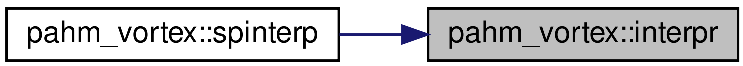
```
real(sz) function pahm_vortex::interpr (
        real(sz), dimension(npoints, 4), intent(in) quadVal,
        integer, intent(in) quadSel,
        real(sz), intent(in) quadDis )
```

Definition at line 1179 of file vortex.F90.

References quadflag4, and quadir4.

Referenced by spinterp().

Here is the caller graph for this function:



7.10.1.17 newvortex()

```
subroutine pahm_vortex::newvortex (
        real(sz), intent(in) pinf,
        real(sz), intent(in) p0,
        real(sz), intent(in) lat,
        real(sz), intent(in) lon,
        real(sz), intent(in) vm )
```

Create a new Vortex object.

On input: pn Ambient surface pressure (mb) pc Surface pressure at center of storm (mb) cLat Latitude of storm center (degrees north) cLon Longitude of storm center (degrees east) vMax Max sustained wind velocity in storm (knots)

On output: A new vortex is created with essential parameters calculated.

Definition at line 505 of file vortex.F90.

References b, bs, clat, clon, corio, pahm_global::deg2rad, pahm_global::kt2ms, pahm_global::mb2pa, pahm_global::omega, pc, pn, pahm_global::rhoair, vmax, and vmbl.

7.10.1.18 newvortexfull()

```
subroutine pahm_vortex::newvortexfull (
        real(sz), intent(in) pinf,
        real(sz), intent(in) p0,
        real(sz), intent(in) lat,
        real(sz), intent(in) lon,
        real(sz), intent(in) vm )
```

A new vortex is created for the full gradient wind balance.

Definition at line 543 of file vortex.F90.

References b, bs, clat, clon, corio, pahm_global::deg2rad, pahm_global::kt2ms, pahm_global::mb2pa, pahm_global::omega, pc, phi, phis, pn, pahm_global::rhoair, vmax, and vmbl.

7.10.1.19 rmw()

```
real(sz) function pahm_vortex::rmw (
        real(sz), intent(in) angle )
```

Calculate the radius of maximum winds.

On input: angle Azimuthal angle (degrees)

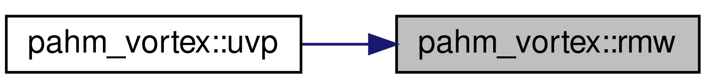
On output: Rmw Radius of maximum winds (meters) from curve fit I DO NOT BELIEVE IT IS IN METERS rjw

Definition at line 1254 of file vortex.F90.

References rmaxes.

Referenced by uvp().

Here is the caller graph for this function:



7.10.1.20 rotate()

```
subroutine pahm_vortex::rotate (
        real(sz), intent(in) x,
        real(sz), intent(in) y,
        real(sz), intent(in) angle,
        real(sz), intent(in) whichWay,
        real(sz), intent(out) xr,
        real(sz), intent(out) yr )
```

Rotate a 2D vector (x, y) by an angle.

On input: x x component of vector y y component of vector angle angle to rotate vector (degrees) whichWay direction of rotation:

- = clockwise, + = counter-clockwise

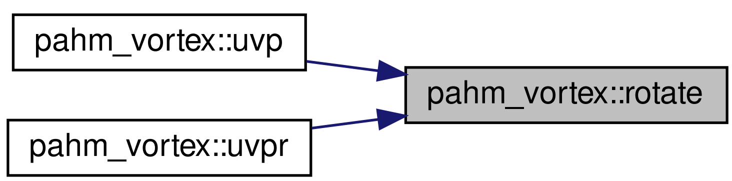
On output: xr x component of rotated vector yr y component of rotated vector

Definition at line 1648 of file vortex.F90.

References pahm_global::deg2rad.

Referenced by uvp(), and uvpr().

Here is the caller graph for this function:



7.10.1.21 setisotachradii()

```
subroutine pahm_vortex::setisotachradii (
        real(sz), dimension(4), intent(in) ir )
```

Definition at line 1048 of file vortex.F90.

References radius.

7.10.1.22 setisotachwindspeeds()

```
subroutine pahm_vortex::setisotachwindspeeds (
        real(sz), dimension(4), intent(in) vrQ )
```

Definition at line 1063 of file vortex.F90.

References vrquadrant.

7.10.1.23 setrmaxes()

```
subroutine pahm_vortex::setrmaxes (
        real(sz), dimension(4), intent(in) rMaxW )
```

Definition at line 618 of file vortex.F90.

References rmaxes.

7.10.1.24 setshapeparameter()

```
subroutine pahm_vortex::setshapeparameter (
        real(sz) param )
```

Definition at line 976 of file vortex.F90.

References b.

7.10.1.25 setusequadrantvr()

```
subroutine pahm_vortex::setusequadrantvr (
        logical, intent(in) u )
```

Definition at line 1078 of file vortex.F90.

References usequadrantvr.

7.10.1.26 setusevmaxesbl()

```
subroutine pahm_vortex::setusevmaxesbl (
        logical, intent(in) u )
```

Definition at line 963 of file vortex.F90.

References usevmaxesbl.

7.10.1.27 setvmaxesbl()

```
subroutine pahm_vortex::setvmaxesbl (
        real(sz), dimension(4), intent(in) vMaxW )
```

Definition at line 925 of file vortex.F90.

References vmbl.

7.10.1.28 setvortex()

```
subroutine pahm_vortex::setvortex (
        real(sz), intent(in) pinf,
        real(sz), intent(in) p0,
        real(sz), intent(in) lat,
        real(sz), intent(in) lon )
```

Set basic parameter for a new Vortex object.

On input: pinf hurricane Ambient pressure p0 hurricane central pressure lat Latitude of storm center (degrees north) lon Longitude of storm center (degrees east)

On output: Aim is to define pn, pc, and corio

Definition at line 591 of file vortex.F90.

References clat, clon, corio, pahm_global::deg2rad, pahm_global::omega, pc, and pn.

7.10.1.29 spinterp()

```
real(sz) function pahm_vortex::spinterp (
        real(sz), intent(in) angle,
        real(sz), intent(in) dist,
        integer, intent(in) opt )
```

Spatial Interpolation function based on angle and r.

On input: angle Azimuthal angle (degrees) r Distnace to storm Center (nm)

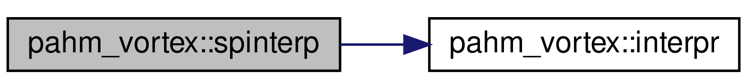
On output: interpolated value for rMax/vMax/B

INTEGER validIsot is used as a marker to indicate how many isotachs are available in a certain quadrant SELECT CASE(validIsot) CASE(1): 1 situation CASE(2): 3 situations CASE(3): 4 situations CASE(4): 5 situations

Definition at line 1124 of file vortex.F90.

References bs4, interpr(), rmaxes4, and vmbl4.

Here is the call graph for this function:



7.10.1.30 uvp()

```
subroutine pahm_vortex::uvp (
        real(sz), intent(in) lat,
        real(sz), intent(in) lon,
        real(sz), intent(in) uTrans,
        real(sz), intent(in) vTrans,
        real(sz), intent(out) u,
        real(sz), intent(out) v,
        real(sz), intent(out) p )
```

Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.

On input: lat Latitude of nodal point (degrees north) lon Longitude of nodal point (degrees east) uTrans x component of translational velocity (kts) vTrans y component of translational velocity (kts)

On output: u x component of wind velocity at nodal point (m/s) v y component of wind velocity at nodal point (m/s) p Surface pressure at nodal point (Pa)

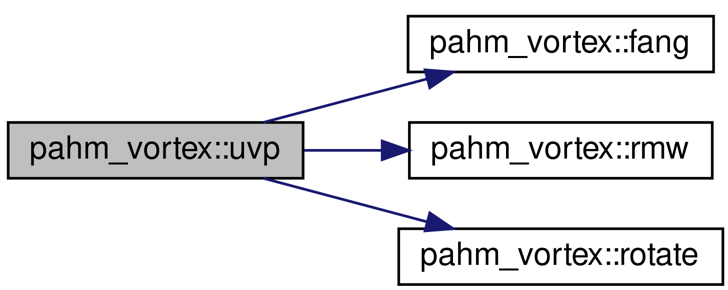
Internal parameters: dampRadii How far out (# of rMax radii) to begin damping out the translational velocity

Note: Subroutine directly accesses global class instance variables

Definition at line 1324 of file vortex.F90.

References b, clat, clon, corio, pahm_global::deg2rad, fang(), pahm_global::kt2ms, latestangle, latestrmax, pahm_global::m2nm, pahm_global::mb2pa, pahm_global::nm2m, pahm_global::one2ten, pc, pn, pahm_global::rad2deg, pahm_global::rearth, rmw(), rotate(), vmax, and pahm_global::windreduction.

Here is the call graph for this function:



7.10.1.31 uvpr()

```
subroutine pahm_vortex::uvpr (
        real(sz), intent(in) iDist,
        real(sz), intent(in) iAngle,
        real(sz), intent(in) iRmx,
        real(sz), intent(in) iRmxTrue,
        real(sz), intent(in) iB,
        real(sz), intent(in) iVm,
        real(sz), intent(in) iPhi,
        real(sz), intent(in) uTrans,
        real(sz), intent(in) vTrans,
        integer, intent(in) geof,
        real(sz), intent(out) u,
        real(sz), intent(out) v,
        real(sz), intent(out) p )
```

Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.

On input: pinf hurricane Ambient pressure p0 hurricane central pressure iDist dist to hurricane center in nautical mile iRmx Rmw iAngle Azimuth Angle iB Holland B parameter iVm vortex maximum velocity at upper boundary iPhi vortex correction factor uTrans x component of translational velocity (kts) vTrans y component of translational velocity (kts)

On output: u x component of wind velocity at nodal point (m/s) v y component of wind velocity at nodal point (m/s) p Surface pressure at nodal point (Pa)

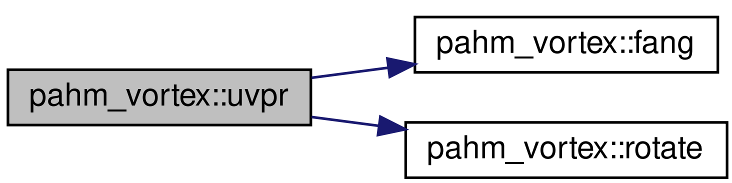
Internal parameters: dampRadii How far out (# of rMax radii) to begin damping out the translational velocity

Note: Subroutine directly accesses global class instance variables

Definition at line 1482 of file vortex.F90.

References b, clat, corio, pahm_global::deg2rad, fang(), pahm_global::kt2ms, pahm_global::mb2pa, pahm_global::nm2m, pahm_global::one2ten, pc, phi, pn, rotate(), vmax, and pahm_global::windreduction.

Here is the call graph for this function:



7.10.1.32 uvtrans()

```
subroutine pahm_vortex::uvtrans (
    real(sz), dimension(:), intent(in) lat,
    real(sz), dimension(:), intent(in) lon,
    real(sz), dimension(:), intent(in) times,
    real(sz), dimension(:), intent(out) u,
    real(sz), dimension(:), intent(out) v,
    integer, intent(out) status,
    integer, intent(in), optional order )
```

This subroutine calculates the translational velocity of a moving hurricane using second order mumerical accuracy and uneven spacing.

On input: lat Latitude values (vector) of the center (degrees north) lon Longitude values (vector) of the center (degrees east) times Time values (vector) at the center locations (seconds) order The accuracy order required for the calculations (1, 2) <= 1: first order approximation for finite differences >= 2: second order approximation for finite differences

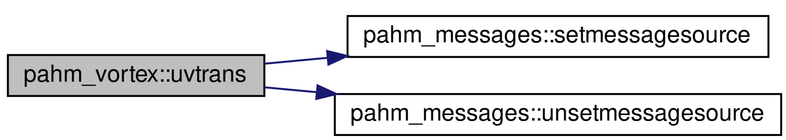
On output: u x component of the translational velocities (m/s) v y component of the translational velocities (m/s) status error status (0 means no error)

Definition at line 241 of file vortex.F90.

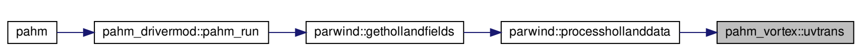
References pahm_global::deg2rad, pahm_messages::error, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by parwind::processhollanddata().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.33 uvtranspoint()

```
subroutine pahm_vortex::uvtranspoint (
        real(sz), intent(in) lat1,
        real(sz), intent(in) lon1,
        real(sz), intent(in) lat2,
        real(sz), intent(in) lon2,
        real(sz), intent(in) time1,
        real(sz), intent(in) time2,
        real(sz), intent(out) u,
        real(sz), intent(out) v )
```

This subroutine calculates the translational velocity of a moving hurricane.

On input: lat1 Previous latitude of center (degrees north) lon1 Previous longitude of center (degrees east) lat2 Current latitude of center (degrees north) lon2 Current longitude of center (degrees east) time1 Previous time (seconds) time1 Current time (seconds)

On output: u x component of translational velocity (m/s) v y component of translational velocity (m/s)

Definition at line 457 of file vortex.F90.

References pahm_global::deg2rad.

7.10.1.34 vhnocori()

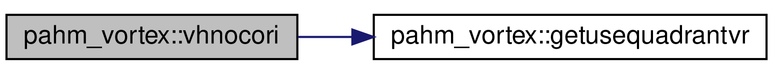
```
real(sz) function pahm_vortex::vhnocori (
        real(sz), intent(in) testRMax )
```

Definition at line 1804 of file vortex.F90.

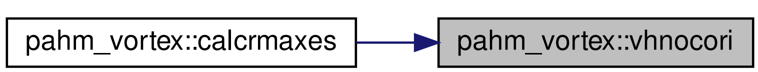
References b, getusequadrantvr(), pahm_global::kt2ms, pahm_global::ms2kt, quad, radius, vmax, vr, and vrquadrant.

Referenced by calcrmaxes().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.35 vhwithcori() `real(sz) function pahm_vortex::vhwithcori ( real(sz), intent(in) testRMax )`

External function f(x) = 0 for which a root is sought using Brent's root-finding method.

On input: x iterative values which converge to root

On output: func f(x)

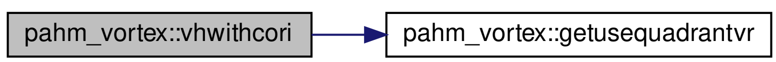
Internal parameters: vortex instance variables via accessor functions

Definition at line 1768 of file vortex.F90.

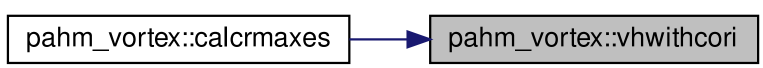
References b, corio, getusequadrantvr(), pahm_global::kt2ms, pahm_global::ms2kt, pahm_global::nm2m, quad, radius, vmax, vr, and vrquadrant.

Referenced by calcrmaxes().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.36 vhwithcorifull() `real(sz) function pahm_vortex::vhwithcorifull ( real(sz), intent(in) testRMax )`

External function f(x) = 0 for which a root is sought using Brent's root-finding method.

On input: x iterative values which converge to root

On output: func f(x)

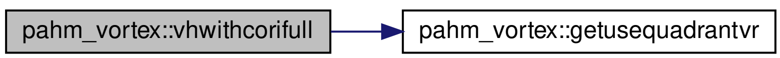
Internal parameters: vortex instance variables via accessor functions

Definition at line 1718 of file vortex.F90.

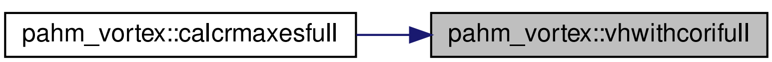
References b, corio, getusequadrantvr(), pahm_global::kt2ms, pahm_global::ms2kt, pahm_global::nm2m, phi, quad, radius, vmax, vr, and vrquadrant.

Referenced by calcrmaxesfull().

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.2 Variable Documentation

7.10.2.1 b `real(sz) pahm_vortex::b`

Definition at line 35 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), findroot(), getshapeparameter(), newvortex(), newvortexfull(), setshapeparameter(), uvp(), uvpr(), vhnocori(), vhwithcori(), and vhwithcorifull().

7.10.2.2 bs `real(sz), dimension(npoints) pahm_vortex::bs`

Definition at line 42 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), getshapeparameters(), newvortex(), and newvortexfull().

7.10.2.3 bs4 `real(sz), dimension(npoints, 4) pahm_vortex::bs4`

Definition at line 43 of file vortex.F90.

Referenced by fitrmaxes4(), and spinterp().

7.10.2.4 clat `real(sz) pahm_vortex::clat`

Definition at line 31 of file vortex.F90.

Referenced by newvortex(), newvortexfull(), setvortex(), uvp(), and uvpr().

7.10.2.5 clon `real(sz) pahm_vortex::clon`

Definition at line 32 of file vortex.F90.

Referenced by newvortex(), newvortexfull(), setvortex(), and uvp().

7.10.2.6 corio `real(sz) pahm_vortex::corio`

Definition at line 36 of file vortex.F90.

Referenced by calcrmaxesfull(), newvortex(), newvortexfull(), setvortex(), uvp(), uvpr(), vhwithcori(), and vhwithcorifull().

7.10.2.7 latestangle `real(sz) pahm_vortex::latestangle`

Definition at line 54 of file vortex.F90.

Referenced by getlatestangle(), and uvp().

7.10.2.8 latestrmax `real(sz) pahm_vortex::latestrmax`

Definition at line 53 of file vortex.F90.

Referenced by getlatestrmax(), and uvp().

7.10.2.9 npoints `integer, parameter pahm_vortex::npoints = NQUADS + 2`

Definition at line 25 of file vortex.F90.

7.10.2.10 nquads `integer, parameter pahm_vortex::nquads = 4`

Definition at line 24 of file vortex.F90.

Referenced by calcrmaxes(), and calcrmaxesfull().

7.10.2.11 pc `real(sz) pahm_vortex::pc`

Definition at line 30 of file vortex.F90.

Referenced by calcrmaxesfull(), newvortex(), newvortexfull(), setvortex(), uvp(), and uvpr().

7.10.2.12 phi `real(sz) pahm_vortex::phi`

Definition at line 38 of file vortex.F90.

Referenced by calcrmaxesfull(), newvortexfull(), uvpr(), and vhwithcorifull().

7.10.2.13 phis `real(sz), dimension(npoints) pahm_vortex::phis`

Definition at line 39 of file vortex.F90.

Referenced by calcrmaxesfull(), getphifactors(), and newvortexfull().

7.10.2.14 phis4 `real(sz), dimension(npoints, 4) pahm_vortex::phis4`

Definition at line 40 of file vortex.F90.

Referenced by fitrmaxes4().

7.10.2.15 pn `real(sz) pahm_vortex::pn`

Definition at line 29 of file vortex.F90.

Referenced by calcrmaxesfull(), newvortex(), newvortexfull(), setvortex(), uvp(), and uvpr().

7.10.2.16 quad `integer pahm_vortex::quad`

Definition at line 51 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), vhnocori(), vhwithcori(), and vhwithcorifull().

7.10.2.17 quadflag4 `integer, dimension(npoints, 4) pahm_vortex::quadflag4`

Definition at line 46 of file vortex.F90.

Referenced by fitrmaxes4(), and interpr().

7.10.2.18 quadir4 `real(sz), dimension(npoints, 4) pahm_vortex::quadir4`

Definition at line 47 of file vortex.F90.

Referenced by fitrmaxes4(), and interpr().

7.10.2.19 radius `real(sz), dimension(nquads) pahm_vortex::radius`

Definition at line 49 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), setisotachradii(), vhnocori(), vhwithcori(), and vhwithcorifull().

7.10.2.20 rmaxes `real(sz), dimension(npoints) pahm_vortex::rmaxes`

Definition at line 26 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), fitrmaxes(), getrmaxes(), rmw(), and setrmaxes().

7.10.2.21 rmaxes4 `real(sz), dimension(npoints, 4) pahm_vortex::rmaxes4`

Definition at line 27 of file vortex.F90.

Referenced by fitrmaxes4(), and spinterp().

7.10.2.22 usequadrantvr `logical pahm_vortex::usequadrantvr`

Definition at line 55 of file vortex.F90.

Referenced by getusequadrantvr(), and setusequadrantvr().

7.10.2.23 usevmaxesbl `logical pahm_vortex::usevmaxesbl`

Definition at line 56 of file vortex.F90.

Referenced by setusevmaxesbl().

7.10.2.24 vmax `real(sz) pahm_vortex::vmax`

Definition at line 33 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), newvortex(), newvortexfull(), uvp(), uvpr(), vhnocori(), vhwithcori(), and vhwithcorifull().

7.10.2.25 vmbl `real(sz), dimension(npoints) pahm_vortex::vmbl`

Definition at line 44 of file vortex.F90.

Referenced by calcrmaxes(), calcrmaxesfull(), getvmaxesbl(), newvortex(), newvortexfull(), and setvmaxesbl().

7.10.2.26 vmbl4 `real(sz), dimension(npoints, 4) pahm_vortex::vmbl4`

Definition at line 45 of file vortex.F90.

Referenced by fitrmaxes4(), and spinterp().

7.10.2.27 vr `real(sz) pahm_vortex::vr`

Definition at line 37 of file vortex.F90.

Referenced by vhnocori(), vhwithcori(), and vhwithcorifull().

7.10.2.28 vrquadrant `real(sz), dimension(nquads) pahm_vortex::vrquadrant`

Definition at line 48 of file vortex.F90.

Referenced by setisotachwindspeeds(), vhnocori(), vhwithcori(), and vhwithcorifull().

7.11 parwind Module Reference

Data Types

- type besttrackdata_t
- type hollanddata_t

Functions/Subroutines

- subroutine readbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine readcsvbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine processhollanddata (idTrFile, strOut, status)

  Subroutine to support the Holland model (GetHolland). Gets the next line from the file, skipping lines that are time repeats.
- subroutine gethollandfields ()

  Calculate wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.
- subroutine writebesttrackdata (inpFile, btrStruc, suffix)

  Writes the best track data (adjusted or not) to the "adjusted" best track output file.
- subroutine allocbtrstruct (str, nRec)

  Subroutine to allocate memory for a best track structure.
- subroutine deallocbtrstruct (str)

  Subroutine to deallocate the memory allocated for a best track structure.
- subroutine allochollstruct (str, nRec)

  Subroutine to allocate memory for a holland structure.
- subroutine deallochollstruct (str)

  Subroutine to deallocate memory of an allocated holland structure.
- subroutine gethollandfields (timeIDX)

  Calculates wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.

Variables

- real(sz) windreftime
- type(besttrackdata_t), dimension(:), allocatable, target besttrackdata

7.11.1 Detailed Description

Author

PanagiotisVelissariou panagiotis.velissariou@noaa.gov

7.11.2 Function/Subroutine Documentation

7.11.2.1 allocbtrstruct()

```
subroutine parwind::allocbtrstruct (
        type(besttrackdata_t) str,
        integer, intent(in) nRec )
```

Subroutine to allocate memory for a best track structure.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

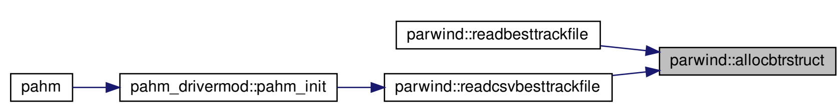
Parameters

str	The best track structure of type BestTrackData_T
nRec	The number of records in the structure

Definition at line 1330 of file parwind-orig.F90.

Referenced by readbesttrackfile(), and readcsvbesttrackfile().

Here is the caller graph for this function:



7.11.2.2 allochollstruct()

```
subroutine parwind::allochollstruct (
        type(hollanddata_t) str,
        integer, intent(in) nRec )
```

Subroutine to allocate memory for a holland structure.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

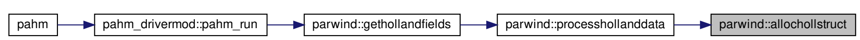
Parameters

str	The holland structure of type HollandData_T
nRec	The number of records in the structure

Definition at line 1449 of file parwind-orig.F90.

Referenced by processhollanddata().

Here is the caller graph for this function:



7.11.2.3 deallocbtrstruct()

```
subroutine parwind::deallocbtrstruct (
        type(besttrackdata_t) str )
```

Subroutine to deallocate the memory allocated for a best track structure.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Parameters

str	The best track structure of type BestTrackData_T

Definition at line 1390 of file parwind-orig.F90.

7.11.2.4 deallochollstruct() `subroutine parwind::deallochollstruct ( type(hollanddata_t), intent(out) str )`

Subroutine to deallocate memory of an allocated holland structure.

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

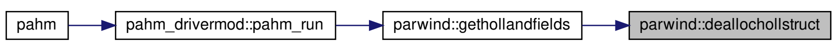
Parameters

str	The holland structure of type HollandData_T

Definition at line 1504 of file parwind-orig.F90.

Referenced by gethollandfields().

Here is the caller graph for this function:



7.11.2.5 gethollandfields() [1/2] `subroutine parwind::gethollandfields`

Calculate wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.

The format statement takes into account whether the track data is hindcast/nowcast (BEST) or forecast (OFCL).

The first line in the file MUST be a hindcast, since the central pressure and the rmw are carried forward from hindcasts into forecasts. So there needs to be at least one hindcast to carry the data forward.

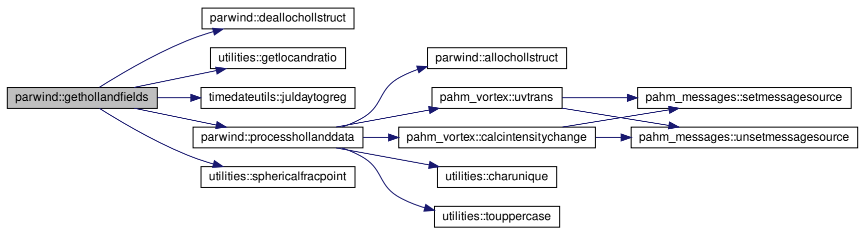
Assumes geographical coordinates.

Definition at line 897 of file parwind-orig.F90.

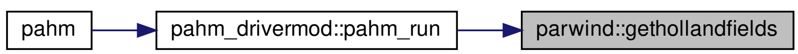
References pahm_global::backgroundatmpress, pahm_global::basee, pahm_global::besttrackfilename, pahm_global::bladjustfac, deallochollstruct(), pahm_global::deg2rad, utilities::getlocandratio(), pahm_global::gravity, pahm_mesh::ismeshok, timedateutils::juldaytogreg(), pahm_global::mb2kpa, pahm_global::mb2pa, pahm_global::mdbegsimtime, pahm_global::mdendsimtime, pahm_global::mdoutdt, pahm_global::nbtrfiles, pahm_global::noutdt, pahm_mesh::np, pahm_global::omega, pahm_global::one2ten, processhollanddata(), pahm_global::rad2deg, pahm_global::rhoair, pahm_global::rhowater, pahm_mesh::sfea, pahm_mesh::slam, utilities::sphericalfracpoint(), pahm_global::times, pahm_global::wpress, pahm_global::wvelx, pahm_global::wvely, pahm_mesh::xcslam, and pahm_mesh::ycsfea.

Referenced by pahm_drivermod::pahm_run().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.2.6 gethollandfields() [2/2]

`subroutine parwind::gethollandfields ( integer, intent(in) timeIDX )`

Calculates wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.

Calculate wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.

The format statement takes into account whether the track data is hindcast/nowcast (BEST) or forecast (OFCL).

The first line in the file MUST be a hindcast, since the central pressure and the rmw are carried forward from hindcasts into forecasts. So there needs to be at least one hindcast to carry the data forward.

Assumes geographical coordinates.

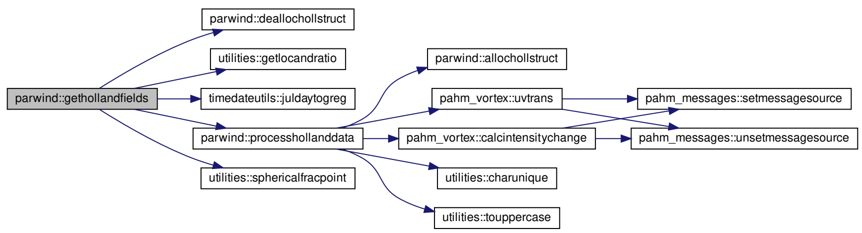
Parameters

timeIDX	The time location to generate the fields for

Definition at line 947 of file parwind.F90.

References pahm_global::backgroundatmpress, pahm_global::basee, pahm_global::besttrackfilename, pahm_global::bladjustfac, deallochollstruct(), pahm_global::deg2rad, utilities::getlocandratio(), pahm_global::gravity, pahm_mesh::ismeshok, timedateutils::juldaytogreg(), pahm_global::lun_btrk, pahm_global::lun_btrk1, pahm_global::mb2kpa, pahm_global::mb2pa, pahm_global::mdbegsimtime, pahm_global::mdendsimtime, pahm_global::mdoutdt, pahm_global::nbtrfiles, pahm_global::noutdt, pahm_mesh::np, pahm_global::omega, pahm_global::one2ten, processhollanddata(), pahm_global::rad2deg, pahm_global::rhoair, pahm_global::rhowater, pahm_mesh::sfea, pahm_mesh::slam, utilities::sphericalfracpoint(), pahm_global::times, pahm_global::wpress, pahm_global::wvelx, pahm_global::wvely, pahm_mesh::xcslam, and pahm_mesh::ycsfea.

Here is the call graph for this function:



7.11.2.7 processhollanddata()

```
subroutine parwind::processhollanddata (
        integer, intent(in) idTrFile,
        type(hollanddata_t), intent(out) strOut,
        integer, intent(out) status )
```

Subroutine to support the Holland model (GetHolland). Gets the next line from the file, skipping lines that are time repeats.

Subroutine to support the Holland model (GetHolland).

- Does conversions to the proper units.
- Uses old values of central pressure and rmw if the line is a forecast, since forecasts do not have that data in them.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Subroutine to support the Holland model (GetHolland). Gets the next line from the file, skipping lines that are time repeats.

- Does conversions to the proper units.
- Uses old values of central pressure and rmw if the line is a forecast, since forecasts do not have that data in them.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Parameters

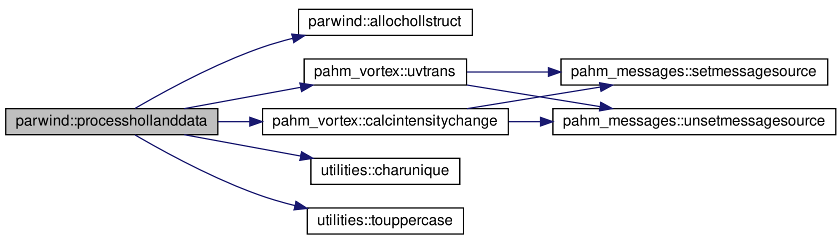
idTrFile	The ID of the input track file (1, 2, ...)
strOut	The HollandData_T structure that stores all Holland model generated data (output)
status	Error status, 0 = no error (output)

Definition at line 678 of file parwind-orig.F90.

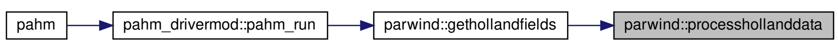
References allochollstruct(), besttrackdata, pahm_vortex::calcintensitychange(), utilities::charunique(), pahm_global::kt2ms, pahm_global::nbtrfiles, pahm_global::nm2m, utilities::touppercase(), and pahm_vortex::uvtrans().

Referenced by gethollandfields().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.2.8 readbesttrackfile() `subroutine parwind::readbesttrackfile`

Subroutine to read all a-deck/b-deck best track files (ATCF format).

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Subroutine to read all a-deck/b-deck best track files (ATCF format).

- a-deck: guidance information
- b-deck: best track information
- Skips lines that are time repeats. ???PV check
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

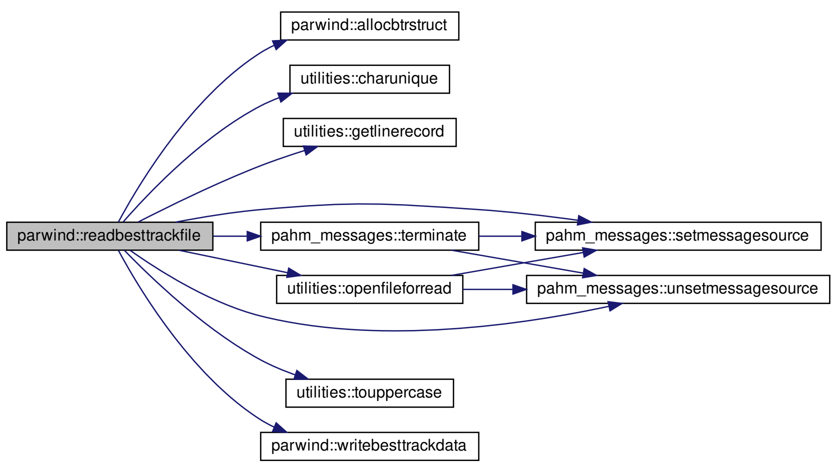
It uses fortran format statements (old approach) to read the ATCF formatted track files as follows:

- a-deck: guidance information
- b-deck: best track information
- Skips lines that are time repeats.
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 165 of file parwind-orig.F90.

References allocbtrstruct(), besttrackdata, pahm_global::besttrackfilename, utilities::charunique(), pahm_messages::error, utilities::getlinerecord(), pahm_messages::info, pahm_global::lun_btrk, pahm_global::lun_btrk1, pahm_global::nbtrfiles, utilities::openfileforread(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_messages::terminate(), utilities::touppercase(), pahm_messages::unsetmessagesource(), and writebesttrackdata().

Here is the call graph for this function:



7.11.2.9 readcsvbesttrackfile() `subroutine parwind::readcsvbesttrackfile ( )`

Subroutine to read all a-deck/b-deck best track files (ATCF format).

It uses PaHM's CSV functionality (preferred approach) to read the ATCF formatted track files as follows:

- a-deck: guidance information
- b-deck: best track information

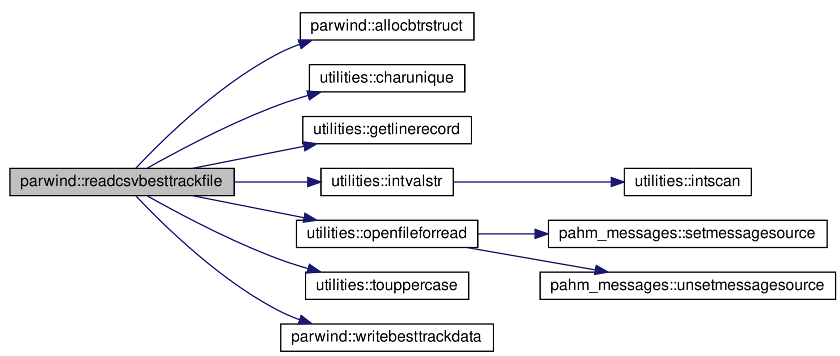
- Skips lines that are time repeats. ???PV check
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 409 of file parwind-orig.F90.

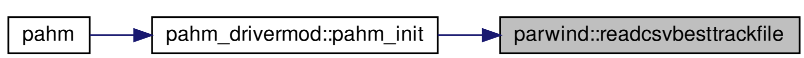
References allocbtrstruct(), besttrackdata, pahm_global::besttrackfilename, utilities::charunique(), utilities::getlinerecord(), utilities::intvalstr(), pahm_global::nbtrfiles, utilities::openfileforread(), utilities::touppercase(), and writebesttrackdata().

Referenced by pahm_drivermod::pahm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.2.10 writebesttrackdata() `subroutine parwind::writebesttrackdata (`

```
character(len=*) inpFile,
type(besttrackdata_t), intent(in) btrStruc,
character(len=*), intent(in), optional suffix )
```

Writes the best track data (adjusted or not) to the "adjusted" best track output file.

Outputs the post-prossed best track data to file.

On Input:

inpFile The name of the input best tack file btrStruc The structure of the "adjusted" best track data suffix The suffix (optional) to be appended to the inpFile (default '_adj')

On Output:

outFile The output file: inpFile // '_adj'

Writes the adjusted (or not) best track data to the "adjusted" best track output file.

Parameters

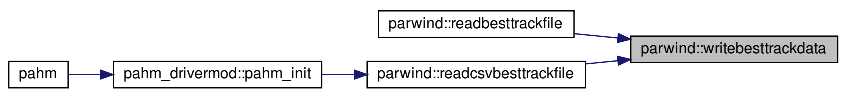
inpFile	The name of the input best track file
btrStruc	The "adjusted" best track data structure that corresponds to the inpFile
suffix	The suffix (optional) to be appended to the inpFile (default '_adj')

Definition at line 1241 of file parwind-orig.F90.

References pahm_global::lun_btrk, and pahm_global::lun_btrk1.

Referenced by readbesttrackfile(), and readcsvbesttrackfile().

Here is the caller graph for this function:



7.11.3 Variable Documentation

7.11.3.1 besttrackdata `type(besttrackdata_t), dimension(:), allocatable, target parwind::besttrackdata`

Definition at line 109 of file parwind-orig.F90.

Referenced by processhollanddata(), readbesttrackfile(), and readcsvbesttrackfile().

7.11.3.2 windreftime `real(sz) parwind::windreftime`

Definition at line 19 of file parwind-orig.F90.

7.12 sortutils Module Reference

Data Types

- interface arraycopy
- interface arrayequal
- interface arth
- interface indexx
- interface swap

Functions/Subroutines

- subroutine indexxint (arr1D, idx1D, status)

    Indexes a 1D integer array in ascending order.
- subroutine indexxint8 (arr1D, idx1D, status)

    Indexes a 1D 32-bit integer array in ascending order.
- subroutine indexxstring (arr1D, idx1D, status, caseSens)

    Indexes a 1D string array in ascending order.
- subroutine indexxsingle (arr1D, idx1D, status)

    Indexes a 1D single precision array in ascending order.
- subroutine indexxdouble (arr1D, idx1D, status)

    Indexes a 1D double precision array in ascending order.
- subroutine quicksort (arr1D, status)

    Sorts the array arr1D into ascending numerical order using Quicksort.
- subroutine sort2 (arr1D, slv1D, status)

    Sorts two 1D arrays into ascending numerical order using Quicksort.
- subroutine arraycopyint (src, dest, nCP, nNCP)

    Copies the 1D source integer array "src" into the 1D destination array "dest".
- subroutine arraycopysingle (src, dest, nCP, nNCP)

    Copies the 1D source single precision array "src" into the 1D destination array "dest".
- subroutine arraycopydouble (src, dest, nCP, nNCP)

    Copies the 1D source double precision array "src" into the 1D destination array "dest".
- logical function arrayequalint (arr1, arr2)

    Compares two one-dimensional integer arrays for equality.
- logical function arrayequalsingle (arr1, arr2)

    Compares two one-dimensional single precision arrays for equality.
- logical function arrayequaldouble (arr1, arr2)

    Compares two one-dimensional double precision arrays for equality.
- integer function stringlexcomp (str1, str2, mSensitive)

    Performs a lexical comparison between two strings.
- subroutine swapint (a, b, mask)

    Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine swapsingle (a, b, mask)

    Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine swapdouble (a, b, mask)

    Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").
- subroutine swapintvec (a, b, mask)

    Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine swapsinglevec (a, b, mask)

    Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine swapdoublevec (a, b, mask)

    Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").
- pure integer function, dimension(n) arthint (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(sp) function, dimension(n) arthsingle (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(hp) function, dimension(n) arthdouble (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

7.12.1 Function/Subroutine Documentation

7.12.1.1 arraycopydouble()

```
subroutine sortutils::arraycopydouble (
        real(hp), dimension(:), intent(in) src,
        real(hp), dimension(:), intent(out) dest,
        integer, intent(out) nCP,
        integer, intent(out) nNCP )
```

Copies the 1D source double precision array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (double precision)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1247 of file sortutils.F90.

7.12.1.2 arraycopyint()

```
subroutine sortutils::arraycopyint (
        integer, dimension(:), intent(in) src,
        integer, dimension(:), intent(out) dest,
        integer, intent(out) nCP,
        integer, intent(out) nNCP )
```

Copies the 1D source integer array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (integer)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1169 of file sortutils.F90.

7.12.1.3 arraycopysingle()

```
subroutine sortutils::arraycopysingle (
    real(sp), dimension(:), intent(in) src,
    real(sp), dimension(:), intent(out) dest,
    integer, intent(out) nCP,
    integer, intent(out) nNCP )
```

Copies the 1D source single precision array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (single precision)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1208 of file sortutils.F90.

7.12.1.4 arrayequaldouble()

```
logical function sortutils::arrayequaldouble (
    real(hp), dimension(:), intent(in) arr1,
    real(hp), dimension(:), intent(in) arr2 )
```

Compares two one-dimensional double precision arrays for equality.

The equality is determined using a tolerance of: 0.00000001, such that the two arrays are considered to be essentially equal on double precision calculations.

Parameters

arr1	The first array in the comparison (double precision)
arr2	The second array in the comparison (double precision)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

Definition at line 1384 of file sortutils.F90.

7.12.1.5 arrayequalint()

```
logical function sortutils::arrayequalint (
        integer, dimension(:), intent(in) arr1,
        integer, dimension(:), intent(in) arr2 )
```

Compares two one-dimensional integer arrays for equality.

Parameters

arr1	The first array in the comparison (integer)
arr2	The second array in the comparison (integer)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

Definition at line 1284 of file sortutils.F90.

7.12.1.6 arrayequalsingle()

```
logical function sortutils::arrayequalsingle (
        real(sp), dimension(:), intent(in) arr1,
        real(sp), dimension(:), intent(in) arr2 )
```

Compares two one-dimensional single precision arrays for equality.

The equality is determined using a tolerance of: 0.00000001, such that the two arrays are considered to be essentially equal on single precision calculations.

Parameters

arr1	The first array in the comparison (single precision)
arr2	The second array in the comparison (single precision)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

Definition at line 1329 of file sortutils.F90.

7.12.1.7 arthdouble()

```
pure real(hp) function, dimension(n) sortutils::arthdouble (
        real(hp), intent(in) first,
        real(hp), intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (double precision)
increment	The value of the increment (double precision)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (double precision)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1952 of file sortutils.F90.

7.12.1.8 arthint()

```
pure integer function, dimension(n) sortutils::arthint (
        integer, intent(in) first,
        integer, intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (integer)
increment	The value of the increment (integer)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (integer)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1818 of file sortutils.F90.

7.12.1.9 arthsingle()

```
pure real(sp) function, dimension(n) sortutils::arthsingle (
        real(sp), intent(in) first,
        real(sp), intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (single precision)
increment	The value of the increment (single precision)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (single precision)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1885 of file sortutils.F90.

7.12.1.10 indexxdouble()

```
subroutine sortutils::indexxdouble (
        real(hp), dimension(:), intent(in) arr1D,
        integer, dimension(:), intent(out) idx1D,
        integer, intent(out), optional status )
```

Indexes a 1D double precision array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (double precision)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 779 of file sortutils.F90.

7.12.1.11 indexxint()

```
subroutine sortutils::indexxint (
        integer, dimension(:), intent(in) arr1D,
        integer, dimension(:), intent(out) idx1D,
        integer, intent(out), optional status )
```

Indexes a 1D integer array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (integer)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 85 of file sortutils.F90.

7.12.1.12 indexxint8()

```
subroutine sortutils::indexxint8 (
        integer(int8), dimension(:), intent(in) arr1D,
        integer, dimension(:), intent(out) idx1D,
        integer, intent(out), optional status )
```

Indexes a 1D 32-bit integer array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (integer)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 257 of file sortutils.F90.

7.12.1.13 indexxsingle()

```
subroutine sortutils::indexxsingle (
        real(sp), dimension(:), intent(in) arr1D,
```

```
integer, dimension(:), intent(out) idx1D,
integer, intent(out), optional status )
```

Indexes a 1D single precision array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (single precision)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 607 of file sortutils.F90.

7.12.1.14 indexxstring()

```
subroutine sortutils::indexxstring (
character(len=*), dimension(:), intent(in) arr1D,
integer, dimension(:), intent(out) idx1D,
integer, intent(out), optional status,
logical, intent(in), optional caseSens )
```

Indexes a 1D string array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed. Modified version of IndexxInt to account for string comparisons

Parameters

arr1D	The array to be indexed (string)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)
caseSens	Logical flag to request case sensitive sort

Definition at line 430 of file sortutils.F90.

7.12.1.15 quicksort()

```
subroutine sortutils::quicksort (
real(sz), dimension(:), intent(inout) arr1D,
integer, intent(out), optional status )
```

Sorts the array arr1D into ascending numerical order using Quicksort.

The array arr1D is replaced on output by its sorted rearrangement. The parameters NN and NSTACK are defined as:

- NN is the size of subarrays sorted by straight insertion, and
- NSTACK is the required auxiliary storage

Parameters

arr1D	The one-dimensional array to be sorted
status	The error status, no error: status = 0 (output)

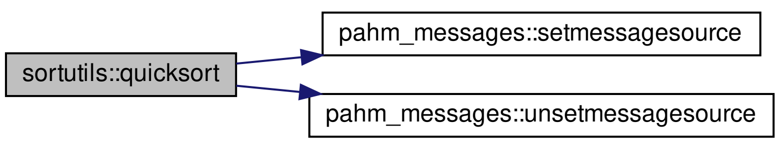
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 951 of file sortutils.F90.

References pahm_messages::error, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



7.12.1.16 sort2()

```
subroutine sortutils::sort2 (
        real(sz), dimension(:), intent(inout) arr1D,
        real(sz), dimension(:), intent(inout) slv1D,
        integer, intent(out), optional status )
```

Sorts two 1D arrays into ascending numerical order using Quicksort.

Sorts the array arr1D into ascending order using Quicksort, while making the corresponding rearrangement of the same-size array slv1D. The sorting and rearrangement are performed by means of the index array.

Parameters

arr1D	The first one-dimensional array to be sorted in ascending order
slv1D	The second one-dimensional array to be sorted in ascending order
status	The error status, no error: status = 0 (output)

Returns

The two sorted arrays arr1D and slv1D

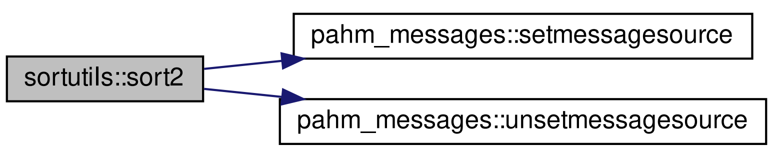
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1098 of file sortutils.F90.

References pahm_messages::error, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



7.12.1.17 stringlexcomp()

```
integer function sortutils::stringlexcomp (
    character(len=*), intent(in) str1,
    character(len=*), intent(in) str2,
    logical, intent(in), optional mSensitive )
```

Performs a lexical comparison between two strings.

Parameters

str1	The first string in the comparison
str2	The second string in the comparison
mSensitive	Logical flag (.TRUE., .FALSE.) to perform case sensitive lexical comparison

Returns

myValOut: The value of the lexical comparison of the two strings (integer)

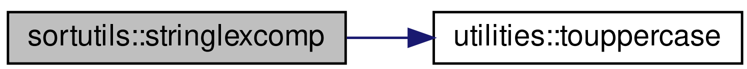
```
myValOut =  0; str1 == str2
myValOut = -1; str1 < str2
myValOut =  1; str1 > str2
```

Definition at line 1443 of file sortutils.F90.

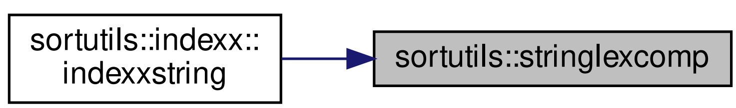
References utilities::touppercase().

Referenced by sortutils::indexx::indexxstring().

Here is the call graph for this function:



Here is the caller graph for this function:



7.12.1.18 swapdouble()

```
subroutine sortutils::swapdouble (
        real(hp), intent(inout) a,
        real(hp), intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (double precision)
b	The second value to be swapped (double precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1613 of file sortutils.F90.

7.12.1.19 swapdoublevec()

```
subroutine sortutils::swapdoublevec (
    real(hp), dimension(:), intent(inout) a,
    real(hp), dimension(:), intent(inout) b,
    logical, intent(in), optional mask )
```

Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (double precision)
b	The second 1D array to be swapped (double precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1769 of file sortutils.F90.

7.12.1.20 swapint()

```
subroutine sortutils::swapint (
    integer, intent(inout) a,
    integer, intent(inout) b,
    logical, intent(in), optional mask )
```

Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (integer)
b	The second value to be swapped (integer)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1509 of file sortutils.F90.

7.12.1.21 swapintvec()

```
subroutine sortutils::swapintvec (
    integer, dimension(:), intent(inout) a,
    integer, dimension(:), intent(inout) b,
    logical, intent(in), optional mask )
```

Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (integer)
b	The second 1D array to be swapped (integer)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1665 of file sortutils.F90.

7.12.1.22 swapsingle()

```
subroutine sortutils::swapsingle (
    real(sp), intent(inout) a,
    real(sp), intent(inout) b,
    logical, intent(in), optional mask )
```

Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (single precision)
b	The second value to be swapped (single precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1561 of file sortutils.F90.

7.12.1.23 swapsinglevec()

```
subroutine sortutils::swapsinglevec (
    real(sp), dimension(:), intent(inout) a,
    real(sp), dimension(:), intent(inout) b,
    logical, intent(in), optional mask )
```

Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (single precision)
b	The second 1D array to be swapped (single precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1717 of file sortutils.F90.

7.13 timedateutils Module Reference

Data Types

- interface gregtojulday
- interface splitdatetimestring
- interface timeconv

Functions/Subroutines

- subroutine timeconvisec (iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- subroutine timeconvrsec (iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- logical function leapyear (iYear)

  Checks for a leap year.
- integer function yeardays (iYear)

  Determines the days of the year.
- integer function monthdays (iYear, iMonth)

  Determines the days in the month of the year.
- integer function dayofyear (iYear, iMonth, iDay)

  Determines the day of the year.
- real(sz) function gregtojuldayisec (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

  Determines the Julian date from a Gregorian date.
- real(sz) function gregtojuldayrsec (iYear, iMonth, iDay, iHour, iMin, rSec, mJD)

  Determines the Julian date from a Gregorian date.
- real(sz) function gregtojulday2 (iDate, iTime, mJD)

  Determines the Julian date from a Gregorian date.
- subroutine juldaytogreg (julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

  Determines the Julian date from a Gregorian date.
- subroutine dayofyeartogreg (inYR, inDY, iYear, iMonth, iDay)

  Determines the Gregorian date (year, month, day) from a day of the year.
- subroutine splitdatetimestring (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)

  Splits a date string into components.
- subroutine splitdatetimestring2 (inDateTime, iDate, iTime)

  Splits a date string into two components.
- character(len=len(indatetime)) function preprocessdatetimestring (inDateTime)

  Pre-processes an arbitrary date string.
- integer function joindate (iYear, iMonth, iDay)

  Pre-processes an arbitrary date string.
- subroutine splitdate (inDate, iYear, iMonth, iDay)

  Pre-processes an arbitrary date string.
- character(len=64) function datetime2string (year, month, day, hour, min, sec, sep, units, zone, err)

  Constructs a NetCDF time string.
- real(sz) function gettimeconvsec (units, invert)

  Calculates the conversion factor between time units and seconds.
- real(sz) function elapsedsecs (inTime1, inTime2, inUnits)

  Calculates the elapsed time in seconds.

Variables

- integer, parameter firstgregdate = 1582 ∗ 10000 + 10 ∗ 100 + 05
- integer, parameter firstgregtime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter offfirstgregday = 2299150.5_HP
- integer, parameter modjuldate = 1858 ∗ 10000 + 11 ∗ 100 + 17
- integer, parameter modjultime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter offmodjulday = 2400000.5_HP
- integer, parameter unixdate = 1970 ∗ 10000 + 1 ∗ 100 + 1
- integer, parameter unixtime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter offunixjulday = 2440587.5_HP
- integer, parameter modeldate = 1990 ∗ 10000 + 1 ∗ 100 + 1
- integer, parameter modeltime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter offmodeljulday = 2447892.5_HP
- integer, parameter usemodjulday = 0
- integer, parameter mdjdate = UNIXDATE
- integer, parameter mdjtime = UNIXTIME
- real(hp), parameter mdjoffset = OFFUNIXJULDAY

7.13.1 Function/Subroutine Documentation

7.13.1.1 datetime2string()

```
character(len=64) function timedateutils::datetime2string (
        integer, intent(in) year,
        integer, intent(in) month,
        integer, intent(in) day,
        integer, intent(in), optional hour,
        integer, intent(in), optional min,
        integer, intent(in), optional sec,
        integer, intent(in), optional sep,
        character(len=*), intent(in), optional units,
        character(len=*), intent(in), optional zone,
        integer, intent(out), optional err )
```

Constructs a NetCDF time string.

This function joins the values of the year, month, day, hour, min, sec to construct the date string used in NetCDF files.

Parameters

year	The year (YYYY)
month	The month of the year (MM)
day	The day of the month (DD)
hour	The hour of the day (hh) (optional - 0 is substituded if not supplied)
min	The minute of the hour (mm) (optional - 0 is substituded if not supplied)
sec	The second of the minute (ss) (optional - 0 is substituded if not supplied)
sep	The seperation character between the date part and the time part
units	The units part to be prepented to the datetime string in the form '<units> since'
zone	The timezone to use (default none/UTC, optional)
err	The error status, no error: status = 0 (output)

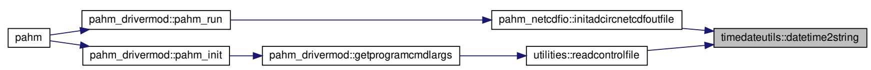
Returns

myValOut: The datetime string ([<units> since]YYYY-MM-DD hh:mm:ss)

Definition at line 1336 of file timedateutils.F90.

Referenced by pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::readcontrolfile().

Here is the caller graph for this function:



7.13.1.2 dayofyear()

```
integer function timedateutils::dayofyear (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay )
```

Determines the day of the year.

This function calculates "the day of year" number given the year, month, day, for a Gregorian year (>= 1582). In case of an error, the value IMISSV (-999999) is returned.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <= 12)
iDay	The day of the month (DD, integer, 1 <= DD <= 31)

Returns

myVal: The day of the year number (also erroneously known as Julian day). This the number of days since the first day of the year (01/01).

Definition at line 460 of file timedateutils.F90.

References pahm_sizes::imissv, and pahm_sizes::rmissv.

7.13.1.3 dayofyeartogreg()

```
subroutine timedateutils::dayofyeartogreg (
    integer, intent(in) inYR,
    integer, intent(in) inDY,
    integer, intent(out) iYear,
    integer, intent(out) iMonth,
    integer, intent(out) iDay )
```

Determines the Gregorian date (year, month, day) from a day of the year.

This subroutine computes the calendar year, month and day from given "year" and "day of the year". In case of error, year is set equal to IMISSV (-999999). Gregorian date (after 10/05/1582), or the value RMISSV if an error occurred.

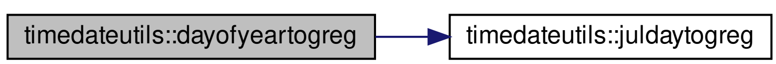
Parameters

inYR	The year (YYYY, integer, 1582 <= YYYY)
inDY	The day of the year (DDD, integer, 1 <= DDD <= 366)
iYear	The year (YYYY, integer, 1582 <= YYYY, output)
iMonth	The month of the year (MM, integer, 1 <= MM <=12, output)
iDay	The day of the month (DD, integer, 1 <= DD <=31, output)

Definition at line 1011 of file timedateutils.F90.

References juldaytogreg().

Here is the call graph for this function:



7.13.1.4 elapsedsecs()

```
real(sz) function timedateutils::elapsedsecs (
    real(sz), intent(in) inTime1,
    real(sz), intent(in) inTime2,
    character(len=*), intent(in), optional inUnits )
```

Calculates the elapsed time in seconds.

This function computes the elapsed time in sec, between times1 and time2, given the units of the times.

Parameters

inTime1	The start time (real)
inTime2	The end time (real)

Parameters

inUnits	The units (string, optional) of the time variables. Available options: For converting days to seconds : inUnits = ['DAYS', 'DAY', 'DA', 'D'] For converting hours to seconds: inUnits = ['HOURS', 'HOUR', 'HOU', 'HO', 'H'] For converting seconds to seconds: inUnits = ['SEC', 'SE', 'SC', 'S'] Default: inUnits = ['SEC', 'SE', 'SC', 'S']

Returns

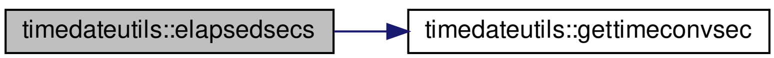
myVal: The elapsed time in seconds (real). If this value is very close, within a tolerance, to the nearest whole number, it is set equal to that number.

Definition at line 1517 of file timedateutils.F90.

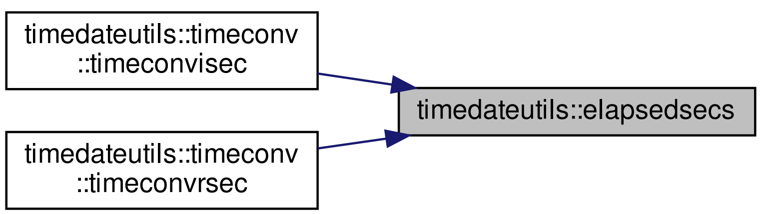
References gettimeconvsec().

Referenced by timedateutils::timeconv::timeconvisec(), and timedateutils::timeconv::timeconvrsec().

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.1.5 gettimeconvsec()

```
real(sz) function timedateutils::gettimeconvsec (
        character(len=*), intent(in) units,
        integer, intent(in), optional invert )
```

Calculates the conversion factor between time units and seconds.

This function returns the converion factor between timeUnit and seconds. If invert $>$ 0 then the function returns the inverse conversion factor, seconds to timeUnit.

Parameters

units	The time unit used in the calculations (string: S, M, H, D, W)
invert	To perform the inverted conversion, froms seconds to timeUnit (optional) where: S=seconds, M=minutes, H=hours, D=days, W=weeks

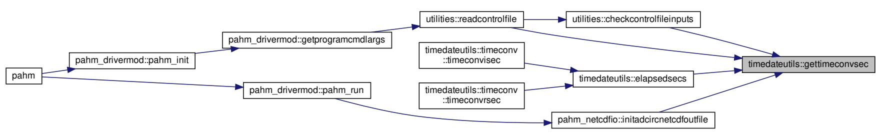
Returns

myValOut: The conversion factor

Definition at line 1431 of file timedateutils.F90.

Referenced by utilities::checkcontrolfileinputs(), elapsedsecs(), pahm_netcdfio::initadcircnetcdfoutfile(), and utilities::readcontrolfile().

Here is the caller graph for this function:



7.13.1.6 gregtojulday2()

```
real(sz) function timedateutils::gregtojulday2 (
        integer, intent(in) iDate,
        integer, intent(in) iTime,
        integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian).
It is usefull to compute differences between dates.
Similar to GregToJulDayISEC but the seconds number is real to allow for second fractions.

Parameters

iDate	The date as YYYYMMDD (integer)
	`YYYY     The year (YYYY, integer, 1582 <= YYYY)`
	`  MM     The month of the year (MM, integer, 1 <= MM <=12)`
	`  DD     The day of the month (DD, integer, 1 <= DD <=31)`

iTime	The time as hhmmss (integer)
	`  hh     The hour of the day (integer, 0 <= hh <= 23)`
	`  mm     The minute of the hour (integer, 0 <= mm <= 59)`
	`  ss     The second of the minute (integer, 0 <= ss <= 60)`

Parameters

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use:                      mJD  < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 776 of file timedateutils.F90.

7.13.1.7 gregtojuldayisec()

```
real(sz) function timedateutils::gregtojuldayisec (
    integer, intent(in) iYear,
    integer, intent(in) iMonth,
    integer, intent(in) iDay,
    integer, intent(in) iHour,
    integer, intent(in) iMin,
    integer, intent(in) iSec,
    integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is usefull to compute differences between dates.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <=12)
iDay	The day of the month (DD, integer, 1 <= DD <=31)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59)
iSec	iSec The second of the minute (ss, integer, 0 <= ss <= 59)

Parameters

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use:                        mJD  < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 536 of file timedateutils.F90.

7.13.1.8 gregtojuldayrsec()

```
real(sz) function timedateutils::gregtojuldayrsec (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay,
        integer, intent(in) iHour,
        integer, intent(in) iMin,
        real(sz), intent(in) rSec,
        integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian).
It is usefull to compute differences between dates.
Similar to GregToJulDayISEC but the seconds number is real to allow for second fractions.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <=12)
iDay	The day of the month (DD, integer, 1 <= DD <=31)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59)

Parameters

rSec	The second of the minute (ss, real, 0 <= ss <= 59)

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use:                     mJD  < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 655 of file timedateutils.F90.

7.13.1.9 joindate()

```
integer function timedateutils::joindate (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay )
```

Pre-processes an arbitrary date string.

This function joins the three integers iYear, iMonth and iDay to calculate the integer inDate (YYYYMMDD). There is no check on the validity of iYear, iMonth, iDay, therefore the user is responsible to supply valid input values.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <=12)
iDay	The day of the month (DD, integer, 1 <= DD <=31)

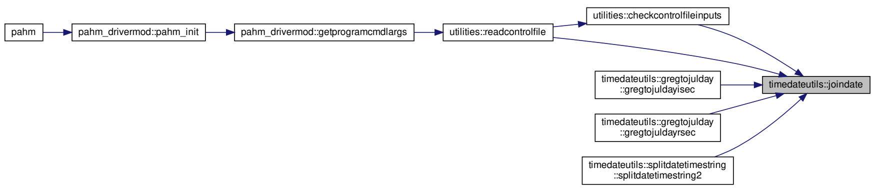
Returns

myValOut: The integer date (YYYYMMDD)

Definition at line 1241 of file timedateutils.F90.

Referenced by utilities::checkcontrolfileinputs(), timedateutils::gregtojulday::gregtojuldayisec(), timedateutils::gregtojulday::gregtojuldayrse utilities::readcontrolfile(), and timedateutils::splitdatetimestring::splitdatetimestring2().

Here is the caller graph for this function:



7.13.1.10 juldaytogreg()

```
subroutine timedateutils::juldaytogreg (
    real(sz), intent(in) julDay,
    integer, intent(out) iYear,
    integer, intent(out) iMonth,
    integer, intent(out) iDay,
    integer, intent(out) iHour,
    integer, intent(out) iMin,
    integer, intent(out) iSec,
    integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This subroutine computes the calendar year, month, day, hour, minute and second corresponding to a given Julian date. The inverse of this procedure is the function GregToJulDay. In case of error, year is set equal to IMISSV (-999999). Considers Gregorian dates (after 10/05/1582) only.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is usefull to compute differences between dates.

Parameters

<table>
<tr><td>julDay</td><td>The Julian day number (double).</td></tr>
<tr><td>mJD</td><td>Flag to use a modified julian day number or not<pre>
To use a modified julian day number use: mJD >= 1
otherwise use: mJD < 1
default: mJD = 0
The modified julian day number (MJD) was defined in
the mid 1950's in the interests of astronomy and space science
as MJD = JD - 2400000.5. The half day shift makes the day start
at midnight, which is the current time standard.
Subtracting the large number shifts the zero day to a more
recent time (November 17, 1858, midnight) allowing smaller numbers
to represent time.
</pre></td></tr>
<tr><td>iYear</td><td>The year (YYYY, integer, 1582 <= YYYY, output)</td></tr>
<tr><td>iMonth</td><td>The month of the year (MM, integer, 1 <= MM <=12, output)</td></tr>
<tr><td>iDay</td><td>The day of the month (DD, integer, 1 <= DD <=31, output)</td></tr>
<tr><td>iHour</td><td>The hour of the day (hh, integer, 0 <= hh <= 23, output)</td></tr>
<tr><td>iMin</td><td>The minute of the hour (mm, integer, 0 <= mm <= 59, output)</td></tr>
<tr><td>iSec</td><td>The second of the minute (ss, integer, 0 <= ss <= 59, output)</td></tr>
</table>

Note

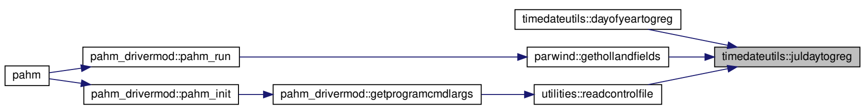
The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 899 of file timedateutils.F90.

References mdjoffset, offfirstgregday, and usemodjulday.

Referenced by dayofyeartogreg(), parwind::gethollandfields(), and utilities::readcontrolfile().

Here is the caller graph for this function:



7.13.1.11 leapyear() `logical function timedateutils::leapyear ( integer, intent(in) iYear )`

Checks for a leap year.

This function tries to determine if a Gregorian year (>= 1582) is a leap year or not.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)

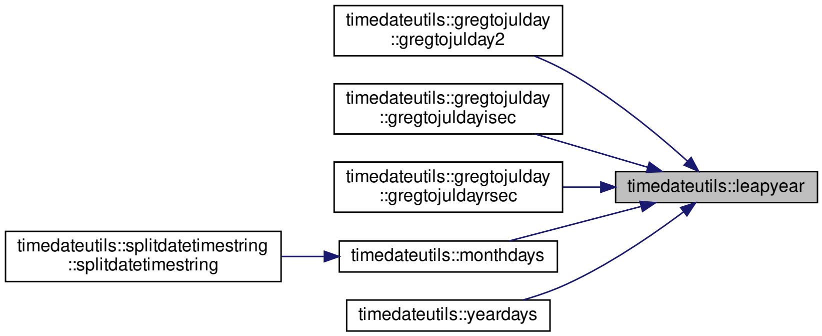
Returns

myVal: .TRUE. if it is a leap year or .FALSE. otherwise

Definition at line 315 of file timedateutils.F90.

Referenced by timedateutils::gregtojulday::gregtojulday2(), timedateutils::gregtojulday::gregtojuldayisec(), timedateutils::gregtojulday::greg monthdays(), and yeardays().

Here is the caller graph for this function:



7.13.1.12 monthdays()

`integer function timedateutils::monthdays (`
`integer, intent(in) iYear,`
`integer, intent(in) iMonth )`

Determines the days in the month of the year.

This function calculates the number of calendar days in a month of a Gregorian year (>= 1582). In case of an error, the value IMISSV (-999999) is returned.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <= 12)

Returns

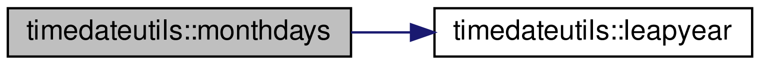
myVal: The days of the month

Definition at line 403 of file timedateutils.F90.

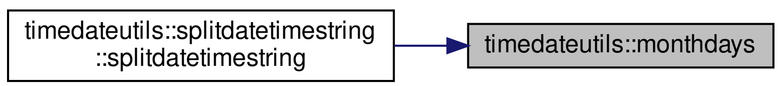
References pahm_sizes::imissv, and leapyear().

Referenced by timedateutils::splitdatetimestring::splitdatetimestring().

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.1.13 preprocessdatetimestring() `character(len=len(indatetime)) function timedateutils::preprocessdatetimestrin`

```
(
            character(len=*), intent(in) inDateTime )
```

Pre-processes an arbitrary date string.

This function returns a date/time string in the format YYYYMMDDhhmmss by removing all non-numeric characters from the string.

Parameters

inDateTime	The input date string

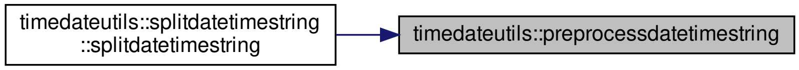
Returns

myValOut: The string datetime as an integer in the form: YYYYMMDDhhmmss

Definition at line 1186 of file timedateutils.F90.

Referenced by timedateutils::splitdatetimestring::splitdatetimestring().

Here is the caller graph for this function:



7.13.1.14 splitdate()

```
subroutine timedateutils::splitdate (
    integer, intent(in) inDate,
    integer, intent(out) iYear,
    integer, intent(out) iMonth,
    integer, intent(out) iDay )
```

Pre-processes an arbitrary date string.

This subroutine splits the integer inDate (YYYYMMDD) in three integers that is, "iYear (YYYY)", "iMonth (MM)" and "iDay (DD)". There is no check on the validity of inDate, the user is responsible to supply a valid input date.

Parameters

inDate	The integer date (YYYYMMDD)
iYear	The year (YYYY, integer, 1582 <= YYYY, output)
iMonth	The month of the year (MM, integer, 1 <= MM <=12, output)
iDay	The day of the month (DD, integer, 1 <= DD <=31, output)

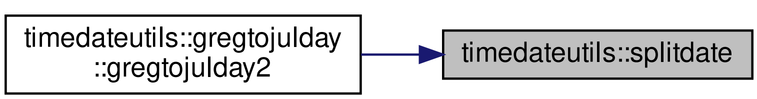
Note

The code was adopted from the D-Flow FM source (time_module.f90/splitDate)

Definition at line 1281 of file timedateutils.F90.

Referenced by timedateutils::gregtojulday::gregtojulday2().

Here is the caller graph for this function:



7.13.1.15 splitdatetimestring()

```
subroutine timedateutils::splitdatetimestring (
    character(len=*), intent(in) inDateTime,
    integer, intent(out) iYear,
    integer, intent(out) iMonth,
    integer, intent(out) iDay,
    integer, intent(out) iHour,
    integer, intent(out) iMin,
    integer, intent(out) iSec )
```

Splits a date string into components.

This subroutine splits the string inDate (YYYYMMDDhhmmss) in six integers that is, "iYear (YYYY)", "iMonth (MM)", "iDay (DD)", "iHour (hh)", "iMin (mm)" and "iSec (ss)".

Parameters

inDateTime	The input date string: YYYYMMDDhhmmss
iYear	The year (YYYY, integer, 1582 <= YYYY, output)
iMonth	The month of the year (MM, integer, 1 <= MM <=12, output)
iDay	The day of the month (DD, integer, 1 <= DD <=31, output)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23, output)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
iSec	The second of the minute (ss, integer, 0 <= ss <= 59, output)

Definition at line 1073 of file timedateutils.F90.

7.13.1.16 splitdatetimestring2()

```
subroutine timedateutils::splitdatetimestring2 (
    character(len=*), intent(in) inDateTime,
    integer, intent(out) iDate,
    integer, intent(out) iTime )
```

Splits a date string into two components.

This subroutine splits the string inDate (YYYYMMDDhhmmss) in two integers that is, "iDate (YYYYMMDD)" and "iTime (hhmmss)".

Parameters

inDateTime	The input date string: YYYYMMDDhhmmss
iDate	The integer date (YYYYMMDD, output)
iTime	The integer time (hhmmss, output)

Definition at line 1141 of file timedateutils.F90.

7.13.1.17 timeconvisec()

```
subroutine timedateutils::timeconvisec (
    integer, intent(in) iYear,
    integer, intent(in) iMonth,
    integer, intent(in) iDay,
    integer, intent(in) iHour,
    integer, intent(in) iMin,
    integer, intent(in) iSec,
    real(sz), intent(out) timeSec )
```

Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

The reference date is defined by the global variables: refYear, refMonth, refDay, refHour, refMin and refSec. It uses GregToJulDay and ElapsedSecs functions to calculate the elapsed time from the reference date.

Parameters

iYear	The year (integer)
iMonth	The month of the year (1-12, integer)
iDay	The day of the month (1-31, integer)
iHour	The hour of the day (0-23, integer)
iMin	The minute of the hour (0-59, integer)
iSec	The second of the minute (0-59, integer)
timeSec	The elapsed time in seconds (real, output)

Definition at line 125 of file timedateutils.F90.

7.13.1.18 timeconvrsec()

```
subroutine timedateutils::timeconvrsec (
    integer, intent(in) iYear,
    integer, intent(in) iMonth,
    integer, intent(in) iDay,
    integer, intent(in) iHour,
    integer, intent(in) iMin,
    real(sz), intent(in) rSec,
    real(sz), intent(out) timeSec )
```

Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

The reference date is defined by the global variables: refYear, refMonth, refDay, refHour, refMin and refSec. It uses GregToJulDay and ElapsedSecs functions to calculate the elapsed time from the reference date. Similar to TimeConv↩ ISEC but seconds are entered as real numbers to allow for fractions of a second.

Parameters

iYear	The year (integer)
iMonth	The month of the year (1-12, integer)
iDay	The day of the month (1-31, integer)
iHour	The hour of the day (0-23, integer)
iMin	The minute of the hour (0-59, integer)
rSec	The second of the minute (0-59, real)
timeSec	The elapsed time in seconds (real, output)

Definition at line 202 of file timedateutils.F90.

7.13.1.19 yeardays() `integer function timedateutils::yeardays ( integer, intent(in) iYear )`

Determines the days of the year.

This function calculates the number of calendar days of a Gregorian year (>= 1582).

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)

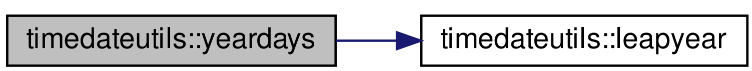
Returns

myVal: The days of the year (365 or 366)

Definition at line 366 of file timedateutils.F90.

References leapyear().

Here is the call graph for this function:



7.13.2 Variable Documentation

7.13.2.1 firstgregdate `integer, parameter timedateutils::firstgregdate = 1582 * 10000 + 10 * 100 + 05`

Definition at line 44 of file timedateutils.F90.

Referenced by utilities::checkcontrolfileinputs(), timedateutils::gregtojulday::gregtojulday2(), timedateutils::gregtojulday::gregtojuldayisec(), and timedateutils::gregtojulday::gregtojuldayrsec().

7.13.2.2 firstgregtime `integer, parameter timedateutils::firstgregtime = 0 * 10000 + 0 * 100 + 0`

Definition at line 45 of file timedateutils.F90.

Referenced by utilities::checkcontrolfileinputs().

7.13.2.3 mdjdate `integer, parameter timedateutils::mdjdate = UNIXDATE`

Definition at line 80 of file timedateutils.F90.

7.13.2.4 mdjoffset `real(hp), parameter timedateutils::mdjoffset = OFFUNIXJULDAY`

Definition at line 82 of file timedateutils.F90.

Referenced by timedateutils::gregtojulday::gregtojulday2(), timedateutils::gregtojulday::gregtojuldayisec(), timedateutils::gregtojulday::greg
and juldaytogreg().

7.13.2.5 mdjtime `integer, parameter timedateutils::mdjtime = UNIXTIME`

Definition at line 81 of file timedateutils.F90.

7.13.2.6 modeldate `integer, parameter timedateutils::modeldate = 1990 * 10000 + 1 * 100 + 1`

Definition at line 65 of file timedateutils.F90.

7.13.2.7 modeltime `integer, parameter timedateutils::modeltime = 0 * 10000 + 0 * 100 + 0`

Definition at line 66 of file timedateutils.F90.

7.13.2.8 modjuldate `integer, parameter timedateutils::modjuldate = 1858 * 10000 + 11 * 100 + 17`

Definition at line 53 of file timedateutils.F90.

7.13.2.9 modjultime `integer, parameter timedateutils::modjultime = 0 * 10000 + 0 * 100 + 0`

Definition at line 54 of file timedateutils.F90.

7.13.2.10 offfirstgregday `real(hp), parameter timedateutils::offfirstgregday = 2299150.5_HP`

Definition at line 46 of file timedateutils.F90.

Referenced by juldaytogreg().

7.13.2.11 offmodeljulday `real(hp), parameter timedateutils::offmodeljulday = 2447892.5_HP`

Definition at line 67 of file timedateutils.F90.

7.13.2.12 offmodjulday `real(hp), parameter timedateutils::offmodjulday = 2400000.5_HP`

Definition at line 55 of file timedateutils.F90.

7.13.2.13 offunixjulday `real(hp), parameter timedateutils::offunixjulday = 2440587.5_HP`

Definition at line 61 of file timedateutils.F90.

7.13.2.14 unixdate `integer, parameter timedateutils::unixdate = 1970 * 10000 + 1 * 100 + 1`

Definition at line 59 of file timedateutils.F90.

7.13.2.15 unixtime `integer, parameter timedateutils::unixtime = 0 * 10000 + 0 * 100 + 0`

Definition at line 60 of file timedateutils.F90.

7.13.2.16 usemodjulday `integer, parameter timedateutils::usemodjulday = 0`

Definition at line 72 of file timedateutils.F90.

Referenced by timedateutils::gregtojulday::gregtojulday2(), timedateutils::gregtojulday::gregtojuldayisec(), timedateutils::gregtojulday::greg
and juldaytogreg().

7.14 utilities Module Reference

Data Types

- interface cpptogeo
- interface geotocpp
- interface sphericaldistance

Functions/Subroutines

- subroutine openfileforread (lun, fileName, errorIO)

    This subroutine opens an existing file for reading.
- subroutine readcontrolfile (inpFile)

    This subroutine reads the program's main control file.
- subroutine printmodelparams ()

    This subroutine prints on the screen the values of the program's parameters.
- integer function getlinerecord (inpLine, outLine, lastCommFlag)

    Gets a line from a file.
- integer function parseline (inpLine, outLine, keyWord, nVal, cVal, rVal)

    This function parses lines of text from input script/control files.
- integer function checkcontrolfileinputs ()

    Checks the user defined control file inputs.
- integer function loadintvar (nInp, vInp, nOut, vOut)

    This function loads input values into a requested model integer variable.
- integer function loadlogvar (nInp, vInp, nOut, vOut)

    This function loads input values into a requested model logical variable.
- integer function loadrealvar (nInp, vInp, nOut, vOut)

    This function loads input values into a requested model real variable.
- pure character(len(inpstring)) function tolowercase (inpString)

    Convert a string to lower-case.
- pure character(len(inpstring)) function touppercase (inpString)

    Convert a string to upper-case.
- real(sz) function convlon (inpLon)

    Convert longitude values from the (0, 360) to the (-180, 180) notation.
- subroutine geotocpp_scalar (lat, lon, lat0, lon0, x, y)

    Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.
- subroutine geotocpp_1d (lat, lon, lat0, lon0, x, y)

    Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.
- subroutine cpptogeo_scalar (x, y, lat0, lon0, lat, lon)

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

- subroutine cpptogeo_1d (x, y, lat0, lon0, lat, lon)

  Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- real(sz) function sphericaldistance_scalar (lat1, lon1, lat2, lon2)

  Calculates the distance of two points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:), allocatable sphericaldistance_1d (lats, lons, lat0, lon0)

  Calculates the distance of points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:, :), allocatable sphericaldistance_2d (lats, lons, lat0, lon0)

  Calculates the distance of points along the great circle using the Vincenty formula.
- real(sz) function sphericaldistanceharv (lat1, lon1, lat2, lon2)

  Calculates the distance of two points along the great circle using the Haversine formula.
- subroutine sphericalfracpoint (lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)

  Calculates the coordinates of an intermediate point between two points along the great circle.
- subroutine getlocandratio (val, arrVal, idx1, idx2, wtRatio)

  Calculates the location of a value in an 1D array of values.
- integer function charunique (inpVec, outVec, idxVec)

  Find the unique non-blank elements in 1D character array.
- real(sp) function valstr (String)

  Returns the value of the leading double precision real numeric string.
- real(hp) function dvalstr (String)

  Returns the value of the leading double precision real numeric string.
- integer function intvalstr (String)

  Returns the value of the leading integer numeric string.
- integer function realscan (String, Pos, Value)

  Scans string looking for the leading single precision real numeric string.
- integer function drealscan (String, Pos, Value)

  Scans string looking for the leading double precision real numeric string.
- integer function intscan (String, Pos, Signed, Value)

  Scans string looking for the leading integer numeric string.

Variables

- real(sz), parameter closetol = 0.001_SZ

7.14.1 Function/Subroutine Documentation

7.14.1.1 charunique()

```
integer function utilities::charunique (
        character(len=*), dimension(:), intent(in) inpVec,
        character(len=*), dimension(:), intent(out) outVec,
        integer, dimension(:), intent(out), allocatable idxVec )
```

Find the unique non-blank elements in 1D character array.

Parameters

inpVec	The input 1D string array
outVec	The output 1D string array of the unique elements (output)
idxVec	The 1D array of indexes of the unique elements in the inpVec array (output)

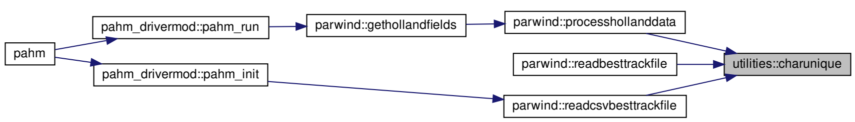
Returns

myRec: The number of the uniques elements in the input array

Definition at line 2576 of file utilities.F90.

Referenced by parwind::processhollanddata(), parwind::readbesttrackfile(), and parwind::readcsvbesttrackfile().

Here is the caller graph for this function:



7.14.1.2 checkcontrolfileinputs() `integer function utilities::checkcontrolfileinputs`

Checks the user defined control file inputs.

The purpose of this subroutine is to process the input parameters and check if the user supplied values in the control file are valid entries. If a value for an input parameter is not supplied, then a default value is assigned to that parameter. If the parameter doesn't have a default value, it is then a mandatory parameter that the user needs to supply a valid value.

Returns

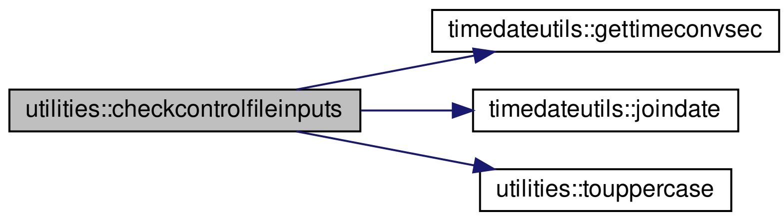
myStatus: The error status, no error: status = 0

Definition at line 1140 of file utilities.F90.

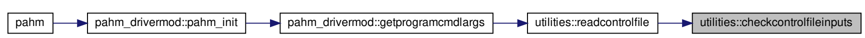
References pahm_global::backgroundatmpress, pahm_global::begsimspecified, pahm_global::begsimtime, pahm_global::besttrackfilenan pahm_global::besttrackfilenamespecified, pahm_global::bladjustfac, closetol, pahm_global::def_ncnam_pres, pahm_global::def_ncnam_wndx, pahm_global::def_ncnam_wndy, pahm_global::defv_atmpress, pahm_global::defv_bladjustfac, pahm_global::defv_gravity, pahm_global::defv_rhoair, pahm_global::defv_rhowater, pahm_global::endsimspecified, pahm_global::endsimtime, timedateutils::firstgregdate, timedateutils::firstgregtime, timedateutils::gettimeconvsec(), pahm_global::gravity, timedateutils::joindate(), pahm_global::mdbegsimtime, pahm_global::mdendsimtime, pahm_global::mdoutdt, pahm_global::meshfileform, pahm_global::meshfilename, pahm_global::meshfilenamespecified, pahm_global::meshfiletype, pahm_global::modeltype, pahm_global::nbtrfiles, pahm_global::ncdeflate, pahm_global::ncdlevel, pahm_global::ncshuffle, pahm_global::ncvarnam_pres, pahm_global::ncvarnam_wndx, pahm_global::ncvarnam_wndy, pahm_global::noutdt, pahm_global::outdt, pahm_global::outfilename, pahm_global::outfilenamespecified, pahm_global::refdate, pahm_global::refdatetime, pahm_global::reftime, pahm_global::rhoair, pahm_global::rhowater, touppercase(), and pahm_global::unittime.

Referenced by readcontrolfile().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.3 convlon() `real(sz) function utilities::convlon ( real(sz) inpLon )`

Convert longitude values from the (0, 360) to the (-180, 180) notation.

Parameters

inpLon	The longitude value to be converted

Returns

myValOut: The converted longitude value

Definition at line 1760 of file utilities.F90.

7.14.1.4 cpptogeo_1d()

```
subroutine utilities::cpptogeo_1d (
        real(sz), dimension(:), intent(in) x,
        real(sz), dimension(:), intent(in) y,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), dimension(:), intent(out) lat,
        real(sz), dimension(:), intent(out) lon )
```

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

Transforms 1D CPP coordinates into 1D geographical coordinates. This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

x	X coordinate: x (m) - real, 1D array
y	Y coordinate: y (m) - real, 1D array
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
lat	Latitude (degrees north) - real, 1D array (output)
lon	Longitude (degrees east) - real, 1D array (output)

Definition at line 1940 of file utilities.F90.

7.14.1.5 cpptogeo_scalar()

```
subroutine utilities::cpptogeo_scalar (
        real(sz), intent(in) x,
        real(sz), intent(in) y,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), intent(out) lat,
        real(sz), intent(out) lon )
```

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

x	X coordinate: x (m) - real, scalar
y	Y coordinate: y (m) - real, scalar
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
lat	Latitude (degrees north) - real, scalar (output)
lon	Longitude (degrees east) - real, scalar (output)

Definition at line 1893 of file utilities.F90.

7.14.1.6 drealscan()

```
integer function utilities::drealscan (
        character(len=*), intent(in) String,
        integer, intent(in) Pos,
        real(hp), intent(out) Value )
```

Scans string looking for the leading double precision real numeric string.

Scanning begins at the position specified by pos and continues to the end of the string. Leading blanks are ignored.

```
The numeric string must have the form:
[sign] d+ ['.' d*] ['e' [sign] d+]        or
[sign]     '.' d+  ['e' [sign] d+]
where sign is '+' or '-',
d* is zero or more digits,
d+ is one  or more digits,
'.' and 'e' are literal (also accept lower case 'e'),
brackets [, ] delimit optional sequences.

Value is set to the numeric value of the string.
The function value is set to the position within the string where
the numeric string ends plus one (i.e., the break character).
```

Parameters

String	The input string
Pos	The position in the input string where the scanning begins
Value	The numeric value of the string

Returns

myVal: The position within the string where the numeric string ends plus one (i.e., the break character)

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

See also

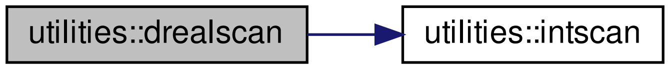
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 2942 of file utilities.F90.

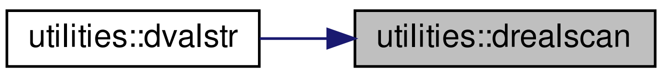
References intscan().

Referenced by dvalstr().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.7 dvalstr() `real(hp) function utilities::dvalstr ( character(len=*), intent(in) String )`

Returns the value of the leading double precision real numeric string.

Parameters

String	The input string

Returns

myVal: The value of the real number in double precision as extracted from the input string

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

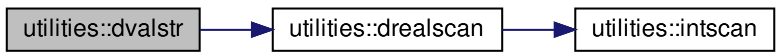
See also

https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 2684 of file utilities.F90.

References drealscan().

Here is the call graph for this function:



7.14.1.8 geotocpp_1d()

```
subroutine utilities::geotocpp_1d (
    real(sz), dimension(:), intent(in) lat,
    real(sz), dimension(:), intent(in) lon,
    real(sz), intent(in) lat0,
    real(sz), intent(in) lon0,
    real(sz), dimension(:), intent(out) x,
    real(sz), dimension(:), intent(out) y )
```

Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

Transforms 1D geographical coordinates into 1D CPP coordinates. This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

lat	Latitude (degrees north) - real, 1D array
lon	Longitude (degrees east) - real, 1D array
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
x	Calculated X coordinate: x (m) - real, 1D array (output)
y	Calculated Y coordinate: y (m) - real, 1D array (output)

Definition at line 1847 of file utilities.F90.

7.14.1.9 geotocpp_scalar()

```
subroutine utilities::geotocpp_scalar (
        real(sz), intent(in) lat,
        real(sz), intent(in) lon,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), intent(out) x,
        real(sz), intent(out) y )
```

Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

lat	Latitude (degrees north) - real, scalar
lon	Longitude (degrees east) - real, scalar
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
x	Calculated X coordinate: x (m) - real, scalar (output)
y	Calculated Y coordinate: y (m) - real, scalar (output)

Definition at line 1800 of file utilities.F90.

7.14.1.10 getlinerecord()

```
integer function utilities::getlinerecord (
        character(len=*), intent(in) inpLine,
        character(len=len(inpline)), intent(out) outLine,
        integer, optional lastCommFlag )
```

Gets a line from a file.

This function reads a line record, which is neither a commented or a blank line, from a file for further processing. Commented lines are those with a first character either "#" or "!".

Parameters

inpLine	The input text line
lastCommFlag	Optional flag to check/remove commented portion at the right of the text line lastCommFlag <= 0 do nothing lastCommFlag > 0 check for "#!" symbols at the right of the text line and remove that portion of the line
outLine	The output line (the left adjusted input line)

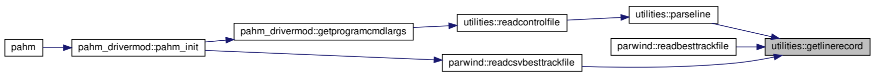
Returns

myLen: The length of outLine (end blanks removed)

Definition at line 773 of file utilities.F90.

Referenced by parseline(), parwind::readbesttrackfile(), and parwind::readcsvbesttrackfile().

Here is the caller graph for this function:



7.14.1.11 getlocandratio()

```
subroutine utilities::getlocandratio (
        real(sz), intent(in) val,
        real(sz), dimension(:), intent(in) arrVal,
        integer, intent(out) idx1,
        integer, intent(out) idx2,
        real(sz), intent(out) wtRatio )
```

Calculates the location of a value in an 1D array of values.

Determines the linear interpolation parameters given the 1D input search array arrVal and the search value val. The linear interpolation is performed using the equation: VAR(estimated) = VAR(idx1) + wtRatio ∗ (VAR(idx2) - VAR(idx1)).

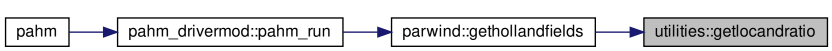
Parameters

val	The value to search for, such that arrVal(idx1) <= val <= arrVal(idx2)
arrVal	The one-dimensional array to search (PV ordered in ascending order?)
idx1	The index of the lowest array bound such that: arrVal(idx1) <= val (output)
idx2	The index of the highest array bound such that: arrVal(idx2) >= val (output)
wtRatio	The ratio factor used in the linear interpolation calculation: VAR(estimated) = VAR(idx1) + wtRatio ∗ (VAR(idx2) - VAR(idx1)) where VAR is the variable to be interpolated

Definition at line 2462 of file utilities.F90.

Referenced by parwind::gethollandfields().

Here is the caller graph for this function:



7.14.1.12 intscan()

```
integer function utilities::intscan (
        character(len=*), intent(in) String,
        integer, intent(in) Pos,
        logical, intent(in) Signed,
        integer, intent(out) Value )
```

Scans string looking for the leading integer numeric string.

Scanning begins at the position specified by pos and continues to the end of the string. Leading blanks are ignored.

```
The search may be for a signed (signed = .true.) or unsigned
(signed = .FALSE.) integer value.  If signed, leading plus (+) or minus (-)
is allowed.  If unsigned, they will terminate the scan as they are
invalid for an unsigned integer.

Value is set to the numeric value of the string.
The function value is set to the position within the string where
the numeric string ends plus one (i.e., the break character).
```

Parameters

String	The input string
Pos	The position in the input string where the scanning begins
Signed	The sign (+, -) of the numeric string, if present
Value	The numeric value of the string

Returns

myVal: The position within the string where the numeric string ends plus one (i.e., the break character)

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

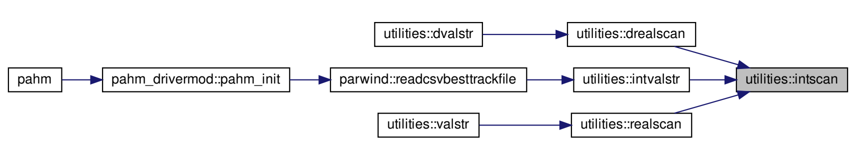
See also

https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 3095 of file utilities.F90.

Referenced by drealscan(), intvalstr(), and realscan().

Here is the caller graph for this function:



7.14.1.13 intvalstr() integer function utilities::intvalstr (character(len=*), intent(in) *String*)

Returns the value of the leading integer numeric string.

Parameters

String	The input string

Returns

myVal: The value of the integer number as extracted from the input string

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

See also

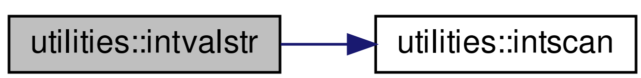
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 2726 of file utilities.F90.

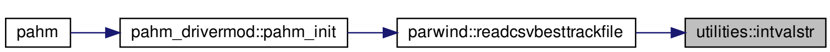
References intscan().

Referenced by parwind::readcsvbesttrackfile().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.14 loadintvar()

```
integer function utilities::loadintvar (
        integer, intent(in) nInp,
        real(sz), dimension(ninp), intent(in) vInp,
        integer, intent(in) nOut,
        integer, dimension(nout), intent(out) vOut )
```

This function loads input values into a requested model integer variable.

Parameters

nInp	Number of input values
vInp	Array of input values
nOut	Number of output values
vOut	Array of output values (integer, output)

Returns

nValsOut: Number of processed output values

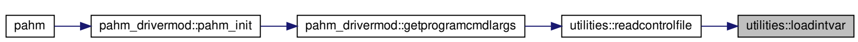
Note

Adopted from the ROMS source (Utility/inp_par.F, load_i)

Definition at line 1478 of file utilities.F90.

Referenced by readcontrolfile().

Here is the caller graph for this function:



7.14.1.15 loadlogvar()

```
integer function utilities::loadlogvar (
        integer, intent(in) nInp,
        character(len=*), dimension(ninp), intent(in) vInp,
        integer, intent(in) nOut,
        logical, dimension(nout), intent(out) vOut )
```

This function loads input values into a requested model logical variable.

Parameters

nInp	Number of input values
vInp	Array of input values
nOut	Number of output values
vOut	Array of output values (logical, output)

Returns

nValsOut: Number of processed output values

Note

Adopted from the ROMS source (Utility/inp_par.F, load_l)

Definition at line 1544 of file utilities.F90.

7.14.1.16 loadrealvar()

```
integer function utilities::loadrealvar (
        integer, intent(in) nInp,
        real(sz), dimension(ninp), intent(in) vInp,
        integer, intent(in) nOut,
        real(sz), dimension(nout), intent(out) vOut )
```

This function loads input values into a requested model real variable.

Parameters

nInp	Number of input values
vInp	Array of input values
nOut	Number of output values
vOut	Array of output values (real, output)

Returns

nValsOut: Number of processed output values

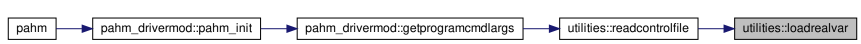
Note

Adopted from the ROMS source (Utility/inp_par.F, load_r)

Definition at line 1622 of file utilities.F90.

Referenced by readcontrolfile().

Here is the caller graph for this function:



7.14.1.17 openfileforread()

```
subroutine utilities::openfileforread (
        integer, intent(in) lun,
        character(len=*), intent(in) fileName,
        integer, intent(out) errorIO )
```

This subroutine opens an existing file for reading.

Parameters

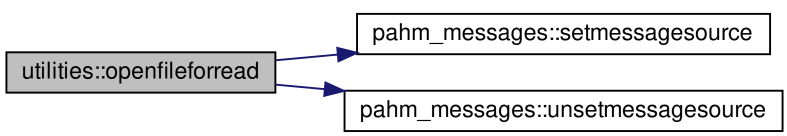
lun	The logical unit number (LUN) to use
fileName	The full pathname of the input file
errorIO	The error status, no error: status = 0 (output)

Definition at line 68 of file utilities.F90.

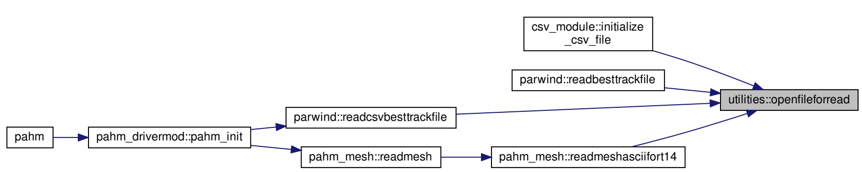
References pahm_messages::error, pahm_messages::info, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Referenced by csv_module::initialize_csv_file(), parwind::readbesttrackfile(), parwind::readcsvbesttrackfile(), and pahm_mesh::readmeshasciifort14().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.18 parseline()

```
integer function utilities::parseline (
        character(len=*), intent(in) inpLine,
        character(len=len(inpline)), intent(out) outLine,
        character(len=40), intent(inout) keyWord,
        integer, intent(inout) nVal,
        character(len=512), dimension(200), intent(inout) cVal,
        real(sz), dimension(200), intent(inout) rVal )
```

This function parses lines of text from input script/control files.

It processes each uncommented or non-blank line in the file to extract the settings for the program's variables. It is called repeatedly from ReadControlFile that sets all required program variables.

Parameters

inpLine	The input text line
outLine	The output line, left adjusted input line (output)
keyWord	The keyword to extract settings for (input/output)
nVal	The number of values provided for the keyword (input/output)
cVal	String array (cVal(nVal)) that holds the string values provided for the keyword (input/output)
rVal	Real array (rVal(nVal)) that holds the values provided for the keyword (input/output)

Returns

myStatus: The error status, no error: status = 0

Note

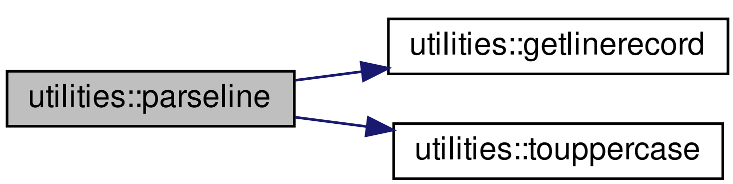
Adopted from the ROMS source (Utility/inp_par.F, decode_line)

Definition at line 869 of file utilities.F90.

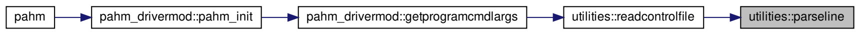
References getlinerecord(), and touppercase().

Referenced by readcontrolfile().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.19 printmodelparams() `subroutine utilities::printmodelparams`

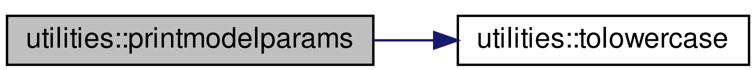
This subroutine prints on the screen the values of the program's parameters.

Definition at line 644 of file utilities.F90.

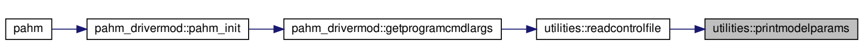
References pahm_global::backgroundatmpress, pahm_global::begdatespecified, pahm_global::begdatetime, pahm_global::begday, pahm_global::beghour, pahm_global::begmin, pahm_global::begmonth, pahm_global::begsec, pahm_global::begsimspecified, pahm_global::begsimtime, pahm_global::begyear, pahm_global::besttrackfilename, pahm_global::bladjustfac, pahm_global::enddatespecified, pahm_global::enddatetime, pahm_global::endday, pahm_global::endhour, pahm_global::endmin, pahm_global::endmonth, pahm_global::endsec, pahm_global::endsimspecified, pahm_global::endsimtime, pahm_global::endyear, pahm_global::gravity, pahm_global::mdbegsimtime, pahm_global::mdendsimtime, pahm_global::mdoutdt, pahm_global::meshfileform, pahm_global::meshfilename, pahm_global::meshfiletype, pahm_global::modeltype, pahm_global::nbtrfiles, pahm_global::ncdeflate, pahm_global::ncdlevel, pahm_global::ncshuffle, pahm_global::ncvarnam_pres, pahm_global::ncvarnam_wndx, pahm_global::ncvarnam_wndy, pahm_global::noutdt, pahm_global::outdt, pahm_global::outfilename, pahm_global::refdatespecified, pahm_global::refdatetime, pahm_global::refday, pahm_global::refhour, pahm_global::refmin, pahm_global::refmonth, pahm_global::refsec, pahm_global::refyear, pahm_global::rhoair, pahm_global::rhowater, pahm_global::title, tolowercase(), pahm_global::unittime, and pahm_global::writeparams.

Referenced by readcontrolfile().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.20 readcontrolfile() `subroutine utilities::readcontrolfile ( character(len=*), intent(in) inpFile )`

This subroutine reads the program's main control file.

Reads the control file of the program and it is repeatedly calling GetLineRecord to process each line in the file. Upon successful processing of the line, it sets the relevant program parameters and variables. This subroutine is called first as it is required by the subsequent model run.
The control file (default filename pahm_control.in) contains all required settings (user configured) required to run the program. Most of the settings have default values, in case the user hasn't supplied a value.

Parameters

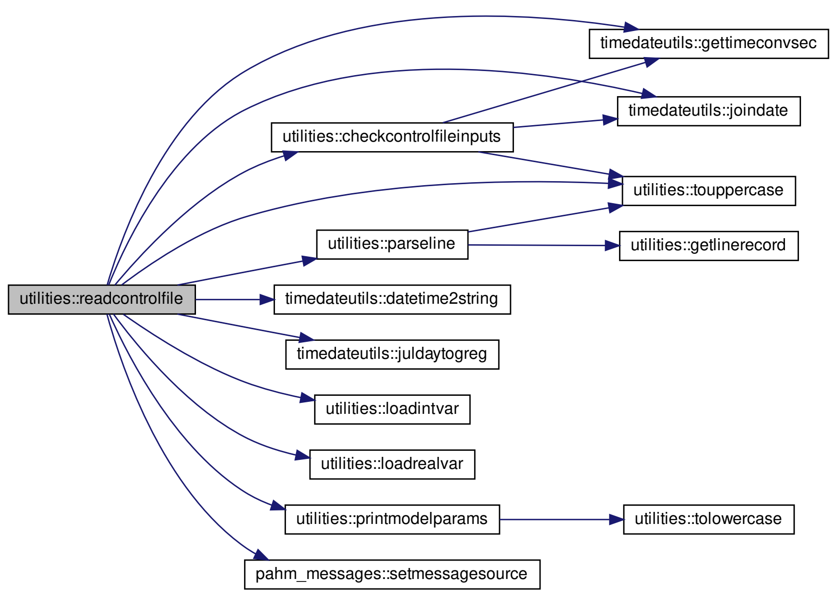
inpFile	The full pathname of the input file

Definition at line 153 of file utilities.F90.

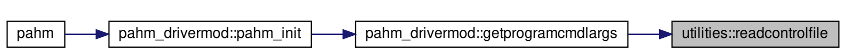
References pahm_sizes::blank, checkcontrolfileinputs(), closetol, timedateutils::datetime2string(), timedateutils::gettimeconvsec(), pahm_sizes::imissv, timedateutils::joindate(), timedateutils::juldaytogreg(), loadintvar(), loadrealvar(), pahm_global::lun_ctrl, pahm_global::lun_screen, parseline(), printmodelparams(), pahm_messages::setmessagesource(), and touppercase().

Referenced by pahm_drivermod::getprogramcmdlargs().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.21 realscan()

```
integer function utilities::realscan (
        character(len=*), intent(in) String,
        integer, intent(in) Pos,
        real(sp), intent(out) Value )
```

Scans string looking for the leading single precision real numeric string.

Scanning begins at the position specified by pos and continues to the end of the string. Leading blanks are ignored.

```
The numeric string must have the form:
[sign] d+ ['.' d*] ['e' [sign] d+]        or
[sign]     '.' d+  ['e' [sign] d+]
where sign is '+' or '-',
d* is zero or more digits,
d+ is one  or more digits,
'.' and 'e' are literal (also accept lower case 'e'),
brackets [, ] delimit optional sequences.

Value is set to the numeric value of the string.
The function value is set to the position within the string where
the numeric string ends plus one (i.e., the break character).
```

Parameters

String	The input string
Pos	The position in the input string where the scanning begins
Value	The numeric value of the string

Returns

myVal: The position within the string where the numeric string ends plus one (i.e., the break character)

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

See also

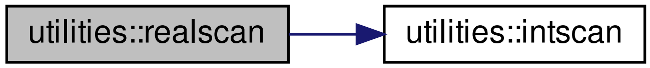
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 2788 of file utilities.F90.

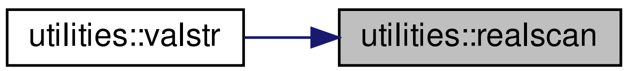
References intscan().

Referenced by valstr().

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.22 sphericaldistance_1d() `real(sz) function, dimension(:), allocatable utilities::sphericaldistance_1d (`

```
            real(sz), dimension(:), intent(in) lats,
            real(sz), dimension(:), intent(in) lons,
            real(sz), intent(in) lat0,
            real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lats	Latitude of first points - real, 1D array
lons	Longitude of first points - real, 1D array
lat0	Latitude of second point - real, scalar
lon0	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters, 1D array

Definition at line 2061 of file utilities.F90.

7.14.1.23 sphericaldistance_2d() `real(sz) function, dimension(:, :), allocatable utilities::sphericaldistance↵_2d (`

```
real(sz), dimension(:, :), intent(in) lats,
real(sz), dimension(:, :), intent(in) lons,
real(sz), intent(in) lat0,
real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

`https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas`

`https://en.wikipedia.org/wiki/Vincenty's_formulae`

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lats	Latitude of first points - real, 2D array
lons	Longitude of first points - real, 2D array
lat0	Latitude of second point - real, scalar
lon0	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters, 2D array

Definition at line 2160 of file utilities.F90.

7.14.1.24 sphericaldistance_scalar()

```
real(sz) function utilities::sphericaldistance_scalar (
        real(sz), intent(in) lat1,
        real(sz), intent(in) lon1,
        real(sz), intent(in) lat2,
        real(sz), intent(in) lon2 )
```

Calculates the distance of two points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lat1	Latitude of first point - real, scalar
lon1	Longitude of first point - real, scalar
lat2	Latitude of second point - real, scalar
lon2	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters

Definition at line 1993 of file utilities.F90.

7.14.1.25 sphericaldistanceharv()

```
real(sz) function utilities::sphericaldistanceharv (
        real(sz), intent(in) lat1,
        real(sz), intent(in) lon1,
```

```
real(sz), intent(in) lat2,
real(sz), intent(in) lon2 )
```

Calculates the distance of two points along the great circle using the Haversine formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Haversine formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Haversine_formula

van Brummelen, Glen Robert (2013). Heavenly Mathematics: The Forgotten Art of Spherical Trigonometry. Princeton University Press. ISBN 9780691148922.0691148929.

Parameters

lat1	Latitude of first point - real, scalar
lon1	Longitude of first point - real, scalar
lat2	Latitude of second point - real, scalar
lon2	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters

Definition at line 2261 of file utilities.F90.

References pahm_global::deg2rad, and pahm_global::rearth.

7.14.1.26 sphericalfracpoint()

```
subroutine utilities::sphericalfracpoint (
real(sz), intent(in) lat1,
real(sz), intent(in) lon1,
real(sz), intent(in) lat2,
real(sz), intent(in) lon2,
real(sz), intent(in) fraction,
real(sz), intent(out) latf,
real(sz), intent(out) lonf,
real(sz), intent(out), optional distf,
real(sz), intent(out), optional dist12 )
```

Calculates the coordinates of an intermediate point between two points along the great circle.

Calculates the latitude and longitude of an intermediate point at any fraction that lies between two points along their great circle path. Compute the great-circle distance using the Haversine formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

http://www.movable-type.co.uk/scripts/latlong.html

Parameters

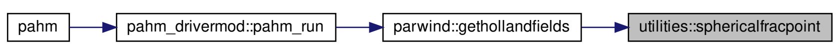
lat1	Latitude of the first point (degrees north)
lon1	Longitude of the first point (degrees east)
lat2	Latitude of the second point (degrees north)
lon2	Longitude of the second point (degrees east)
fraction	The fraction of the distance between points 1 and 2 where the intemediate point is located (0 <= fraction <= 1)
latf	The caclulated latitude of the intermidiate point (degrees north, output)
lonf	The caclulated longitude of the intermidiate point (degrees east, output)
distf	The great circle distance between the first and the intermediate point (m, output)
dist12	The great circle distance between the first and the second point (m, output)

Definition at line 2364 of file utilities.F90.

References pahm_global::deg2rad, pahm_global::rad2deg, and pahm_global::rearth.

Referenced by parwind::gethollandfields().

Here is the caller graph for this function:



7.14.1.27 tolowercase() `pure character(len(inpstring)) function utilities::tolowercase ( character(*), intent(in) inpString )`

Convert a string to lower-case.

Parameters

inpString	The input string

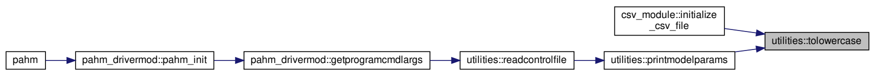
Returns

outString: The ouput string in lower case

Definition at line 1680 of file utilities.F90.

Referenced by csv_module::initialize_csv_file(), and printmodelparams().

Here is the caller graph for this function:



7.14.1.28 touppercase()

`pure character(len(inpstring)) function utilities::touppercase ( character(*), intent(in) inpString )`

Convert a string to upper-case.

Parameters

inpString	The input string

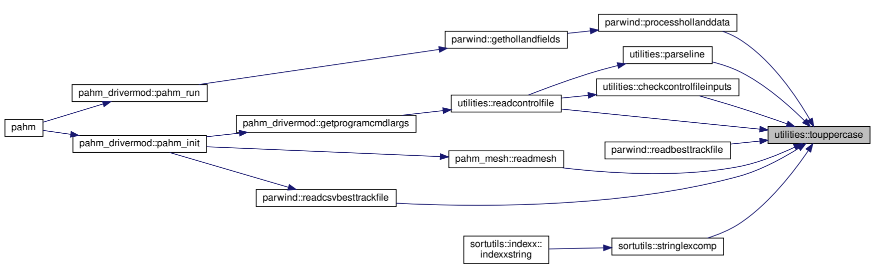
Returns

outString: The ouput string in upper case

Definition at line 1720 of file utilities.F90.

Referenced by checkcontrolfileinputs(), parseline(), parwind::processhollanddata(), parwind::readbesttrackfile(), readcontrolfile(), parwind::readcsvbesttrackfile(), pahm_mesh::readmesh(), and sortutils::stringlexcomp().

Here is the caller graph for this function:



7.14.1.29 valstr()

`real(sp) function utilities::valstr ( character(len=*), intent(in) String )`

Returns the value of the leading double precision real numeric string.

Parameters

String	The input string

Returns

myVal: The value of the double precision real number as extracted from the input string

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

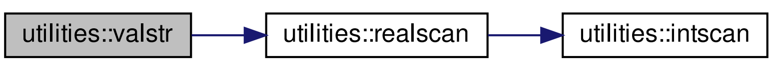
See also

`https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html`

Definition at line 2642 of file utilities.F90.

References realscan().

Here is the call graph for this function:



7.14.2 Variable Documentation

7.14.2.1 closetol `real(sz), parameter utilities::closetol = 0.001_SZ`

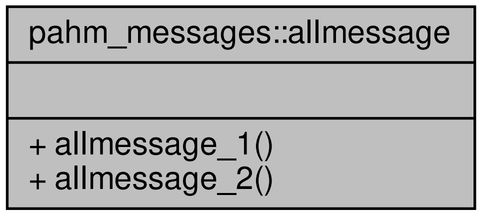
Definition at line 24 of file utilities.F90.

Referenced by checkcontrolfileinputs(), and readcontrolfile().

8 Data Type Documentation

8.1 pahm_messages::allmessage Interface Reference

Collaboration diagram for pahm_messages::allmessage:



Public Member Functions

- subroutine allmessage_1 (message)

  General purpose subroutine to write a message to both the screen and the log file.
- subroutine allmessage_2 (level, message)

8.1.1 Detailed Description

Definition at line 59 of file messages.F90.

8.1.2 Member Function/Subroutine Documentation

8.1.2.1 allmessage_1() `subroutine pahm_messages::allmessage::allmessage_1 ( character(len=*), intent(in) message )`

General purpose subroutine to write a message to both the screen and the log file.

Definition at line 309 of file messages.F90.

8.1.2.2 allmessage_2() `subroutine pahm_messages::allmessage::allmessage_2 (`
`integer, intent(in) level,`
`character(len=*), intent(in) message )`

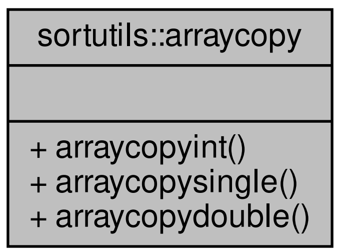
Definition at line 321 of file messages.F90.

The documentation for this interface was generated from the following file:

- messages.F90

8.2 sortutils::arraycopy Interface Reference

Collaboration diagram for sortutils::arraycopy:



Public Member Functions

- subroutine arraycopyint (src, dest, nCP, nNCP)

    Copies the 1D source integer array "src" into the 1D destination array "dest".
- subroutine arraycopysingle (src, dest, nCP, nNCP)

    Copies the 1D source single precision array "src" into the 1D destination array "dest".
- subroutine arraycopydouble (src, dest, nCP, nNCP)

    Copies the 1D source double precision array "src" into the 1D destination array "dest".

8.2.1 Detailed Description

Definition at line 38 of file sortutils.F90.

8.2.2 Member Function/Subroutine Documentation

8.2.2.1 arraycopydouble()

```
subroutine sortutils::arraycopy::arraycopydouble (
        real(hp), dimension(:), intent(in) src,
        real(hp), dimension(:), intent(out) dest,
        integer, intent(out) nCP,
        integer, intent(out) nNCP )
```

Copies the 1D source double precision array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (double precision)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1247 of file sortutils.F90.

8.2.2.2 arraycopyint()

```
subroutine sortutils::arraycopy::arraycopyint (
        integer, dimension(:), intent(in) src,
        integer, dimension(:), intent(out) dest,
        integer, intent(out) nCP,
        integer, intent(out) nNCP )
```

Copies the 1D source integer array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (integer)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1169 of file sortutils.F90.

8.2.2.3 arraycopysingle()

```
subroutine sortutils::arraycopy::arraycopysingle (
        real(sp), dimension(:), intent(in) src,
        real(sp), dimension(:), intent(out) dest,
        integer, intent(out) nCP,
        integer, intent(out) nNCP )
```

Copies the 1D source single precision array "src" into the 1D destination array "dest".

Parameters

src	The one-dimensional array to be copied (single precision)
dest	The copied array (output)
nCP	The number of elements of "src" array that copied (output)
nNCP	The number of elements of "src" array that failed to be copied (output)

Note

Adopted from Numerical Recipes for Fortran 90

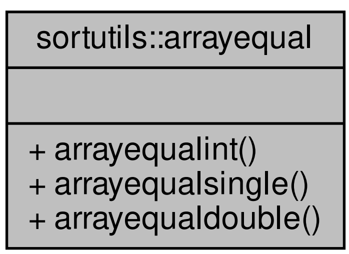
Definition at line 1208 of file sortutils.F90.

The documentation for this interface was generated from the following file:

- sortutils.F90

8.3 sortutils::arrayequal Interface Reference

Collaboration diagram for sortutils::arrayequal:



Public Member Functions

- logical function arrayequalint (arr1, arr2)

  Compares two one-dimensional integer arrays for equality.
- logical function arrayequalsingle (arr1, arr2)

  Compares two one-dimensional single precision arrays for equality.
- logical function arrayequaldouble (arr1, arr2)

  Compares two one-dimensional double precision arrays for equality.

8.3.1 Detailed Description

Definition at line 44 of file sortutils.F90.

8.3.2 Member Function/Subroutine Documentation

8.3.2.1 arrayequaldouble()

```
logical function sortutils::arrayequal::arrayequaldouble (
        real(hp), dimension(:), intent(in) arr1,
        real(hp), dimension(:), intent(in) arr2 )
```

Compares two one-dimensional double precision arrays for equality.

The equality is determined using a tolerance of: 0.00000001, such that the two arrays are considered to be essentially equal on double precision calculations.

Parameters

arr1	The first array in the comparison (double precision)
arr2	The second array in the comparison (double precision)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

Definition at line 1384 of file sortutils.F90.

8.3.2.2 arrayequalint()

```
logical function sortutils::arrayequal::arrayequalint (
        integer, dimension(:), intent(in) arr1,
        integer, dimension(:), intent(in) arr2 )
```

Compares two one-dimensional integer arrays for equality.

Parameters

arr1	The first array in the comparison (integer)
arr2	The second array in the comparison (integer)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

Definition at line 1284 of file sortutils.F90.

8.3.2.3 arrayequalsingle()

```
logical function sortutils::arrayequal::arrayequalsingle (
    real(sp), dimension(:), intent(in) arr1,
    real(sp), dimension(:), intent(in) arr2 )
```

Compares two one-dimensional single precision arrays for equality.

The equality is determined using a tolerance of: 0.00000001, such that the two arrays are considered to be essentially equal on single precision calculations.

Parameters

arr1	The first array in the comparison (single precision)
arr2	The second array in the comparison (single precision)

Returns

myValOut: The value of the comparison (logical). TRUE if all the elements of arr1 are equal to all elements of arr2, FALSE otherwise.

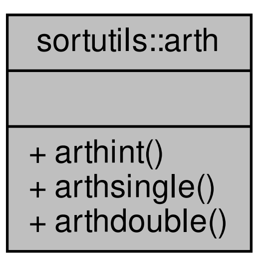
Definition at line 1329 of file sortutils.F90.

The documentation for this interface was generated from the following file:

- sortutils.F90

8.4 sortutils::arth Interface Reference

Collaboration diagram for sortutils::arth:



Public Member Functions

- pure integer function, dimension(n) arthint (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(sp) function, dimension(n) arthsingle (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(hp) function, dimension(n) arthdouble (first, increment, n)

    Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

8.4.1 Detailed Description

Definition at line 32 of file sortutils.F90.

8.4.2 Member Function/Subroutine Documentation

8.4.2.1 arthdouble()

```
pure real(hp) function, dimension(n) sortutils::arth::arthdouble (
        real(hp), intent(in) first,
        real(hp), intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (double precision)
increment	The value of the increment (double precision)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (double precision)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1952 of file sortutils.F90.

8.4.2.2 arthint()

```
pure integer function, dimension(n) sortutils::arth::arthint (
        integer, intent(in) first,
        integer, intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (integer)
increment	The value of the increment (integer)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (integer)

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1818 of file sortutils.F90.

8.4.2.3 arthsingle()

```
pure real(sp) function, dimension(n) sortutils::arth::arthsingle (
        real(sp), intent(in) first,
        real(sp), intent(in) increment,
        integer, intent(in) n )
```

Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

Parameters

first	The value of the first term (single precision)
increment	The value of the increment (single precision)
n	The total number of terms in the return 1D array (integer)

Returns

arthOut: The 1D array that contains the arithmetic progression (single precision)

Note

Adopted from Numerical Recipes for Fortran 90

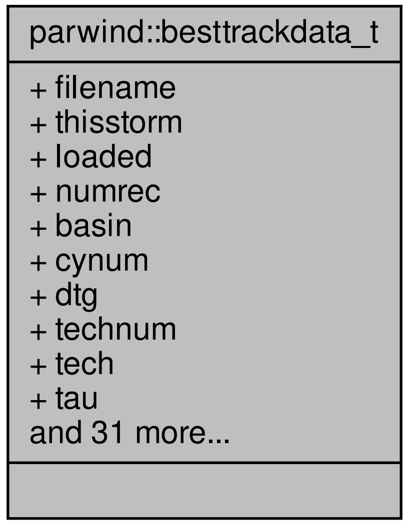
Definition at line 1885 of file sortutils.F90.

The documentation for this interface was generated from the following file:

- sortutils.F90

8.5 parwind::besttrackdata_t Type Reference

Collaboration diagram for parwind::besttrackdata_t:



Public Attributes

- character(len=fnamelen) filename
- character(len=10) thisstorm
- logical loaded = .FALSE.
- integer numrec
- character(len=2), dimension(:), allocatable basin
- integer, dimension(:), allocatable cynum
- character(len=10), dimension(:), allocatable dtg
- integer, dimension(:), allocatable technum
- character(len=4), dimension(:), allocatable tech
- integer, dimension(:), allocatable tau
- integer, dimension(:), allocatable intlat
- integer, dimension(:), allocatable intlon
- character(len=1), dimension(:), allocatable ew
- character(len=1), dimension(:), allocatable ns
- integer, dimension(:), allocatable intvmax
- integer, dimension(:), allocatable intmslp
- character(len=2), dimension(:), allocatable ty
- integer, dimension(:), allocatable rad
- character(len=3), dimension(:), allocatable windcode
- integer, dimension(:), allocatable intrad1

- integer, dimension(:), allocatable intrad2
- integer, dimension(:), allocatable intrad3
- integer, dimension(:), allocatable intrad4
- integer, dimension(:), allocatable intpouter
- integer, dimension(:), allocatable introuter
- integer, dimension(:), allocatable intrmw
- integer, dimension(:), allocatable gusts
- integer, dimension(:), allocatable eye
- character(len=3), dimension(:), allocatable subregion
- integer, dimension(:), allocatable maxseas
- character(len=3), dimension(:), allocatable initials
- integer, dimension(:), allocatable dir
- integer, dimension(:), allocatable intspeed
- character(len=10), dimension(:), allocatable stormname
- integer, dimension(:), allocatable cyclenum
- integer, dimension(:), allocatable year
- integer, dimension(:), allocatable month
- integer, dimension(:), allocatable day
- integer, dimension(:), allocatable hour
- real(sz), dimension(:), allocatable lat
- real(sz), dimension(:), allocatable lon

8.5.1 Detailed Description

Definition at line 26 of file parwind-orig.F90.

8.5.2 Member Data Documentation

8.5.2.1 basin `character(len=2), dimension(:), allocatable parwind::besttrackdata_t::basin`

Definition at line 33 of file parwind-orig.F90.

8.5.2.2 cyclenum `integer, dimension(:), allocatable parwind::besttrackdata_t::cyclenum`

Definition at line 101 of file parwind-orig.F90.

8.5.2.3 cynum `integer, dimension(:), allocatable parwind::besttrackdata_t::cynum`

Definition at line 34 of file parwind-orig.F90.

8.5.2.4 day `integer, dimension(:), allocatable parwind::besttrackdata_t::day`

Definition at line 104 of file parwind-orig.F90.

8.5.2.5 dir `integer, dimension(:), allocatable parwind::besttrackdata_t::dir`

Definition at line 96 of file parwind-orig.F90.

8.5.2.6 dtg `character(len=10), dimension(:), allocatable parwind::besttrackdata_t::dtg`

Definition at line 35 of file parwind-orig.F90.

8.5.2.7 ew `character(len=1), dimension(:), allocatable parwind::besttrackdata_t::ew`

Definition at line 43 of file parwind-orig.F90.

8.5.2.8 eye `integer, dimension(:), allocatable parwind::besttrackdata_t::eye`

Definition at line 83 of file parwind-orig.F90.

8.5.2.9 filename `character(len=fnamelen) parwind::besttrackdata_t::filename`

Definition at line 27 of file parwind-orig.F90.

8.5.2.10 gusts `integer, dimension(:), allocatable parwind::besttrackdata_t::gusts`

Definition at line 82 of file parwind-orig.F90.

8.5.2.11 hour `integer, dimension(:), allocatable parwind::besttrackdata_t::hour`

Definition at line 104 of file parwind-orig.F90.

8.5.2.12 initials `character(len=3), dimension(:), allocatable parwind::besttrackdata_t::initials`

Definition at line 95 of file parwind-orig.F90.

8.5.2.13 intlat `integer, dimension(:), allocatable parwind::besttrackdata_t::intlat`

Definition at line 41 of file parwind-orig.F90.

8.5.2.14 intlon `integer, dimension(:), allocatable parwind::besttrackdata_t::intlon`

Definition at line 42 of file parwind-orig.F90.

8.5.2.15 intmslp `integer, dimension(:), allocatable parwind::besttrackdata_t::intmslp`

Definition at line 47 of file parwind-orig.F90.

8.5.2.16 intpouter `integer, dimension(:), allocatable parwind::besttrackdata_t::intpouter`

Definition at line 79 of file parwind-orig.F90.

8.5.2.17 intrad1 `integer, dimension(:), allocatable parwind::besttrackdata_t::intrad1`

Definition at line 71 of file parwind-orig.F90.

8.5.2.18 intrad2 `integer, dimension(:), allocatable parwind::besttrackdata_t::intrad2`

Definition at line 73 of file parwind-orig.F90.

8.5.2.19 intrad3 `integer, dimension(:), allocatable parwind::besttrackdata_t::intrad3`

Definition at line 75 of file parwind-orig.F90.

8.5.2.20 intrad4 `integer, dimension(:), allocatable parwind::besttrackdata_t::intrad4`

Definition at line 77 of file parwind-orig.F90.

8.5.2.21 intrmw `integer, dimension(:), allocatable parwind::besttrackdata_t::intrmw`

Definition at line 81 of file parwind-orig.F90.

8.5.2.22 introuter `integer, dimension(:), allocatable parwind::besttrackdata_t::introuter`

Definition at line 80 of file parwind-orig.F90.

8.5.2.23 intspeed `integer, dimension(:), allocatable parwind::besttrackdata_t::intspeed`

Definition at line 97 of file parwind-orig.F90.

8.5.2.24 intvmax `integer, dimension(:), allocatable parwind::besttrackdata_t::intvmax`

Definition at line 46 of file parwind-orig.F90.

8.5.2.25 lat `real(sz), dimension(:), allocatable parwind::besttrackdata_t::lat`

Definition at line 105 of file parwind-orig.F90.

8.5.2.26 loaded `logical parwind::besttrackdata_t::loaded = .FALSE.`

Definition at line 29 of file parwind-orig.F90.

8.5.2.27 lon `real(sz), dimension(:), allocatable parwind::besttrackdata_t::lon`

Definition at line 105 of file parwind-orig.F90.

8.5.2.28 maxseas `integer, dimension(:), allocatable parwind::besttrackdata_t::maxseas`

Definition at line 94 of file parwind-orig.F90.

8.5.2.29 month `integer, dimension(:), allocatable parwind::besttrackdata_t::month`

Definition at line 104 of file parwind-orig.F90.

8.5.2.30 ns `character(len=1), dimension(:), allocatable parwind::besttrackdata_t::ns`

Definition at line 44 of file parwind-orig.F90.

8.5.2.31 numrec `integer parwind::besttrackdata_t::numrec`

Definition at line 30 of file parwind-orig.F90.

8.5.2.32 rad `integer, dimension(:), allocatable parwind::besttrackdata_t::rad`

Definition at line 67 of file parwind-orig.F90.

8.5.2.33 stormname `character(len=10), dimension(:), allocatable parwind::besttrackdata_t::stormname`

Definition at line 98 of file parwind-orig.F90.

8.5.2.34 subregion `character(len=3), dimension(:), allocatable parwind::besttrackdata_t::subregion`

Definition at line 84 of file parwind-orig.F90.

8.5.2.35 tau `integer, dimension(:), allocatable parwind::besttrackdata_t::tau`

Definition at line 39 of file parwind-orig.F90.

8.5.2.36 tech `character(len=4), dimension(:), allocatable parwind::besttrackdata_t::tech`

Definition at line 37 of file parwind-orig.F90.

8.5.2.37 technum `integer, dimension(:), allocatable parwind::besttrackdata_t::technum`

Definition at line 36 of file parwind-orig.F90.

8.5.2.38 thisstorm `character(len=10) parwind::besttrackdata_t::thisstorm`

Definition at line 28 of file parwind-orig.F90.

8.5.2.39 ty `character(len=2), dimension(:), allocatable parwind::besttrackdata_t::ty`

Definition at line 48 of file parwind-orig.F90.

8.5.2.40 windcode `character(len=3), dimension(:), allocatable parwind::besttrackdata_t::windcode`

Definition at line 68 of file parwind-orig.F90.

8.5.2.41 year `integer, dimension(:), allocatable parwind::besttrackdata_t::year`

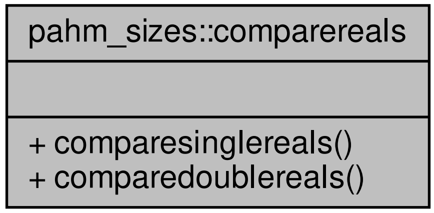
Definition at line 104 of file parwind-orig.F90.

The documentation for this type was generated from the following files:

- parwind-orig.F90
- parwind.F90

8.6 pahm_sizes::comparereals Interface Reference

Collaboration diagram for pahm_sizes::comparereals:



Public Member Functions

- integer function comparesinglereals (rVal1, rVal2, eps)

  Compares two single precision numbers.
- integer function comparedoublereals (rVal1, rVal2, eps)

  Compares two double precision numbers.

8.6.1 Detailed Description

Definition at line 22 of file sizes.F90.

8.6.2 Member Function/Subroutine Documentation

8.6.2.1 comparedoublereals()

```
integer function pahm_sizes::comparereals::comparedoublereals (
        real(hp), intent(in) rVal1,
        real(hp), intent(in) rVal2,
        real(hp), intent(in), optional eps )
```

Compares two double precision numbers.

Allow users to define the value of eps. If not, eps equals to the default machine eps.

Parameters

rVal1	The first value (double precision number) in the comparison
rVal2	The second value (double precision number) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut

```
-1 (if rVal1 < rVal2)
 0 (if rVal1 = rVal2)
+1 (if rVal1 > rVal2)
```

Note

The code was adopted from the D-Flow FM source (...src/precision_basics.F90)

Definition at line 101 of file sizes.F90.

8.6.2.2 comparesinglereals() `integer function pahm_sizes::comparereals::comparesinglereals ( real(sp), intent(in) rVal1, real(sp), intent(in) rVal2, real(sp), intent(in), optional eps )`

Compares two single precision numbers.

Allow users to define the value of eps. If not, eps equals to the default machine eps.

Parameters

rVal1	The first value (single precision number) in the comparison
rVal2	The second value (single precision number) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut

```
-1 (if rVal1 < rVal2)
 0 (if rVal1 = rVal2)
+1 (if rVal1 > rVal2)
```

Note

The code was adopted from the D-Flow FM source (...src/precision_basics.F90)

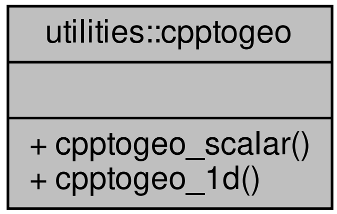
Definition at line 168 of file sizes.F90.

The documentation for this interface was generated from the following file:

- sizes.F90

8.7 utilities::cpptogeo Interface Reference

Collaboration diagram for utilities::cpptogeo:



Public Member Functions

- subroutine cpptogeo_scalar (x, y, lat0, lon0, lat, lon)

  Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- subroutine cpptogeo_1d (x, y, lat0, lon0, lat, lon)

  Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

8.7.1 Detailed Description

Definition at line 34 of file utilities.F90.

8.7.2 Member Function/Subroutine Documentation

8.7.2.1 cpptogeo_1d()

```
subroutine utilities::cpptogeo::cpptogeo_1d (
        real(sz), dimension(:), intent(in) x,
        real(sz), dimension(:), intent(in) y,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), dimension(:), intent(out) lat,
        real(sz), dimension(:), intent(out) lon )
```

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

Transforms 1D CPP coordinates into 1D geographical coordinates. This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

x	X coordinate: x (m) - real, 1D array
y	Y coordinate: y (m) - real, 1D array
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
lat	Latitude (degrees north) - real, 1D array (output)
lon	Longitude (degrees east) - real, 1D array (output)

Definition at line 1940 of file utilities.F90.

References pahm_global::deg2rad, and pahm_global::rearth.

8.7.2.2 cpptogeo_scalar()

```
subroutine utilities::cpptogeo::cpptogeo_scalar (
        real(sz), intent(in) x,
        real(sz), intent(in) y,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), intent(out) lat,
        real(sz), intent(out) lon )
```

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

x	X coordinate: x (m) - real, scalar
y	Y coordinate: y (m) - real, scalar
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
lat	Latitude (degrees north) - real, scalar (output)
lon	Longitude (degrees east) - real, scalar (output)

Definition at line 1893 of file utilities.F90.

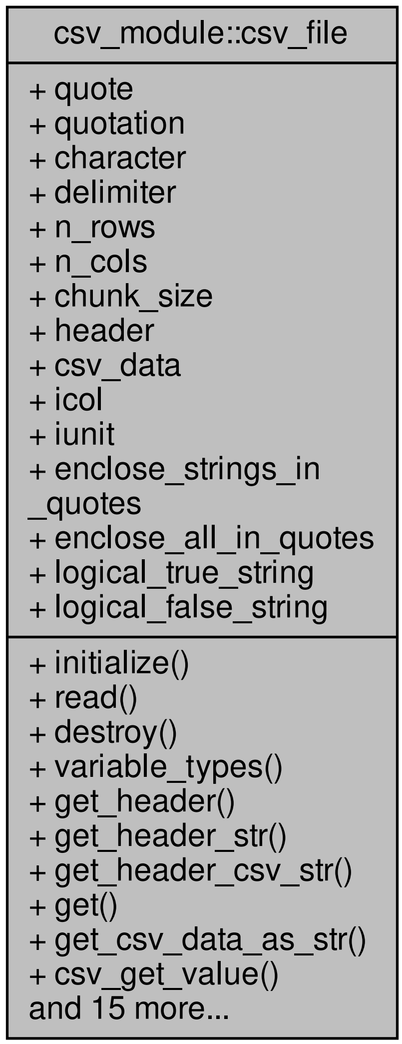
References pahm_global::deg2rad, and pahm_global::rearth.

The documentation for this interface was generated from the following file:

- utilities.F90

8.8 csv_module::csv_file Type Reference

Collaboration diagram for csv_module::csv_file:



Public Member Functions

- procedure, public initialize => initialize_csv_file
- procedure, public read => read_csv_file
- procedure, public destroy => destroy_csv_file
- procedure, public variable_types
- generic, public get_header => get_header_str, get_header_csv_str

- procedure get_header_str
- procedure get_header_csv_str
- generic, public get => get_csv_data_as_str, csv_get_value, get_real_column, get_integer_column, get_logical_column, get_character_column, get_csv_string_column
- procedure get_csv_data_as_str
- procedure csv_get_value
- procedure get_real_column
- procedure get_integer_column
- procedure get_logical_column
- procedure get_character_column
- procedure get_csv_string_column
- procedure, public open => open_csv_file
- generic, public add => add_cell, add_vector, add_matrix
- procedure add_cell
- procedure add_vector
- procedure add_matrix
- procedure, public next_row
- procedure, public close => close_csv_file
- procedure tokenize => tokenize_csv_line
- procedure read_line_from_file
- procedure get_column

Public Attributes

- character(len=1) quote = '"'

  the main class for reading and writing CSV files.
- character(len=1) quotation
- character(len=1) character
- character(len=1) delimiter = ','
- integer, public n_rows = 0
- integer, public n_cols = 0
- integer chunk_size = 100
- type(csv_string), dimension(:), allocatable header
- type(csv_string), dimension(:,:), allocatable csv_data
- integer icol = 0
- integer iunit = LUN_BTRK
- logical enclose_strings_in_quotes = .true.
- logical enclose_all_in_quotes = .false.
- character(len=1) logical_true_string = 'T'
- character(len=1) logical_false_string = 'F'

8.8.1 Detailed Description

Definition at line 45 of file csv_module.F90.

8.8.2 Member Function/Subroutine Documentation

8.8.2.1 add() `generic, public csv_module::csv_file::add`

Definition at line 116 of file csv_module.F90.

8.8.2.2 add_cell() `procedure csv_module::csv_file::add_cell`

Definition at line 119 of file csv_module.F90.

8.8.2.3 add_matrix() `procedure csv_module::csv_file::add_matrix`

Definition at line 121 of file csv_module.F90.

8.8.2.4 add_vector() `procedure csv_module::csv_file::add_vector`

Definition at line 120 of file csv_module.F90.

8.8.2.5 close() `procedure, public csv_module::csv_file::close`

Definition at line 124 of file csv_module.F90.

8.8.2.6 csv_get_value() `procedure csv_module::csv_file::csv_get_value`

Definition at line 107 of file csv_module.F90.

8.8.2.7 destroy() `procedure, public csv_module::csv_file::destroy`

Definition at line 87 of file csv_module.F90.

8.8.2.8 get() `generic, public csv_module::csv_file::get`

Definition at line 99 of file csv_module.F90.

8.8.2.9 get_character_column() `procedure csv_module::csv_file::get_character_column`

Definition at line 111 of file csv_module.F90.

8.8.2.10 get_column() `procedure csv_module::csv_file::get_column`

Definition at line 128 of file csv_module.F90.

8.8.2.11 get_csv_data_as_str() `procedure csv_module::csv_file::get_csv_data_as_str`

Definition at line 106 of file csv_module.F90.

8.8.2.12 get_csv_string_column() `procedure csv_module::csv_file::get_csv_string_column`

Definition at line 112 of file csv_module.F90.

8.8.2.13 get_header() `generic, public csv_module::csv_file::get_header`

Definition at line 91 of file csv_module.F90.

8.8.2.14 get_header_csv_str() `procedure csv_module::csv_file::get_header_csv_str`

Definition at line 94 of file csv_module.F90.

8.8.2.15 get_header_str() `procedure csv_module::csv_file::get_header_str`

Definition at line 93 of file csv_module.F90.

8.8.2.16 get_integer_column() `procedure csv_module::csv_file::get_integer_column`

Definition at line 109 of file csv_module.F90.

8.8.2.17 get_logical_column() `procedure csv_module::csv_file::get_logical_column`

Definition at line 110 of file csv_module.F90.

8.8.2.18 get_real_column() `procedure csv_module::csv_file::get_real_column`

Definition at line 108 of file csv_module.F90.

8.8.2.19 initialize() `procedure, public csv_module::csv_file::initialize`

Definition at line 85 of file csv_module.F90.

8.8.2.20 next_row() `procedure, public csv_module::csv_file::next_row`

Definition at line 123 of file csv_module.F90.

8.8.2.21 open() `procedure, public csv_module::csv_file::open`

Definition at line 114 of file csv_module.F90.

8.8.2.22 read() `procedure, public csv_module::csv_file::read`

Definition at line 86 of file csv_module.F90.

8.8.2.23 read_line_from_file() `procedure csv_module::csv_file::read_line_from_file`

Definition at line 127 of file csv_module.F90.

8.8.2.24 tokenize() `procedure csv_module::csv_file::tokenize`

Definition at line 126 of file csv_module.F90.

8.8.2.25 variable_types() `procedure, public csv_module::csv_file::variable_types`

Definition at line 89 of file csv_module.F90.

8.8.3 Member Data Documentation

8.8.3.1 character `csv_module::csv_file::character`

Definition at line 55 of file csv_module.F90.

8.8.3.2 chunk_size `integer csv_module::csv_file::chunk_size = 100`

Definition at line 61 of file csv_module.F90.

8.8.3.3 csv_data `type(csv_string), dimension(:,:), allocatable csv_module::csv_file::csv_data`

Definition at line 63 of file csv_module.F90.

8.8.3.4 delimiter `character(len=1) csv_module::csv_file::delimiter = ','`

Definition at line 56 of file csv_module.F90.

8.8.3.5 enclose_all_in_quotes `logical csv_module::csv_file::enclose_all_in_quotes = .false.`

Definition at line 70 of file csv_module.F90.

8.8.3.6 enclose_strings_in_quotes `logical csv_module::csv_file::enclose_strings_in_quotes = .true.`

Definition at line 68 of file csv_module.F90.

8.8.3.7 header `type(csv_string), dimension(:), allocatable csv_module::csv_file::header`

Definition at line 62 of file csv_module.F90.

8.8.3.8 icol `integer csv_module::csv_file::icol = 0`

Definition at line 66 of file csv_module.F90.

8.8.3.9 iunit `integer csv_module::csv_file::iunit = LUN_BTRK`

Definition at line 67 of file csv_module.F90.

8.8.3.10 logical_false_string `character(len=1) csv_module::csv_file::logical_false_string = 'F'`

Definition at line 76 of file csv_module.F90.

8.8.3.11 logical_true_string `character(len=1) csv_module::csv_file::logical_true_string = 'T'`

Definition at line 72 of file csv_module.F90.

8.8.3.12 n_cols `integer, public csv_module::csv_file::n_cols = 0`

Definition at line 60 of file csv_module.F90.

8.8.3.13 n_rows `integer, public csv_module::csv_file::n_rows = 0`

Definition at line 59 of file csv_module.F90.

8.8.3.14 quotation `character(len=1) csv_module::csv_file::quotation`

Definition at line 55 of file csv_module.F90.

8.8.3.15 quote `character(len=1) csv_module::csv_file::quote = '"'`

the main class for reading and writing CSV files.

Note

A CSV file is assumed to contain the same number of columns in each row. It may optionally contain a header row.

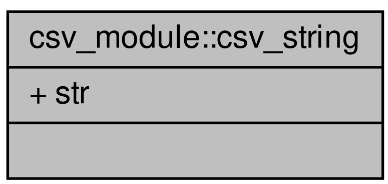
Definition at line 55 of file csv_module.F90.

The documentation for this type was generated from the following file:

- csv_module.F90

8.9 csv_module::csv_string Type Reference

Collaboration diagram for csv_module::csv_string:



Public Attributes

- character(len=:), allocatable str
  a cell from a CSV file.

8.9.1 Detailed Description

Definition at line 37 of file csv_module.F90.

8.9.2 Member Data Documentation

8.9.2.1 str `character(len=:), allocatable csv_module::csv_string::str`

a cell from a CSV file.

This is used to store the data internally in the [[csv_file]] class.

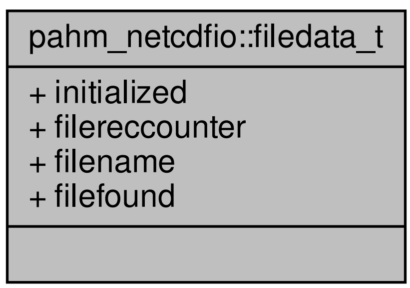
Definition at line 42 of file csv_module.F90.

The documentation for this type was generated from the following file:

- csv_module.F90

8.10 pahm_netcdfio::filedata_t Type Reference

Collaboration diagram for pahm_netcdfio::filedata_t:



Public Attributes

- logical initialized = .FALSE.
- integer filereccounter = 0
- character(len=fnamelen) filename
- logical filefound = .FALSE.

8.10.1 Detailed Description

Definition at line 31 of file netcdfio-nems.F90.

8.10.2 Member Data Documentation

8.10.2.1 filefound `logical pahm_netcdfio::filedata_t::filefound = .FALSE.`

Definition at line 35 of file netcdfio-nems.F90.

8.10.2.2 filename `character(len=fnamelen) pahm_netcdfio::filedata_t::filename`

Definition at line 34 of file netcdfio-nems.F90.

8.10.2.3 filereccounter `integer pahm_netcdfio::filedata_t::filereccounter = 0`

Definition at line 33 of file netcdfio-nems.F90.

8.10.2.4 initialized `logical pahm_netcdfio::filedata_t::initialized = .FALSE.`

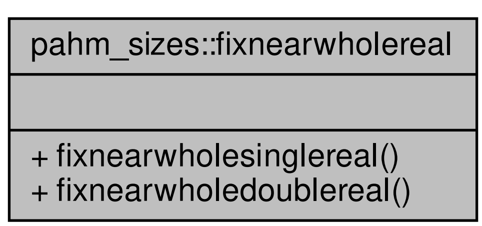
Definition at line 32 of file netcdfio-nems.F90.

The documentation for this type was generated from the following files:

- netcdfio-nems.F90
- netcdfio-orig.F90
- netcdfio.F90

8.11 pahm_sizes::fixnearwholereal Interface Reference

Collaboration diagram for pahm_sizes::fixnearwholereal:



Public Member Functions

- real(sp) function fixnearwholesinglereal (rVal, eps)

  Rounds a single precision real number to its nearest whole number.
- real(hp) function fixnearwholedoublereal (rVal, eps)

  Rounds a double precision real number to its nearest whole number.

8.11.1 Detailed Description

Definition at line 27 of file sizes.F90.

8.11.2 Member Function/Subroutine Documentation

8.11.2.1 fixnearwholedoublereal() `real(hp) function pahm_sizes::fixnearwholereal::fixnearwholedoublereal (`

```
real(hp), intent(in) rVal,
real(hp), intent(in), optional eps )
```

Rounds a double precision real number to its nearest whole number.

Rounds a double precision real number to its nearest whole number. If the real number is very close (within a tolerance) to its nearest whole number then it is set equal to its nearest whole number. Allow users to define the value of the tolerance "eps". If not, then eps equals to the default machine eps.

Parameters

rVal	The real number value (double precision) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut : Either **rVal** or its nearest integer **iVar** converted to double

```
rVal (if abs(rVal - iVal) >  eps
iVal (if abs(rVal - iVal) <= eps
```

Definition at line 235 of file sizes.F90.

8.11.2.2 fixnearwholesinglereal() `real(sp) function pahm_sizes::fixnearwholereal::fixnearwholesinglereal (`

```
real(sp), intent(in) rVal,
real(sp), intent(in), optional eps )
```

Rounds a single precision real number to its nearest whole number.

Rounds a single precision real number to its nearest whole number. If the real number is very close (within a tolerance) to its nearest whole number then it is set equal to its nearest whole number. Allow users to define the value of the tolerance "eps". If not, then eps equals to the default machine eps.

Parameters

rVal	The real number value (single precision) in the comparison
eps	The tolerance (optional) for the comparison

Returns

myValOut : Either **rVal** or its nearest integer **iVar** converted to real

```
rVal (if abs(rVal - iVal) >  eps
iVal (if abs(rVal - iVal) <= eps
```

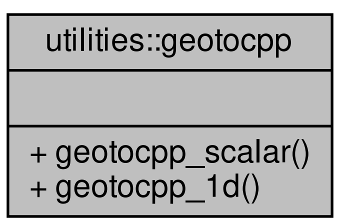
Definition at line 291 of file sizes.F90.

The documentation for this interface was generated from the following file:

- sizes.F90

8.12 utilities::geotocpp Interface Reference

Collaboration diagram for utilities::geotocpp:



Public Member Functions

- subroutine geotocpp_scalar (lat, lon, lat0, lon0, x, y)

  Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.
- subroutine geotocpp_1d (lat, lon, lat0, lon0, x, y)

  Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

8.12.1 Detailed Description

Definition at line 29 of file utilities.F90.

8.12.2 Member Function/Subroutine Documentation

8.12.2.1 geotocpp_1d()

```
subroutine utilities::geotocpp::geotocpp_1d (
        real(sz), dimension(:), intent(in) lat,
        real(sz), dimension(:), intent(in) lon,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), dimension(:), intent(out) x,
        real(sz), dimension(:), intent(out) y )
```

Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

Transforms 1D geographical coordinates into 1D CPP coordinates. This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

lat	Latitude (degrees north) - real, 1D array
lon	Longitude (degrees east) - real, 1D array
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
x	Calculated X coordinate: x (m) - real, 1D array (output)
y	Calculated Y coordinate: y (m) - real, 1D array (output)

Definition at line 1847 of file utilities.F90.

References pahm_global::deg2rad, and pahm_global::rearth.

8.12.2.2 geotocpp_scalar()

```
subroutine utilities::geotocpp::geotocpp_scalar (
        real(sz), intent(in) lat,
        real(sz), intent(in) lon,
        real(sz), intent(in) lat0,
        real(sz), intent(in) lon0,
        real(sz), intent(out) x,
        real(sz), intent(out) y )
```

Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

lat	Latitude (degrees north) - real, scalar
lon	Longitude (degrees east) - real, scalar
lat0	Latitude of projection origin (degrees north) - real, scalar
lon0	Longitude of projection origin (degrees east) - real, scalar
x	Calculated X coordinate: x (m) - real, scalar (output)
y	Calculated Y coordinate: y (m) - real, scalar (output)

Definition at line 1800 of file utilities.F90.

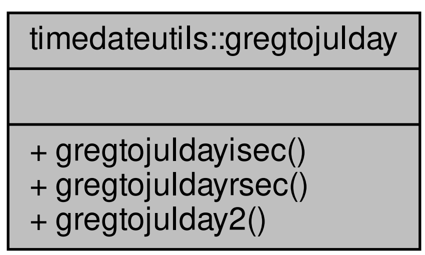
References pahm_global::deg2rad, and pahm_global::rearth.

The documentation for this interface was generated from the following file:

- utilities.F90

8.13 timedateutils::gregtojulday Interface Reference

Collaboration diagram for timedateutils::gregtojulday:



Public Member Functions

- real(sz) function gregtojuldayisec (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

    Determines the Julian date from a Gregorian date.
- real(sz) function gregtojuldayrsec (iYear, iMonth, iDay, iHour, iMin, rSec, mJD)

    Determines the Julian date from a Gregorian date.
- real(sz) function gregtojulday2 (iDate, iTime, mJD)

    Determines the Julian date from a Gregorian date.

8.13.1 Detailed Description

Definition at line 31 of file timedateutils.F90.

8.13.2 Member Function/Subroutine Documentation

8.13.2.1 gregtojulday2()

```
real(sz) function timedateutils::gregtojulday::gregtojulday2 (
        integer, intent(in) iDate,
        integer, intent(in) iTime,
        integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian).
It is usefull to compute differences between dates.
Similar to GregToJulDayISEC but the seconds number is real to allow for second fractions.

Parameters

iDate	The date as YYYYMMDD (integer)
	`YYYY      The year (YYYY, integer, 1582 <= YYYY)`
	`  MM      The month of the year (MM, integer, 1 <= MM <=12)`
	`  DD      The day of the month (DD, integer, 1 <= DD <=31)`

iTime	The time as hhmmss (integer)
	`  hh      The hour of the day (integer, 0 <= hh <= 23)`
	`  mm      The minute of the hour (integer, 0 <= mm <= 59)`
	`  ss      The second of the minute (integer, 0 <= ss <= 60)`

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use:                          mJD  < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

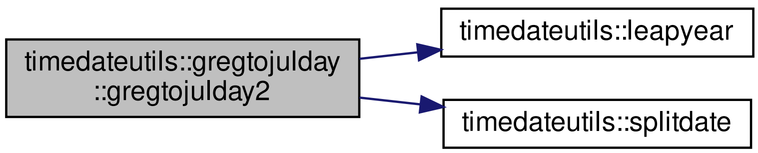
Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 776 of file timedateutils.F90.

References timedateutils::firstgregdate, timedateutils::leapyear(), timedateutils::mdjoffset, timedateutils::splitdate(), and timedateutils::usemodjulday.

Here is the call graph for this function:



8.13.2.2 gregtojuldayisec() `real(sz) function timedateutils::gregtojulday::gregtojuldayisec (`

```
integer, intent(in) iYear,
integer, intent(in) iMonth,
integer, intent(in) iDay,
integer, intent(in) iHour,
integer, intent(in) iMin,
integer, intent(in) iSec,
integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is usefull to compute differences between dates.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <=12)
iDay	The day of the month (DD, integer, 1 <= DD <=31)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59)
iSec	iSec The second of the minute (ss, integer, 0 <= ss <= 59)

Parameters

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use: mJD < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

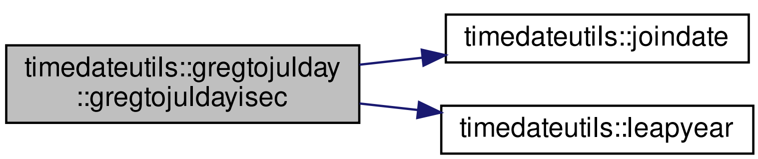
Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 536 of file timedateutils.F90.

References timedateutils::firstgregdate, pahm_sizes::hp, timedateutils::joindate(), timedateutils::leapyear(), timedateutils::mdjoffset, pahm_sizes::rmissv, and timedateutils::usemodjulday.

Here is the call graph for this function:



8.13.2.3 gregtojuldayrsec()

```
real(sz) function timedateutils::gregtojulday::gregtojuldayrsec (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay,
        integer, intent(in) iHour,
        integer, intent(in) iMin,
        real(sz), intent(in) rSec,
        integer, intent(in), optional mJD )
```

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.
The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian).
It is usefull to compute differences between dates.
Similar to GregToJulDayISEC but the seconds number is real to allow for second fractions.

Parameters

iYear	The year (YYYY, integer, 1582 <= YYYY)
iMonth	The month of the year (MM, integer, 1 <= MM <=12)
iDay	The day of the month (DD, integer, 1 <= DD <=31)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59)
rSec	The second of the minute (ss, real, 0 <= ss <= 59)

mJD	Flag to use a modified julian day number or not
	`To use a modified julian day number use: mJD >= 1`
	`otherwise use:                          mJD  < 1`
	`default: mJD = 0`
	`The modified julian day number (MJD) was defined in`
	`the mid 1950's in the interests of astronomy and space science`
	`as MJD = JD - 2400000.5. The half day shift makes the day start`
	`at midnight, which is the current time standard.`
	`Subtracting the large number shifts the zero day to a more`
	`recent time (November 17, 1858, midnight) allowing smaller numbers`
	`to represent time.`

Returns

myVal: The julian day number (days) since January 1, 4713 BC at 12h00

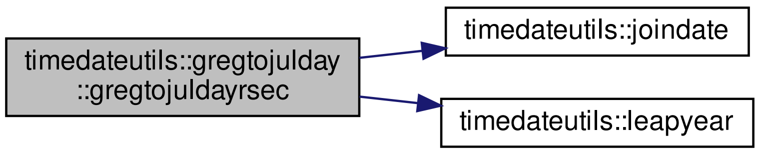
Note

The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 655 of file timedateutils.F90.

References timedateutils::firstgregdate, timedateutils::joindate(), timedateutils::leapyear(), timedateutils::mdjoffset, and timedateutils::usemodjulday.

Here is the call graph for this function:



The documentation for this interface was generated from the following file:

- timedateutils.F90

8.14 parwind::hollanddata_t Type Reference

Collaboration diagram for parwind::hollanddata_t:

Public Attributes

- character(len=fnamelen) filename
- character(len=10) thisstorm
- logical loaded = .FALSE.
- integer numrec
- character(len=2), dimension(:), allocatable basin
- integer, dimension(:), allocatable stormnumber
- character(len=10), dimension(:), allocatable dtg
- integer, dimension(:), allocatable year
- integer, dimension(:), allocatable month
- integer, dimension(:), allocatable day
- integer, dimension(:), allocatable hour
- real(sz), dimension(:), allocatable casttime
- character(len=4), dimension(:), allocatable casttype
- integer, dimension(:), allocatable fcstinc
- integer, dimension(:), allocatable ilat
- integer, dimension(:), allocatable ilon
- real(sz), dimension(:), allocatable lat
- real(sz), dimension(:), allocatable lon
- integer, dimension(:), allocatable ispeed
- real(sz), dimension(:), allocatable speed
- integer, dimension(:), allocatable icpress
- real(sz), dimension(:), allocatable cpress
- integer, dimension(:), allocatable irrp
- real(sz), dimension(:), allocatable rrp
- integer, dimension(:), allocatable irmw
- real(sz), dimension(:), allocatable rmw
- real(sz), dimension(:), allocatable cprdt
- real(sz), dimension(:), allocatable trvx
- real(sz), dimension(:), allocatable trvy

8.14.1 Detailed Description

Definition at line 115 of file parwind-orig.F90.

8.14.2 Member Data Documentation

8.14.2.1 basin `character(len=2), dimension(:), allocatable parwind::hollanddata_t::basin`

Definition at line 121 of file parwind-orig.F90.

8.14.2.2 casttime `real(sz), dimension(:), allocatable parwind::hollanddata_t::casttime`

Definition at line 125 of file parwind-orig.F90.

8.14.2.3 casttype `character(len=4), dimension(:), allocatable parwind::hollanddata_t::casttype`

Definition at line 126 of file parwind-orig.F90.

8.14.2.4 cprdt `real(sz), dimension(:), allocatable parwind::hollanddata_t::cprdt`

Definition at line 144 of file parwind-orig.F90.

8.14.2.5 cpress `real(sz), dimension(:), allocatable parwind::hollanddata_t::cpress`

Definition at line 136 of file parwind-orig.F90.

8.14.2.6 day `integer, dimension(:), allocatable parwind::hollanddata_t::day`

Definition at line 124 of file parwind-orig.F90.

8.14.2.7 dtg `character(len=10), dimension(:), allocatable parwind::hollanddata_t::dtg`

Definition at line 123 of file parwind-orig.F90.

8.14.2.8 fcstinc `integer, dimension(:), allocatable parwind::hollanddata_t::fcstinc`

Definition at line 127 of file parwind-orig.F90.

8.14.2.9 filename `character(len=fnamelen) parwind::hollanddata_t::filename`

Definition at line 116 of file parwind-orig.F90.

8.14.2.10 hour `integer, dimension(:), allocatable parwind::hollanddata_t::hour`

Definition at line 124 of file parwind-orig.F90.

8.14.2.11 icpress `integer, dimension(:), allocatable parwind::hollanddata_t::icpress`

Definition at line 135 of file parwind-orig.F90.

8.14.2.12 ilat `integer, dimension(:), allocatable parwind::hollanddata_t::ilat`

Definition at line 129 of file parwind-orig.F90.

8.14.2.13 ilon `integer, dimension(:), allocatable parwind::hollanddata_t::ilon`

Definition at line 129 of file parwind-orig.F90.

8.14.2.14 irmw `integer, dimension(:), allocatable parwind::hollanddata_t::irmw`

Definition at line 141 of file parwind-orig.F90.

8.14.2.15 irrp `integer, dimension(:), allocatable parwind::hollanddata_t::irrp`

Definition at line 138 of file parwind-orig.F90.

8.14.2.16 ispeed `integer, dimension(:), allocatable parwind::hollanddata_t::ispeed`

Definition at line 132 of file parwind-orig.F90.

8.14.2.17 lat `real(sz), dimension(:), allocatable parwind::hollanddata_t::lat`

Definition at line 130 of file parwind-orig.F90.

8.14.2.18 loaded `logical parwind::hollanddata_t::loaded = .FALSE.`

Definition at line 118 of file parwind-orig.F90.

8.14.2.19 lon `real(sz), dimension(:), allocatable parwind::hollanddata_t::lon`

Definition at line 130 of file parwind-orig.F90.

8.14.2.20 month `integer, dimension(:), allocatable parwind::hollanddata_t::month`

Definition at line 124 of file parwind-orig.F90.

8.14.2.21 numrec `integer parwind::hollanddata_t::numrec`

Definition at line 119 of file parwind-orig.F90.

8.14.2.22 rmw `real(sz), dimension(:), allocatable parwind::hollanddata_t::rmw`

Definition at line 142 of file parwind-orig.F90.

8.14.2.23 rrp `real(sz), dimension(:), allocatable parwind::hollanddata_t::rrp`

Definition at line 139 of file parwind-orig.F90.

8.14.2.24 speed `real(sz), dimension(:), allocatable parwind::hollanddata_t::speed`

Definition at line 133 of file parwind-orig.F90.

8.14.2.25 stormnumber `integer, dimension(:), allocatable parwind::hollanddata_t::stormnumber`

Definition at line 122 of file parwind-orig.F90.

8.14.2.26 thisstorm `character(len=10) parwind::hollanddata_t::thisstorm`

Definition at line 117 of file parwind-orig.F90.

8.14.2.27 trvx `real(sz), dimension(:), allocatable parwind::hollanddata_t::trvx`

Definition at line 145 of file parwind-orig.F90.

8.14.2.28 trvy `real(sz), dimension(:), allocatable parwind::hollanddata_t::trvy`

Definition at line 145 of file parwind-orig.F90.

8.14.2.29 year `integer, dimension(:), allocatable parwind::hollanddata_t::year`

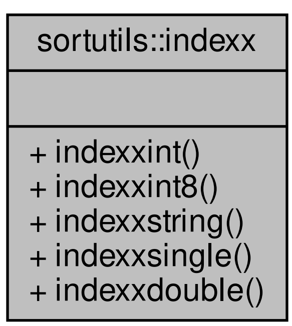
Definition at line 124 of file parwind-orig.F90.

The documentation for this type was generated from the following files:

- parwind-orig.F90
- parwind.F90

8.15 sortutils::indexx Interface Reference

Collaboration diagram for sortutils::indexx:



Public Member Functions

- subroutine indexxint (arr1D, idx1D, status)
  Indexes a 1D integer array in ascending order.
- subroutine indexxint8 (arr1D, idx1D, status)
  Indexes a 1D 32-bit integer array in ascending order.
- subroutine indexxstring (arr1D, idx1D, status, caseSens)
  Indexes a 1D string array in ascending order.
- subroutine indexxsingle (arr1D, idx1D, status)
  Indexes a 1D single precision array in ascending order.
- subroutine indexxdouble (arr1D, idx1D, status)
  Indexes a 1D double precision array in ascending order.

8.15.1 Detailed Description

Definition at line 24 of file sortutils.F90.

8.15.2 Member Function/Subroutine Documentation

8.15.2.1 indexxdouble()

```
subroutine sortutils::indexx::indexxdouble (
        real(hp), dimension(:), intent(in) arr1D,
        integer, dimension(:), intent(out) idx1D,
        integer, intent(out), optional status )
```

Indexes a 1D double precision array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (double precision)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

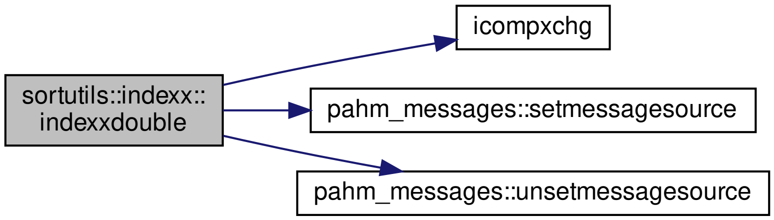
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 779 of file sortutils.F90.

References pahm_messages::error, icompxchg(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



8.15.2.2 indexxint()

```
subroutine sortutils::indexx::indexxint (
        integer, dimension(:), intent(in) arr1D,
        integer, dimension(:), intent(out) idx1D,
        integer, intent(out), optional status )
```

Indexes a 1D integer array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (integer)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

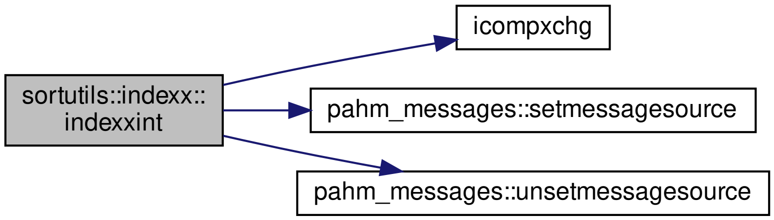
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 85 of file sortutils.F90.

References pahm_messages::error, icompxchg(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



8.15.2.3 indexxint8()

```
subroutine sortutils::indexx::indexxint8 (
    integer(int8), dimension(:), intent(in) arr1D,
    integer, dimension(:), intent(out) idx1D,
    integer, intent(out), optional status )
```

Indexes a 1D 32-bit integer array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (integer)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

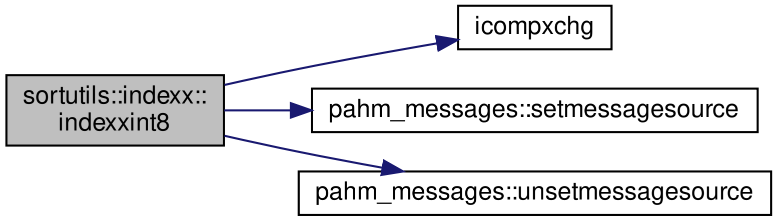
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 257 of file sortutils.F90.

References pahm_messages::error, icompxchg(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



8.15.2.4 indexxsingle() `subroutine sortutils::indexx::indexxsingle (`

```
real(sp), dimension(:), intent(in) arr1D,
integer, dimension(:), intent(out) idx1D,
integer, intent(out), optional status )
```

Indexes a 1D single precision array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed.

Parameters

arr1D	The array to be indexed (single precision)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)

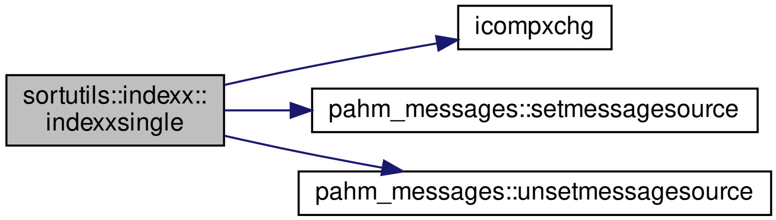
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 607 of file sortutils.F90.

References pahm_messages::error, icompxchg(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



8.15.2.5 indexxstring()

```
subroutine sortutils::indexx::indexxstring (
    character(len=*), dimension(:), intent(in) arr1D,
    integer, dimension(:), intent(out) idx1D,
    integer, intent(out), optional status,
    logical, intent(in), optional caseSens )
```

Indexes a 1D string array in ascending order.

Indexes the 1D array arr1D, i.e., outputs the array index of length N such that arr1D(idx1D(j)) is in ascending order for j = 1, 2, . . . , N. The input quantity arr1D is not changed. Modified version of IndexxInt to account for string comparisons

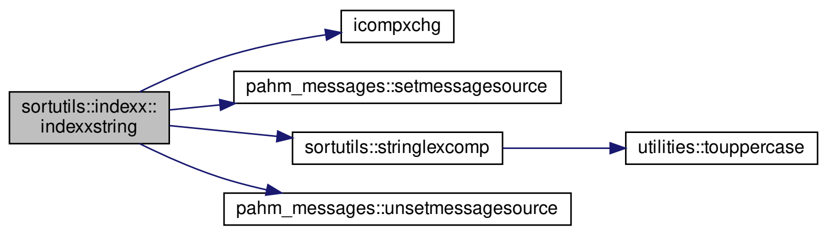
Parameters

arr1D	The array to be indexed (string)
idx1D	The array of "indexed" indexes of arr1D (output)
status	The error status, no error: status = 0 (output)
caseSens	Logical flag to request case sensitive sort

Definition at line 430 of file sortutils.F90.

References pahm_messages::error, icompxchg(), pahm_messages::scratchmessage, pahm_messages::setmessagesource(), sortutils::stringlexcomp(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:

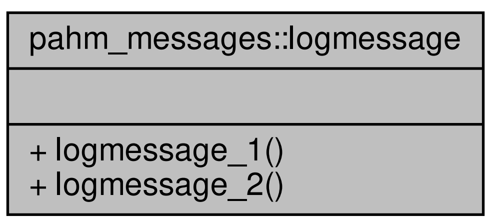


The documentation for this interface was generated from the following file:

- sortutils.F90

8.16 pahm_messages::logmessage Interface Reference

Collaboration diagram for pahm_messages::logmessage:



Public Member Functions

- subroutine logmessage_1 (message)

  General purpose subroutine to write a message to the log file.
- subroutine logmessage_2 (level, message)

8.16.1 Detailed Description

Definition at line 49 of file messages.F90.

8.16.2 Member Function/Subroutine Documentation

8.16.2.1 logmessage_1() `subroutine pahm_messages::logmessage::logmessage_1 ( character(len=*), intent(in) message )`

General purpose subroutine to write a message to the log file.

This subroutine assumes that the global variable "caller" has been set to the name of the subroutine calling it. Therefore, the SetMessageSource subroutine must be called at the beginning of the subroutine that calls this one, and Unset←MessageSource must be called at the end.

Definition at line 245 of file messages.F90.

References pahm_messages::logfileopened, pahm_messages::loginitcalled, pahm_global::lun_log, pahm_messages::messagesources, and pahm_messages::sourcenumber.

8.16.2.2 logmessage_2() `subroutine pahm_messages::logmessage::logmessage_2 ( integer, intent(in) level, character(len=*), intent(in) message )`

Definition at line 269 of file messages.F90.

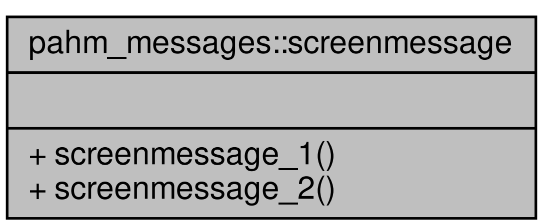
References pahm_messages::logfileopened, pahm_messages::loginitcalled, pahm_messages::loglevelnames, pahm_global::lun_log, pahm_messages::messagesources, and pahm_messages::sourcenumber.

The documentation for this interface was generated from the following file:

- messages.F90

8.17 pahm_messages::screenmessage Interface Reference

Collaboration diagram for pahm_messages::screenmessage:



Public Member Functions

- subroutine screenmessage_1 (message)

  General purpose subroutine to write a message to the screen.
- subroutine screenmessage_2 (level, message)

8.17.1 Detailed Description

Definition at line 54 of file messages.F90.

8.17.2 Member Function/Subroutine Documentation

8.17.2.1 screenmessage_1()

```
subroutine pahm_messages::screenmessage::screenmessage_1 (
        character(len=*), intent(in) message )
```

General purpose subroutine to write a message to the screen.

General purpose subroutine to write a message to the screen with a certain "logging level", and subject to the user's selection of where to write screen output.

This subroutine assumes that the global variable "caller" has been set to the name of the subroutine calling it. Therefore, the SetMessageSource subroutine must be called at the beginning of the subroutine that calls this one, and Unset↩ MessageSource must be called at the end.

Definition at line 177 of file messages.F90.

References pahm_messages::loginitcalled, pahm_global::lun_screen, pahm_messages::messagesources, pahm_messages::nscreen, and pahm_messages::sourcenumber.

8.17.2.2 screenmessage_2()

```
subroutine pahm_messages::screenmessage::screenmessage_2 (
        integer, intent(in) level,
        character(len=*), intent(in) message )
```

Definition at line 201 of file messages.F90.

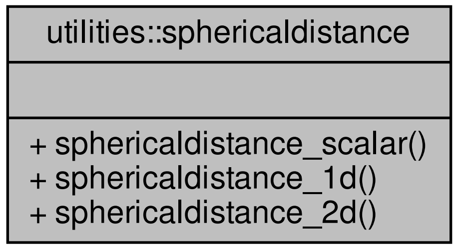
References pahm_messages::loginitcalled, pahm_messages::loglevelnames, pahm_global::lun_screen, pahm_messages::messagesourc pahm_messages::nscreen, and pahm_messages::sourcenumber.

The documentation for this interface was generated from the following file:

- messages.F90

8.18 utilities::sphericaldistance Interface Reference

Collaboration diagram for utilities::sphericaldistance:



Public Member Functions

- real(sz) function sphericaldistance_scalar (lat1, lon1, lat2, lon2)

  Calculates the distance of two points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:), allocatable sphericaldistance_1d (lats, lons, lat0, lon0)

  Calculates the distance of points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:, :), allocatable sphericaldistance_2d (lats, lons, lat0, lon0)

  Calculates the distance of points along the great circle using the Vincenty formula.

8.18.1 Detailed Description

Definition at line 39 of file utilities.F90.

8.18.2 Member Function/Subroutine Documentation

8.18.2.1 sphericaldistance_1d()

```
real(sz) function, dimension(:), allocatable utilities::sphericaldistance↩
::sphericaldistance_1d (
            real(sz), dimension(:), intent(in) lats,
            real(sz), dimension(:), intent(in) lons,
            real(sz), intent(in) lat0,
            real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lats	Latitude of first points - real, 1D array
lons	Longitude of first points - real, 1D array
lat0	Latitude of second point - real, scalar
lon0	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters, 1D array

Definition at line 2061 of file utilities.F90.

References pahm_global::deg2rad, and pahm_global::rearth.

8.18.2.2 sphericaldistance_2d()

```
real(sz) function, dimension(:, :), allocatable utilities::sphericaldistance↩
::sphericaldistance_2d (
            real(sz), dimension(:, :), intent(in) lats,
            real(sz), dimension(:, :), intent(in) lons,
            real(sz), intent(in) lat0,
            real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lats	Latitude of first points - real, 2D array
lons	Longitude of first points - real, 2D array
lat0	Latitude of second point - real, scalar
lon0	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters, 2D array

Definition at line 2160 of file utilities.F90.

References pahm_global::deg2rad, and pahm_global::rearth.

8.18.2.3 sphericaldistance_scalar()

```
real(sz) function utilities::sphericaldistance::sphericaldistance_scalar (
            real(sz), intent(in) lat1,
            real(sz), intent(in) lon1,
            real(sz), intent(in) lat2,
            real(sz), intent(in) lon2 )
```

Calculates the distance of two points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas

https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".
Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points
(Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

lat1	Latitude of first point - real, scalar
lon1	Longitude of first point - real, scalar
lat2	Latitude of second point - real, scalar
lon2	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters

Definition at line 1993 of file utilities.F90.

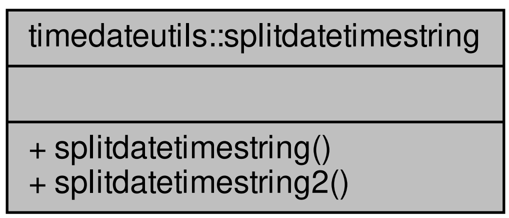
References pahm_global::deg2rad, and pahm_global::rearth.

The documentation for this interface was generated from the following file:

- utilities.F90

8.19 timedateutils::splitdatetimestring Interface Reference

Collaboration diagram for timedateutils::splitdatetimestring:



Public Member Functions

- subroutine splitdatetimestring (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
  Splits a date string into components.
- subroutine splitdatetimestring2 (inDateTime, iDate, iTime)
  Splits a date string into two components.

8.19.1 Detailed Description

Definition at line 37 of file timedateutils.F90.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 splitdatetimestring() `subroutine timedateutils::splitdatetimestring::splitdatetimestring (`

```
character(len=*), intent(in) inDateTime,
integer, intent(out) iYear,
integer, intent(out) iMonth,
integer, intent(out) iDay,
integer, intent(out) iHour,
integer, intent(out) iMin,
integer, intent(out) iSec )
```

Splits a date string into components.

This subroutine splits the string inDate (YYYYMMDDhhmmss) in six integers that is, "iYear (YYYY)", "iMonth (MM)", "iDay (DD)", "iHour (hh)", "iMin (mm)" and "iSec (ss)".

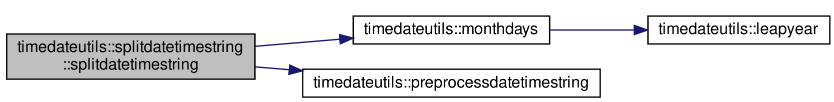
Parameters

inDateTime	The input date string: YYYYMMDDhhmmss
iYear	The year (YYYY, integer, 1582 <= YYYY, output)
iMonth	The month of the year (MM, integer, 1 <= MM <=12, output)
iDay	The day of the month (DD, integer, 1 <= DD <=31, output)
iHour	The hour of the day (hh, integer, 0 <= hh <= 23, output)
iMin	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
iSec	The second of the minute (ss, integer, 0 <= ss <= 59, output)

Definition at line 1073 of file timedateutils.F90.

References timedateutils::monthdays(), and timedateutils::preprocessdatetimestring().

Here is the call graph for this function:



8.19.3 Member Function/Subroutine Documentation

8.19.3.1 splitdatetimestring2() `subroutine timedateutils::splitdatetimestring::splitdatetimestring2 (`

```
character(len=*), intent(in) inDateTime,
integer, intent(out) iDate,
integer, intent(out) iTime )
```

Splits a date string into two components.

This subroutine splits the string inDate (YYYYMMDDhhmmss) in two integers that is, "iDate (YYYYMMDD)" and "iTime (hhmmss)".

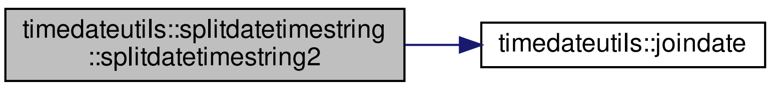
Parameters

inDateTime	The input date string: YYYYMMDDhhmmss
iDate	The integer date (YYYYMMDD, output)
iTime	The integer time (hhmmss, output)

Definition at line 1141 of file timedateutils.F90.

References timedateutils::joindate().

Here is the call graph for this function:

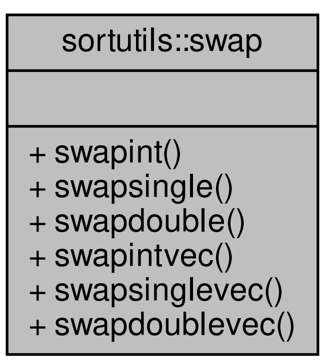


The documentation for this interface was generated from the following file:

- timedateutils.F90

8.20 sortutils::swap Interface Reference

Collaboration diagram for sortutils::swap:



Public Member Functions

- subroutine swapint (a, b, mask)

  Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine swapsingle (a, b, mask)

  Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine swapdouble (a, b, mask)

  Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").
- subroutine swapintvec (a, b, mask)

  Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine swapsinglevec (a, b, mask)

  Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine swapdoublevec (a, b, mask)

  Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").

8.20.1 Detailed Description

Definition at line 50 of file sortutils.F90.

8.20.2 Member Function/Subroutine Documentation

8.20.2.1 swapdouble()

```
subroutine sortutils::swap::swapdouble (
        real(hp), intent(inout) a,
        real(hp), intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (double precision)
b	The second value to be swapped (double precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1613 of file sortutils.F90.

8.20.2.2 swapdoublevec()

```
subroutine sortutils::swap::swapdoublevec (
        real(hp), dimension(:), intent(inout) a,
        real(hp), dimension(:), intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (double precision)
b	The second 1D array to be swapped (double precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1769 of file sortutils.F90.

8.20.2.3 swapint()

```
subroutine sortutils::swap::swapint (
        integer, intent(inout) a,
        integer, intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (integer)
b	The second value to be swapped (integer)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1509 of file sortutils.F90.

8.20.2.4 swapintvec()

```
subroutine sortutils::swap::swapintvec (
        integer, dimension(:), intent(inout) a,
        integer, dimension(:), intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (integer)
b	The second 1D array to be swapped (integer)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1665 of file sortutils.F90.

8.20.2.5 swapsingle()

```
subroutine sortutils::swap::swapsingle (
        real(sp), intent(inout) a,
        real(sp), intent(inout) b,
        logical, intent(in), optional mask )
```

Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").

Parameters

a	The first value to be swapped (single precision)
b	The second value to be swapped (single precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped value
b: The first swapped value
```

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1561 of file sortutils.F90.

8.20.2.6 swapsinglevec() `subroutine sortutils::swap::swapsinglevec ( real(sp), dimension(:), intent(inout) a, real(sp), dimension(:), intent(inout) b, logical, intent(in), optional mask )`

Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").

Parameters

a	The first 1D array to be swapped (single precision)
b	The second 1D array to be swapped (single precision)
mask	Logical flag to perform the swap, default mask = 'TRUE. (optional)

Returns

```
a: The second swapped 1D array
b: The first swapped 1D array
```

Note

Adopted from Numerical Recipes for Fortran 90

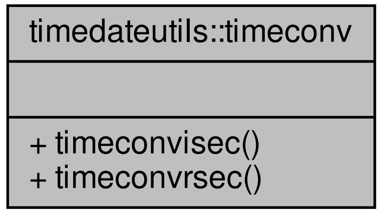
Definition at line 1717 of file sortutils.F90.

The documentation for this interface was generated from the following file:

- sortutils.F90

8.21 timedateutils::timeconv Interface Reference

Collaboration diagram for timedateutils::timeconv:



Public Member Functions

- subroutine timeconvisec (iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- subroutine timeconvrsec (iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

8.21.1 Detailed Description

Definition at line 26 of file timedateutils.F90.

8.21.2 Member Function/Subroutine Documentation

8.21.2.1 timeconvisec()

```
subroutine timedateutils::timeconv::timeconvisec (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay,
        integer, intent(in) iHour,
        integer, intent(in) iMin,
        integer, intent(in) iSec,
        real(sz), intent(out) timeSec )
```

Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

The reference date is defined by the global variables: refYear, refMonth, refDay, refHour, refMin and refSec. It uses GregToJulDay and ElapsedSecs functions to calculate the elapsed time from the reference date.

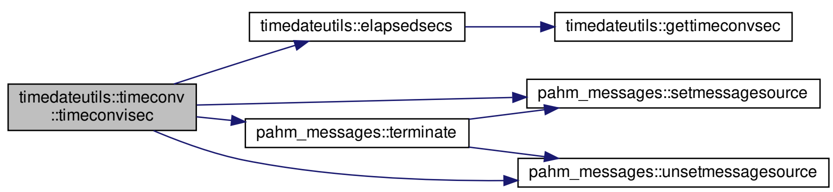
Parameters

iYear	The year (integer)
iMonth	The month of the year (1-12, integer)
iDay	The day of the month (1-31, integer)
iHour	The hour of the day (0-23, integer)
iMin	The minute of the hour (0-59, integer)
iSec	The second of the minute (0-59, integer)
timeSec	The elapsed time in seconds (real, output)

Definition at line 125 of file timedateutils.F90.

References timedateutils::elapsedsecs(), pahm_messages::error, pahm_global::refday, pahm_global::refhour, pahm_global::refmin, pahm_global::refmonth, pahm_global::refsec, pahm_global::refyear, pahm_sizes::rmissv, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_messages::terminate(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:



8.21.2.2 timeconvrsec()

```
subroutine timedateutils::timeconv::timeconvrsec (
        integer, intent(in) iYear,
        integer, intent(in) iMonth,
        integer, intent(in) iDay,
        integer, intent(in) iHour,
        integer, intent(in) iMin,
        real(sz), intent(in) rSec,
        real(sz), intent(out) timeSec )
```

Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

The reference date is defined by the global variables: refYear, refMonth, refDay, refHour, refMin and refSec. It uses GregToJulDay and ElapsedSecs functions to calculate the elapsed time from the reference date. Similar to TimeConv↩ISEC but seconds are entered as real numbers to allow for fractions of a second.

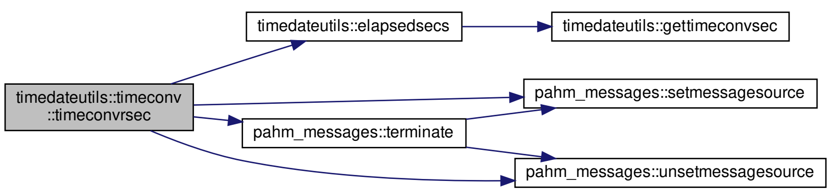
Parameters

iYear	The year (integer)
iMonth	The month of the year (1-12, integer)
iDay	The day of the month (1-31, integer)
iHour	The hour of the day (0-23, integer)
iMin	The minute of the hour (0-59, integer)
rSec	The second of the minute (0-59, real)
timeSec	The elapsed time in seconds (real, output)

Definition at line 202 of file timedateutils.F90.

References timedateutils::elapsedsecs(), pahm_messages::error, pahm_global::refday, pahm_global::refhour, pahm_global::refmin, pahm_global::refmonth, pahm_global::refsec, pahm_global::refyear, pahm_sizes::rmissv, pahm_messages::scratchmessage, pahm_messages::setmessagesource(), pahm_messages::terminate(), and pahm_messages::unsetmessagesource().

Here is the call graph for this function:

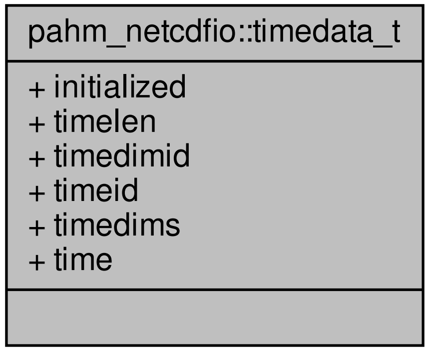


The documentation for this interface was generated from the following file:

- timedateutils.F90

8.22 pahm_netcdfio::timedata_t Type Reference

Collaboration diagram for pahm_netcdfio::timedata_t:



Public Attributes

- logical initialized = .FALSE.
- integer timelen = 1
- integer timedimid
- integer timeid
- integer, dimension(1) timedims
- real(sz), dimension(:), allocatable time

8.22.1 Detailed Description

Definition at line 38 of file netcdfio-nems.F90.

8.22.2 Member Data Documentation

8.22.2.1 initialized `logical pahm_netcdfio::timedata_t::initialized = .FALSE.`

Definition at line 39 of file netcdfio-nems.F90.

8.22.2.2 time `real(sz), dimension(:), allocatable pahm_netcdfio::timedata_t::time`

Definition at line 44 of file netcdfio-nems.F90.

8.22.2.3 timedimid `integer pahm_netcdfio::timedata_t::timedimid`

Definition at line 41 of file netcdfio-nems.F90.

8.22.2.4 timedims `integer, dimension(1) pahm_netcdfio::timedata_t::timedims`

Definition at line 43 of file netcdfio-nems.F90.

8.22.2.5 timeid `integer pahm_netcdfio::timedata_t::timeid`

Definition at line 42 of file netcdfio-nems.F90.

8.22.2.6 timelen `integer pahm_netcdfio::timedata_t::timelen = 1`

Definition at line 40 of file netcdfio-nems.F90.

The documentation for this type was generated from the following files:

- netcdfio-nems.F90
- netcdfio-orig.F90
- netcdfio.F90

9 File Documentation

9.1 dev_doc.md File Reference

9.2 mainpage.md File Reference

9.3 model.md File Reference

9.4 csv_module.F90 File Reference

For reading and writing CSV files.

Data Types

- type csv_module::csv_string
- type csv_module::csv_file

Modules

- module csv_module

Functions/Subroutines

- subroutine csv_module::initialize_csv_file (me, quote, delimiter, enclose_strings_in_quotes, enclose_all_in_quotes, logical_true_string, logical_false_string, chunk_size)

    Initialize a [[csv_file(type)]].

Variables

- integer, parameter, public csv_module::csv_type_string = 1
- integer, parameter, public csv_module::a
- integer, parameter, public csv_module::character
- integer, parameter, public csv_module::string
- integer, parameter, public csv_module::cell
- integer, parameter, public csv_module::csv_type_double = 2
- integer, dimension(wp), parameter, public csv_module::real
- integer, parameter, public csv_module::csv_type_integer = 3
- integer, parameter, public csv_module::an
- integer, dimension(ip), parameter, public csv_module::integer
- integer, parameter, public csv_module::csv_type_logical = 4
- integer, parameter, public csv_module::logical

9.4.1 Detailed Description

For reading and writing CSV files.

Author

Jacob Williams

Copyright

License BSD

Definition in file csv_module.F90.

9.5 csv_module.F90

Go to the documentation of this file.

```
!------------------------------------------------------------------
!                M O D U L E   C S V _ M O D U L E
!------------------------------------------------------------------
!------------------------------------------------------------------

    module csv_module

    USE pahm_sizes, ONLY : wp, ip
    USE pahm_global, ONLY : lun_btrk, lun_btrk1
    USE pahm_messages
    USE utilities, ONLY : openfileforread, tolowercase
    use csv_utilities
    use csv_parameters

    implicit none

    private

    ! the different types of variables that can be in a CSV file.
    integer,parameter,public :: csv_type_string  = 1
    integer,parameter,public :: csv_type_double  = 2
    integer,parameter,public :: csv_type_integer = 3
    integer,parameter,public :: csv_type_logical = 4

    real(wp),parameter :: zero = 0.0_wp

    type,public :: csv_string
        character(len=:),allocatable :: str
    end type csv_string

    type,public :: csv_file


        private

        character(len=1) :: quote     = '"'
        character(len=1) :: delimiter = ','

        ! for reading a csv file:
        integer,public :: n_rows = 0  ! number of rows in the file
        integer,public :: n_cols = 0  ! number of columns in the file
        integer :: chunk_size = 100   ! for expanding vectors
        type(csv_string),dimension(:),allocatable :: header      ! the header
        type(csv_string),dimension(:,:),allocatable :: csv_data  ! the data in the file

        ! for writing a csv file:
        integer :: icol = 0           ! last column written in current row
        integer :: iunit = lun_btrk   ! file unit for writing
        logical :: enclose_strings_in_quotes = .true.  ! if true, all string cells
                                                       ! will be enclosed in quotes.
        logical :: enclose_all_in_quotes = .false.     ! if true, *all* cells will
                                                       ! be enclosed in quotes.
        character(len=1) :: logical_true_string = 'T'  ! when writing a logical `true`
                                                       ! value to a CSV file, this
                                                       ! is the string to use
                                                       ! (default is `T`)
        character(len=1) :: logical_false_string = 'F' ! when writing a logical `false`
                                                       ! value to a CSV file, this
                                                       ! is the string to use
                                                       ! (default is `F`)

    contains

        private

        procedure,public :: initialize => initialize_csv_file
        procedure,public :: read => read_csv_file
        procedure,public :: destroy => destroy_csv_file

        procedure,public :: variable_types

        generic,public :: get_header => get_header_str,&
                                        get_header_csv_str
        procedure :: get_header_str
        procedure :: get_header_csv_str

```

```
        ! For getting data from the class
        ! after the file has been read.
        generic,public :: get => get_csv_data_as_str,&
                                 csv_get_value,&
                                 get_real_column,&
                                 get_integer_column,&
                                 get_logical_column,&
                                 get_character_column,&
                                 get_csv_string_column
        procedure :: get_csv_data_as_str
        procedure :: csv_get_value
        procedure :: get_real_column
        procedure :: get_integer_column
        procedure :: get_logical_column
        procedure :: get_character_column
        procedure :: get_csv_string_column

        procedure,public :: open => open_csv_file

        generic,public :: add => add_cell,&
                                 add_vector,&
                                 add_matrix
        procedure :: add_cell
        procedure :: add_vector
        procedure :: add_matrix

        procedure,public :: next_row
        procedure,public :: close => close_csv_file

        procedure :: tokenize => tokenize_csv_line
        procedure :: read_line_from_file
        procedure :: get_column

    end type csv_file

    contains
!*****************************************************************************************

  !---------------------------------------------------------------
  ! S U B R O U T I N E   E X P A N D _ V E C T O R
  !---------------------------------------------------------------
  !---------------------------------------------------------------
    subroutine initialize_csv_file(me,quote,delimiter,&
                                    enclose_strings_in_quotes,&
                                    enclose_all_in_quotes,&
                                    logical_true_string,&
                                    logical_false_string,&
                                    chunk_size)

    implicit none

    class(csv_file),intent(out) :: me
    character(len=1),intent(in),optional :: quote             ! note: can only be one character
                                                              ! (Default is '"')
    character(len=1),intent(in),optional :: delimiter         ! note: can only be one character
                                                              ! (Default is ',')
    logical,intent(in),optional :: enclose_strings_in_quotes  ! if true, all string cells
                                                              ! will be enclosed in quotes.
                                                              ! (Default is True)
    logical,intent(in),optional :: enclose_all_in_quotes      ! if true, *all* cells will
                                                              ! be enclosed in quotes.
                                                              ! (Default is False)
    character(len=1),intent(in),optional :: logical_true_string ! when writing a logical 'true'
                                                                ! value to a CSV file, this
                                                                ! is the string to use
                                                                ! (default is 'T')
    character(len=1),intent(in),optional :: logical_false_string ! when writing a logical 'false'
                                                                 ! value to a CSV file, this
                                                                 ! is the string to use
                                                                 ! (default is 'F')
    integer,intent(in),optional :: chunk_size  ! factor for expanding vectors
                                               ! (default is 100)

    if (present(quote)) me%quote = quote
    if (present(delimiter)) me%delimiter = delimiter
    if (present(enclose_strings_in_quotes)) &
        me%enclose_strings_in_quotes = enclose_strings_in_quotes
    if (present(enclose_all_in_quotes)) &
        me%enclose_all_in_quotes = enclose_all_in_quotes
    if (present(logical_true_string))  &
        me%logical_true_string = logical_true_string
```

```
    if (present(logical_false_string)) &
        me%logical_false_string = logical_false_string
    if (present(chunk_size)) me%chunk_size = chunk_size

    ! override:
    if (me%enclose_all_in_quotes) me%enclose_strings_in_quotes = .true.

    end subroutine initialize_csv_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine destroy_csv_file(me)

    implicit none

    class(csv_file),intent(out) :: me

    end subroutine destroy_csv_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine read_csv_file(me,filename,header_row,skip_rows,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    character(len=*),intent(in) :: filename
    logical,intent(out) :: status_ok
    integer,intent(in),optional :: header_row
    integer,dimension(:),intent(in),optional :: skip_rows

    type(csv_string),dimension(:),allocatable :: row_data
    type(csv_string) :: empty_data
    integer,dimension(:),allocatable :: rows_to_skip
    character(len=:),allocatable :: line
    integer :: i
    integer :: j
    integer :: irow
    integer :: n_rows_in_file
    integer :: n_rows
    integer :: n_cols
    integer :: istat
    integer :: line_n_cols
    integer :: iunit
    logical :: arrays_allocated
    integer :: iheader
    character(len=1) :: tmp

    empty_data%str = ' '
    iunit = lun_btrk

    CALL setmessagesource("read_csv_file")

    call me%destroy()
    arrays_allocated = .false.

    CALL openfileforread(iunit, trim(adjustl(filename)), istat)

    IF (istat /= 0) THEN
      WRITE(scratchmessage, '(a)') 'Error opening the file: ' // trim(adjustl(filename))
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      CALL terminate()
    ELSE
      WRITE(scratchmessage, '(a)') 'Processing the file: ' // trim(adjustl(filename))
      CALL logmessage(info, scratchmessage)
    END IF

!     if (istat==0) then

        !get number of lines in the file
        n_rows_in_file = number_of_lines_in_file(iunit)

        !get number of lines in the data array
        if (present(skip_rows)) then
```

```
            !get size of unique elements in skip_rows,
            !and subtract from n_rows_in_file
            rows_to_skip = unique(skip_rows,chunk_size=me%chunk_size)
            n_rows = n_rows_in_file - size(rows_to_skip)
        else
            n_rows = n_rows_in_file
        end if
        if (present(header_row)) then
            iheader = max(0,header_row)
            n_rows = n_rows - 1
        else
            iheader = 0
        end if

        me%n_rows = n_rows

        ! we don't know the number of columns
        ! until we parse the first row (or the header)
        ! Panagiotis Velissariou: some csv files do not have the same number
        ! of columns, so we need to determine the nax number of columns
        ! for the allocation of the arrays
        !--- PV
        n_cols = 0
        do i=1,n_rows_in_file
          call me%read_line_from_file(iunit,line,status_ok)
          call me%tokenize(line,row_data)
          n_cols = max(n_cols,size(row_data))
        end do
        rewind(iunit)

        me%n_cols = n_cols
        allocate(me%csv_data(n_rows,n_cols))
        if (iheader/=0) allocate(me%header(n_cols))
        arrays_allocated = .true.
        !--- PV

        !read each line in the file, parse it, and populate data
        irow = 0
        do i=1,n_rows_in_file

            ! skip row if necessary
            if (allocated(rows_to_skip)) then
                if (any(i==rows_to_skip)) then
                    read(iunit,fmt='(A1)',iostat=istat) tmp
                    if (istat/=0) then
                        scratchmessage = 'Error skipping row in file: '//trim(filename)
                        CALL  allmessage(error, scratchmessage)

                        close(unit=iunit,iostat=istat)
                        status_ok = .false.

                        CALL unsetmessagesource()
                        return
                    end if
                    cycle
                end if
            end if

            call me%read_line_from_file(iunit,line,status_ok)
            if (.not. status_ok) then
              CALL unsetmessagesource()
              return ! file read error
            end if
            call me%tokenize(line,row_data)
            line_n_cols = size(row_data)

            if (i==iheader) then
                do j=1,me%n_cols
                    me%header(j)%str = row_data(j)%str
                end do
            else
                irow = irow + 1
                do j=1,n_cols
                    if(j <= line_n_cols) then
                      me%csv_data(irow,j) = row_data(j) !%str
                    else
                      me%csv_data(irow,j) = empty_data !%str
                    end if
                end do
            end if

```

```
        end do

        ! close the file
        close(unit=iunit,iostat=istat)

        status_ok = .true.

!    else
!        scratchMessage = 'Error opening file: '//trim(filename)
!        CALL  AllMessage(ERROR, scratchMessage)
!        status_ok = .false.
!    end if

    CALL unsetmessagesource()

    end subroutine read_csv_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine open_csv_file(me,filename,n_cols,status_ok,append)

    implicit none

    class(csv_file),intent(inout)   :: me
    character(len=*),intent(in)     :: filename
    integer,intent(in)              :: n_cols
    logical,intent(out)             :: status_ok
    logical,intent(in),optional     :: append

    integer :: istat
    logical :: append_flag
    logical :: file_exists

    CALL setmessagesource("open_csv_file")

    call me%destroy()

    me%n_cols = n_cols

    ! optional append argument:
    append_flag = .false.
    file_exists = .false.
    if (present(append)) then
        append_flag = append
        if (append) inquire(file=filename, exist=file_exists)
    end if

    if (append_flag .and. file_exists) then
        open(unit=me%iunit,file=filename,status='OLD',position='APPEND',iostat=istat)
    else
        open(unit=me%iunit,file=filename,status='REPLACE',iostat=istat)
    end if

    if (istat==0) then
        status_ok = .true.
    else
        scratchmessage = 'Error opening file: '//trim(filename)
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

   CALL unsetmessagesource()

    end subroutine open_csv_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine close_csv_file(me,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    logical,intent(out) :: status_ok

    integer :: istat

    close(me%iunit,iostat=istat)
```

```fortran
    status_ok = istat==0

    end subroutine close_csv_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine add_cell(me,val,int_fmt,real_fmt,trim_str)

    implicit none

    class(csv_file),intent(inout) :: me
    class(*),intent(in) :: val
    character(len=*),intent(in),optional :: int_fmt
    character(len=*),intent(in),optional :: real_fmt
    logical,intent(in),optional :: trim_str

    integer :: istat
    character(len=:),allocatable :: ifmt
    character(len=:),allocatable :: rfmt
    logical :: trimstr
    character(len=max_real_str_len) :: real_val
    character(len=max_integer_str_len) :: int_val

    CALL setmessagesource("add_cell")

    ! make sure the row isn't already finished
    if (me%icol<me%n_cols) then

        me%icol = me%icol + 1

        if (me%enclose_all_in_quotes) then
            write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
        end if

        select type (val)
        type is (integer(ip))
            if (present(int_fmt)) then
                ifmt = trim(adjustl(int_fmt))
            else
                ifmt = default_int_fmt
            end if
            write(int_val,fmt=ifmt,iostat=istat) val
            write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(int_val))
        type is (real(wp))
            if (present(real_fmt)) then
                rfmt = trim(adjustl(real_fmt))
            else
                rfmt = default_real_fmt
            end if
            write(real_val,fmt=rfmt,iostat=istat) val
            write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(real_val))
        type is (logical)
            if (val) then
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_true_string
            else
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_false_string
            end if
        type is (character(len=*))
            if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
            if (present(trim_str)) then
                trimstr = trim_str
            else
                trimstr = .false.
            end if
            if (trimstr) then
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val)
            else
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val
            end if
            if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
        type is (csv_string)
            if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
            if (present(trim_str)) then
                trimstr = trim_str
            else
                trimstr = .false.
```

```
            end if
            if (trimstr) then
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val%str)
            else
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val%str
            end if
            if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
                write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
        class default
            scratchmessage = 'Error: cannot write unknown variable type to CSV file.'
            CALL  allmessage(error, scratchmessage)
        end select

        if (me%enclose_all_in_quotes) then
            write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
        end if
        if (me%icol<me%n_cols) write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%delimiter

    else
        scratchmessage = 'Error: cannot write more cells to the current row.'
        CALL  allmessage(error, scratchmessage)
    end if

    CALL unsetmessagesource()

    end subroutine add_cell
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine add_vector(me,val,int_fmt,real_fmt,trim_str)

    implicit none

    class(csv_file),intent(inout) :: me
    class(*),dimension(:),intent(in) :: val
    character(len=*),intent(in),optional :: int_fmt
    character(len=*),intent(in),optional :: real_fmt
    logical,intent(in),optional :: trim_str

    integer :: i

    do i=1,size(val)

#if defined __GFORTRAN__
        ! This is a stupid workaround for gfortran bugs (tested with 7.2.0)
        select type (val)
        type is (character(len=*))
            call me%add(val(i),int_fmt,real_fmt,trim_str)
        class default
            call me%add(val(i),int_fmt,real_fmt,trim_str)
        end select
#else
        call me%add(val(i),int_fmt,real_fmt,trim_str)
#endif

    end do

    end subroutine add_vector
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine add_matrix(me,val,int_fmt,real_fmt,trim_str)

    implicit none

    class(csv_file),intent(inout) :: me
    class(*),dimension(:,:),intent(in) :: val
    character(len=*),intent(in),optional :: int_fmt
    character(len=*),intent(in),optional :: real_fmt
    logical,intent(in),optional :: trim_str

    integer :: i

    ! add each row:
    do i=1,size(val,1)
        call me%add(val(i,:),int_fmt,real_fmt,trim_str)
        call me%next_row()
```

```fortran
    end do

    end subroutine add_matrix
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine next_row(me)

    implicit none

    class(csv_file),intent(inout) :: me

    integer :: i
    integer :: n

    if (me%icol>0) then
        n = me%n_cols - me%icol
        do i=1,n
            if (i==n) then !no trailing delimiter
                if (me%enclose_strings_in_quotes) then
                    write(me%iunit,'(A)',advance='NO') me%quote//me%quote
                end if
            else
                if (me%enclose_strings_in_quotes) then
                    write(me%iunit,'(A)',advance='NO') me%quote//me%quote//me%delimiter
                else
                    write(me%iunit,'(A)',advance='NO') me%delimiter
                end if
            end if
        end do
        write(me%iunit,'(A)') '' ! new line
    end if

    me%icol = 0  ! this row is finished

    end subroutine next_row
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_header_csv_str(me,header,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    type(csv_string),dimension(:),allocatable,intent(out) :: header
    logical,intent(out) :: status_ok

    integer :: i

    CALL setmessagesource("get_header_csv_str")

    if (allocated(me%header)) then

        allocate(header(me%n_cols))
        do i=1,me%n_cols
            header(i) = me%header(i)
        end do
        status_ok = .false.

    else
        scratchmessage = 'Error: no header in class.'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_header_csv_str
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_header_str(me,header,status_ok)

    implicit none

```

```fortran
    class(csv_file),intent(inout) :: me
    character(len=*),dimension(:),allocatable,intent(out) :: header
    logical,intent(out) :: status_ok

    integer :: i

    CALL setmessagesource("get_header_str")

    if (allocated(me%header)) then

        allocate(header(me%n_cols))
        do i=1,me%n_cols
            header(i) = me%header(i)%str
        end do
        status_ok = .false.

    else
        scratchmessage = 'Error: no header in class.'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_header_str
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_csv_data_as_str(me,csv_data,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    character(len=*),dimension(:,:),allocatable,intent(out) :: csv_data
    logical,intent(out) :: status_ok

    integer :: i
    integer :: j

    CALL setmessagesource("get_csv_data_as_str")

    if (allocated(me%csv_data)) then
        ! size the output array:
        allocate(csv_data(me%n_rows,me%n_cols))
        ! convert each element to a string:
        do concurrent(i=1:me%n_rows)
            do concurrent(j=1:me%n_cols)
                csv_data(i,j) = me%csv_data(i,j)%str
            end do
        end do
        status_ok = .true.
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_csv_data_as_str
!*****************************************************************************************

!*****************************************************************************************
!

    pure elemental subroutine to_real(str,val,status_ok)

    implicit none

    character(len=*),intent(in) :: str
    real(wp),intent(out) :: val
    logical,intent(out) :: status_ok

    integer :: istat

    read(str,fmt=*,iostat=istat) val
    if (istat==0) then
        status_ok = .true.
    else
```

```
        status_ok = .false.
        val = zero
    end if

    end subroutine to_real
!*****************************************************************************************

!*****************************************************************************************
!

    pure elemental subroutine to_integer(str,val,status_ok)

    implicit none

    character(len=*),intent(in) :: str
    integer(ip),intent(out) :: val
    logical,intent(out) :: status_ok

    integer :: istat

    read(str,fmt=default_int_fmt,iostat=istat) val
    if (istat==0) then
        status_ok = .true.
    else
        status_ok = .false.
        val = 0
    end if

    end subroutine to_integer
!*****************************************************************************************

!*****************************************************************************************
!

    pure elemental subroutine to_logical(str,val,status_ok)

    implicit none

    character(len=*),intent(in) :: str
    logical,intent(out) :: val
    logical,intent(out) :: status_ok

    character(len=:),allocatable :: tmp

    ! True and False options (all lowercase):
    character(len=*),dimension(4),parameter :: true_str  = ['1     ',&
                                                             't     ',&
                                                             'true  ',&
                                                             '.true.']
    character(len=*),dimension(4),parameter :: false_str = ['0      ',&
                                                             'f      ',&
                                                             'false  ',&
                                                             '.false.']

    tmp = tolowercase(str)
    if ( any(tmp==true_str) ) then
        val = .true.
        status_ok = .true.
    else if ( any(tmp==false_str) ) then
        val = .false.
        status_ok = .true.
    else
        val = .false.
        status_ok = .false.
    end if

    end subroutine to_logical
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine variable_types(me,itypes,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,dimension(:),allocatable,intent(out) :: itypes
    logical,intent(out) :: status_ok

    integer :: i
```

```fortran

    CALL setmessagesource("variable_types")

    if (allocated(me%csv_data)) then
        allocate(itypes(me%n_cols))
        do i=1,me%n_cols
            call infer_variable_type(me%csv_data(1,i)%str,itypes(i))
        end do
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine variable_types
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine infer_variable_type(str,itype)

    implicit none

    character(len=*),intent(in) :: str
    integer,intent(out) :: itype

    real(wp)    :: rval
    integer(ip) :: ival
    logical     :: lval
    logical     :: status_ok

    call to_integer(str,ival,status_ok)
    if (status_ok) then
        itype = csv_type_integer
        return
    end if

    call to_real(str,rval,status_ok)
    if (status_ok) then
        itype = csv_type_double
        return
    end if

    call to_logical(str,lval,status_ok)
    if (status_ok) then
        itype = csv_type_logical
        return
    end if

    ! default is string:
    itype = csv_type_string

    end subroutine infer_variable_type
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine csv_get_value(me,row,col,val,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in)    :: row
    integer,intent(in)    :: col
    class(*),intent(out) :: val
    logical,intent(out)   :: status_ok

    select type (val)
    type is (integer(ip))
        call to_integer(me%csv_data(row,col)%str,val,status_ok)
    type is (real(wp))
        call to_real(me%csv_data(row,col)%str,val,status_ok)
    type is (logical)
        call to_logical(me%csv_data(row,col)%str,val,status_ok)
    type is (character(len=*))
        status_ok = .true.
        if (allocated(me%csv_data(row,col)%str)) then
```

```
            val = me%csv_data(row,col)%str
        else
            val = ''
        end if
    type is (csv_string)
        status_ok = .true.
        val = me%csv_data(row,col)
    class default
        status_ok = .false.
    end select

    end subroutine csv_get_value
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    class(*),dimension(:),intent(out) :: r
    logical,intent(out) :: status_ok

    integer :: i
#if defined __GFORTRAN__
    character(len=:),allocatable :: tmp
#endif

    CALL setmessagesource("get_column")

    ! we know the data is allocated, since that
    ! was checked by the calling routines.

    if (me%n_cols>=icol .and. icol>0) then

        do i=1,me%n_rows  ! row loop

#if defined __GFORTRAN__
            ! the following is a workaround for gfortran bugs:
            select type (r)
            type is (character(len=*))
                tmp = repeat(' ',len(r)) ! size the string
                call me%csv_get_value(i,icol,tmp,status_ok)
                r(i) = tmp
            class default
                call me%csv_get_value(i,icol,r(i),status_ok)
            end select
#else
            call me%csv_get_value(i,icol,r(i),status_ok)
#endif
            if (.not. status_ok) then
                select type (r)
                ! note: character conversion can never fail, so not
                ! checking for that here. also we know it is real,
                ! integer, or logical at this point.
                type is (integer(ip))
                    scratchmessage = 'Error converting string to integer: '//trim(me%csv_data(i,icol)%str)
                    CALL  allmessage(error, scratchmessage)
                    r(i) = 0
                type is (real(wp))
                    scratchmessage = 'Error converting string to real: '//trim(me%csv_data(i,icol)%str)
                    CALL  allmessage(error, scratchmessage)
                    r(i) = zero
                type is (logical)
                    scratchmessage = 'Error converting string to logical: '//trim(me%csv_data(i,icol)%str)
                    CALL  allmessage(error, scratchmessage)
                    r(i) = .false.
                end select
            end if

        end do

    else
        WRITE(scratchmessage,'(A,1X,I5)') 'Error: invalid column number: ', icol
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

```

```
    CALL unsetmessagesource()

    end subroutine get_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_real_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    real(wp),dimension(:),allocatable,intent(out) :: r
    logical,intent(out) :: status_ok

    CALL setmessagesource("get_real_column")

    if (allocated(me%csv_data)) then
        allocate(r(me%n_rows))  ! size the output vector
        call me%get_column(icol,r,status_ok)
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_real_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_integer_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    integer(ip),dimension(:),allocatable,intent(out) :: r
    logical,intent(out) :: status_ok

    CALL setmessagesource("get_integer_column")

    if (allocated(me%csv_data)) then
        allocate(r(me%n_rows))  ! size the output vector
        call me%get_column(icol,r,status_ok)
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_integer_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_logical_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    logical,dimension(:),allocatable,intent(out) :: r
    logical,intent(out) :: status_ok

    CALL setmessagesource("get_logical_column")

    if (allocated(me%csv_data)) then
        allocate(r(me%n_rows))  ! size the output vector
        call me%get_column(icol,r,status_ok)
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
```

```
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_logical_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_character_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    character(len=*),dimension(:),allocatable,intent(out) :: r
    logical,intent(out) :: status_ok

    CALL setmessagesource("get_character_column")

    if (allocated(me%csv_data)) then
        allocate(r(me%n_rows))  ! size the output vector
        call me%get_column(icol,r,status_ok)
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_character_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine get_csv_string_column(me,icol,r,status_ok)

    implicit none

    class(csv_file),intent(inout) :: me
    integer,intent(in) :: icol
    type(csv_string),dimension(:),allocatable,intent(out) :: r
    logical,intent(out) :: status_ok

    CALL setmessagesource("get_csv_string_column")

    if (allocated(me%csv_data)) then
        allocate(r(me%n_rows))  ! size the output vector
        call me%get_column(icol,r,status_ok)
    else
        scratchmessage = 'Error: class has not been initialized'
        CALL  allmessage(error, scratchmessage)
        status_ok = .false.
    end if

    CALL unsetmessagesource()

    end subroutine get_csv_string_column
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine tokenize_csv_line(me,line,cells)

    implicit none

    class(csv_file),intent(inout) :: me
    character(len=*),intent(in) :: line
    type(csv_string),dimension(:),allocatable,intent(out) :: cells

    integer :: i
    character(len=:),allocatable :: tmp
    integer :: n

    call split(line,me%delimiter,me%chunk_size,cells)

```

```
    ! remove quotes if present:
    do i = 1, size(cells)

        ! remove whitespace from the string:
        tmp = trim(adjustl(cells(i)%str))
        n = len(tmp)

        if (n>1) then
            ! if the first and last non-blank character is
            ! a quote, then remove them and replace with what
            ! is inside the quotes. Otherwise, leave it as is.
            if (tmp(1:1)==me%quote .and. tmp(n:n)==me%quote) then
                if (n>2) then
                    cells(i)%str = tmp(2:n-1)  ! remove the quotes
                else
                    cells(i)%str = ''  ! empty string
                end if
            end if
        end if

    end do

    end subroutine tokenize_csv_line
!*****************************************************************************************

!*****************************************************************************************
!

    function number_of_lines_in_file(iunit) result(n_lines)

    implicit none

    integer,intent(in)  :: iunit
    integer :: n_lines

    character(len=1) :: tmp
    integer :: istat

    rewind(iunit)
    n_lines = 0
    do
        read(iunit,fmt='(A1)',iostat=istat) tmp
        if (is_iostat_end(istat)) exit
        n_lines = n_lines + 1
    end do
    rewind(iunit)

    end function number_of_lines_in_file
!*****************************************************************************************

!*****************************************************************************************
!

    subroutine read_line_from_file(me,iunit,line,status_ok)

    implicit none

    class(csv_file),intent(in) :: me
    integer,intent(in) :: iunit
    character(len=:),allocatable,intent(out) :: line
    logical,intent(out) :: status_ok

    integer :: nread
    integer :: istat
    character(len=me%chunk_size) :: buffer

    CALL setmessagesource("read_line_from_file")

    nread  = 0
    buffer = ''
    line   = ''
    status_ok = .true.

    do
        ! read in the next block of text from the line:
        read(iunit,fmt='(A)',advance='NO',size=nread,iostat=istat) buffer
        if (is_iostat_end(istat) .or. is_iostat_eor(istat)) then
            ! add the last block of text before the end of record
            if (nread>0) line = line//buffer(1:nread)
            exit
        else if (istat==0) then ! all the characters were read
```

```
            line = line//buffer  ! add this block of text to the string
        else  ! some kind of error
            WRITE(scratchmessage,'(A,1X,I5)') 'Read error for file unit: ', iunit
            CALL  allmessage(error, scratchmessage)
            status_ok = .false.
            exit
        end if
    end do

    CALL unsetmessagesource()

    end subroutine read_line_from_file
!*******************************************************************************************

!*******************************************************************************************
!

    pure subroutine split(str,token,chunk_size,vals)

    implicit none

    character(len=*),intent(in)  :: str
    character(len=*),intent(in)  :: token
    integer,intent(in)           :: chunk_size
    type(csv_string),dimension(:),allocatable,intent(out) :: vals

    integer :: i
    integer :: len_str
    integer :: len_token
    integer :: n_tokens
    integer :: i1
    integer :: i2
    integer :: j
    integer,dimension(:),allocatable :: itokens

    len_token = len(token)  ! length of the token
    n_tokens  = 0           ! initialize the token counter
    j         = 0           ! index to start looking for the next token

    ! first, count the number of times the token
    ! appears in the string, and get the token indices.
    !
    ! Examples:
    !  ',         '    --> 1
    !  '1234,67,90'    --> 5,8
    !  '123,      '    --> 4

    ! length of the string
    if (token == ' ') then
        ! in this case, we can't ignore trailing space
        len_str = len(str)
    else
        ! safe to ignore trailing space when looking for tokens
        len_str = len_trim(str)
    end if

    j = 1
    n_tokens = 0
    do
        if (j>len_str) exit      ! end of string, finished
        i = index(str(j:),token) ! index of next token in remaining string
        if (i<=0) exit           ! no more tokens found
        call expand_vector(itokens,n_tokens,chunk_size,i+j-1)  ! save the token location
        j = j + i + (len_token - 1)
    end do
    call expand_vector(itokens,n_tokens,chunk_size,finished=.true.)  ! resize the vector

    allocate(vals(n_tokens+1))

    if (n_tokens>0) then

        len_str = len(str)

        i1 = 1
        i2 = itokens(1)-1
        if (i2>=i1) then
            vals(1)%str = str(i1:i2)
        else
            vals(1)%str = ''  !the first character is a token
        end if

```

```
        !      1 2 3
        !    'a,b,c,d'

        do i=2,n_tokens
            i1 = itokens(i-1)+len_token
            i2 = itokens(i)-1
            if (i2>=i1) then
                vals(i)%str = str(i1:i2)
            else
                vals(i)%str = ''  !empty element (e.g., 'abc,,def')
            end if
        end do

        i1 = itokens(n_tokens) + len_token
        i2 = len_str
        if (itokens(n_tokens)+len_token<=len_str) then
            vals(n_tokens+1)%str = str(i1:i2)
        else
            vals(n_tokens+1)%str = ''  !the last character was a token
        end if

    else
        !no tokens present, so just return the original string:
        vals(1)%str = str
    end if

    end subroutine split
!*****************************************************************************************

!*****************************************************************************************
    end module csv_module
!*****************************************************************************************
```

9.6 csv_parameters.F90 File Reference

Various parameters.

Modules

- module csv_parameters

Variables

- integer(ip), parameter, public csv_parameters::max_real_str_len = 27
- integer(ip), parameter, public csv_parameters::maximum
- integer(ip), parameter, public csv_parameters::string
- integer(ip), parameter, public csv_parameters::length
- integer(ip), parameter, public csv_parameters::of
- integer(ip), parameter, public csv_parameters::a
- integer(ip), parameter, public csv_parameters::real
- integer(ip), parameter, public csv_parameters::number
- character(len= ∗), parameter, public csv_parameters::default_real_fmt = '(E27.17E4)'
- integer(ip), parameter, public csv_parameters::max_integer_str_len = 256

    default real number format statement (for writing real values to strings and files).
- integer(ip), parameter, public csv_parameters::an
- integer(ip), parameter, public csv_parameters::integer
- character(len= ∗), parameter, public csv_parameters::default_int_fmt = '(I256)'

9.6.1 Detailed Description

Various parameters.

Author

Jacob Williams

Copyright

License BSD

Definition in file csv_parameters.F90.

9.7 csv_parameters.F90

Go to the documentation of this file.

```
!--------------------------------------------------------------------
!                 M O D U L E   C S V _ P A R A M E T E R S
!--------------------------------------------------------------------
!--------------------------------------------------------------------

    module csv_parameters

    USE pahm_sizes, ONLY : wp, ip

    private

    integer(ip),parameter,public :: max_real_str_len = 27
    character(len=*),parameter,public :: default_real_fmt = '(E27.17E4)'

    integer(ip),parameter,public :: max_integer_str_len = 256
    character(len=*),parameter,public :: default_int_fmt  = '(I256)'

    end module csv_parameters
!*******************************************************************************
```

9.8 csv_utilities.F90 File Reference

Utility routines.

Modules

- module csv_utilities

Functions/Subroutines

- pure subroutine, public csv_utilities::expand_vector (vec, n, chunk_size, val, finished)

  Add elements to the integer vector in chunks.
- integer function, dimension(:), allocatable, public csv_utilities::unique (vec, chunk_size)

  Returns only the unique elements of the vector.
- subroutine, public csv_utilities::sort_ascending (ivec)

  Sorts an integer array `ivec` in increasing order. Uses a basic recursive quicksort (with insertion sort for partitions with $\leq$ 20 elements).

9.8.1 Detailed Description

Utility routines.

Author

Jacob Williams

Copyright

License BSD

Definition in file csv_utilities.F90.

9.9 csv_utilities.F90

Go to the documentation of this file.

```
!------------------------------------------------------------------
!              M O D U L E    C S V _ U T I L I T I E S
!------------------------------------------------------------------
!------------------------------------------------------------------

    module csv_utilities

    USE pahm_sizes, ONLY : wp, ip
    use csv_parameters

    private

    integer,parameter :: max_size_for_insertion_sort = 20

    public :: unique
    public :: expand_vector
    public :: sort_ascending

    contains
!*******************************************************************************

  !------------------------------------------------------------------
  ! S U B R O U T I N E   E X P A N D _ V E C T O R
  !------------------------------------------------------------------
  !------------------------------------------------------------------
    pure subroutine expand_vector(vec, n, chunk_size, val, finished)

    implicit none

    integer,dimension(:),allocatable,intent(inout) :: vec
    integer,intent(inout)       :: n           ! counter for last element added to `vec`.
                                               ! must be initialized to `size(vec)`
                                               ! (or 0 if not allocated) before first call
    integer,intent(in)          :: chunk_size  ! allocate `vec` in blocks of this size (>0)
    integer,intent(in),optional :: val         ! the value to add to `vec`
    logical,intent(in),optional :: finished    ! set to true to return `vec`
                                               ! as its correct size (`n`)

    integer,dimension(:),allocatable :: tmp  ! temporary array

    if (present(val)) then
        if (allocated(vec)) then
            if (n==size(vec)) then
                ! have to add another chunk:
                allocate(tmp(size(vec)+chunk_size))
                tmp(1:size(vec)) = vec
                call move_alloc(tmp,vec)
            end if
            n = n + 1
        else
```

```
            ! the first element:
            allocate(vec(chunk_size))
            n = 1
        end if
        vec(n) = val
    end if

    if (present(finished)) then
        if (finished) then
            ! set vec to actual size (n):
            if (allocated(tmp)) deallocate(tmp)
            allocate(tmp(n))
            tmp = vec(1:n)
            call move_alloc(tmp,vec)
        end if
    end if

    end subroutine expand_vector
!*****************************************************************************************

!*****************************************************************************************
!

    function unique(vec,chunk_size) result(ivec_unique)

    implicit none

    integer,dimension(:),intent(in)     :: vec
    integer,intent(in)                  :: chunk_size
    integer,dimension(:),allocatable    :: ivec_unique

    integer,dimension(size(vec)) :: ivec
    integer :: i
    integer :: n

    ! first we sort it:
    ivec = vec ! make a copy
    call sort_ascending(ivec)

    ! add the first element:
    n = 1
    ivec_unique = [ivec(1)]

    ! walk through array and get the unique ones:
    if (size(ivec)>1) then
        do i = 2, size(ivec)
            if (ivec(i)/=ivec(i-1)) then
                call expand_vector(ivec_unique,n,chunk_size,val=ivec(i))
            end if
        end do
        call expand_vector(ivec_unique,n,chunk_size,finished=.true.)
    end if

    end function unique
!*****************************************************************************************

!*****************************************************************************************

    subroutine sort_ascending(ivec)

    implicit none

    integer,dimension(:),intent(inout) :: ivec

    call quicksort(1,size(ivec))

    contains

        recursive subroutine quicksort(ilow,ihigh)


        implicit none

        integer,intent(in) :: ilow
        integer,intent(in) :: ihigh

        integer :: ipivot
        integer :: i
        integer :: j

        if ( ihigh-ilow<=max_size_for_insertion_sort .and. ihigh>ilow ) then
```

```

            ! do insertion sort:
            do i = ilow + 1,ihigh
                do j = i,ilow + 1,-1
                    if ( ivec(j) < ivec(j-1) ) then
                        call swap(ivec(j),ivec(j-1))
                    else
                        exit
                    end if
                end do
            end do

        else if ( ihigh-ilow>max_size_for_insertion_sort ) then

            ! do the normal quicksort:
            call partition(ilow,ihigh,ipivot)
            call quicksort(ilow,ipivot - 1)
            call quicksort(ipivot + 1,ihigh)

        end if

        end subroutine quicksort

        subroutine partition(ilow,ihigh,ipivot)


        implicit none

        integer,intent(in)  :: ilow
        integer,intent(in)  :: ihigh
        integer,intent(out) :: ipivot

        integer :: i,ip

        call swap(ivec(ilow),ivec((ilow+ihigh)/2))
        ip = ilow
        do i = ilow + 1, ihigh
            if ( ivec(i) < ivec(ilow) ) then
                ip = ip + 1
                call swap(ivec(ip),ivec(i))
            end if
        end do
        call swap(ivec(ilow),ivec(ip))
        ipivot = ip

        end subroutine partition

    end subroutine sort_ascending
!*******************************************************************************

!*******************************************************************************

    pure elemental subroutine swap(i1,i2)

    implicit none

    integer,intent(inout) :: i1
    integer,intent(inout) :: i2

    integer :: tmp

    tmp = i1
    i1  = i2
    i2  = tmp

    end subroutine swap
!*******************************************************************************

!*******************************************************************************
    end module csv_utilities
!*******************************************************************************
```

9.10 doxy_sample.F90 File Reference

9.11 doxy_sample.F90

Go to the documentation of this file.

```
finished: netcdfio.f90
!----------------------------------------------------------------
!               M O D U L E   S I Z E S
!----------------------------------------------------------------
!----------------------------------------------------------------



  !----------------------------------------------------------------
  ! F U N C T I O N   CompareDoubleReals
  !----------------------------------------------------------------
  !----------------------------------------------------------------
```

9.12 driver_mod.F90 File Reference

Modules

- module pahm_drivermod

Functions/Subroutines

- subroutine pahm_drivermod::getprogramcmdlargs ()

  Prints on the screen the help system of the PaHM program.
- subroutine pahm_drivermod::pahm_init ()

  Subroutine to initialize a PaHM run.
- subroutine pahm_drivermod::pahm_run (nTimeSTP)

  Subroutine to run PaHM (timestepping).
- subroutine pahm_drivermod::pahm_finalize ()

  Subroutine to finalize a PaHM run.

Variables

- integer, save pahm_drivermod::cnttimebegin
- integer, save pahm_drivermod::cnttimeend

9.12.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file driver_mod.F90.

9.13 driver_mod.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!               M O D U L E   P A H M  D R I V E R  M O D
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_drivermod

  USE pahm_messages
  USE utilities
  !USE TimeDateUtils

  IMPLICIT NONE

  INTEGER, SAVE :: cnttimebegin, cnttimeend


CONTAINS


  !-----------------------------------------------------------------------
  !     S U B R O U T I N E   G E T  P R O G R A M  C M D L  A R G S
  !-----------------------------------------------------------------------
  !-----------------------------------------------------------------------
  SUBROUTINE getprogramcmdlargs()

    USE pahm_global, ONLY : controlfilename

    IMPLICIT NONE

    INTEGER         :: argNumb, argCnt      ! number of command line arguments and argument counter
    CHARACTER(1024) :: argCmdLine

    CALL initlogging()

    argnumb = iargc()
    IF (argnumb > 0) THEN
      argcnt = 0
      DO WHILE (argcnt < argnumb)
        argcnt = argcnt + 1
        CALL getarg(argcnt, argcmdline)

        SELECT CASE(trim(argcmdline))
          CASE("-V", "-v", "--V", "--v", "--version")
            CALL programversion
            stop

          CASE("-H", "-h", "--H", "--h", "--help")
            CALL programhelp
            stop

          CASE DEFAULT
            ! Do nothing
        END SELECT
      END DO
      ! This is the first argument if not "-v/-h" were supplied.
      ! It is assumed that this argument is the filename of the user control file.
      CALL getarg(1, argcmdline)
      controlfilename = trim(adjustl(argcmdline))
    ENDIF

    CALL readcontrolfile(trim(controlfilename))

  END SUBROUTINE getprogramcmdlargs

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   P A H M  M O D E L  I N I T
  !----------------------------------------------------------------
  !-----------------------------------------------------------------------
  SUBROUTINE pahm_init()

    USE pahm_global, ONLY : noutdt
    USE pahm_mesh, ONLY : readmesh
    USE parwind, ONLY : readcsvbesttrackfile

```

```
    ! Initialize the logging system, needs to be called first
    CALL initlogging()

    CALL setmessagesource("PaHM_Init")

    ! Get possible command line arguments
    CALL getprogramcmdlargs()

    ! Read the mesh/grid of the domain or the generic mesh/grid input file
    CALL readmesh()

    ! Read all track files and save the data into the array of the best track structures
    ! for subsequent access by the P-W models in the program
    !CALL ReadBestTrackFile()
    CALL readcsvbesttrackfile()

    cnttimebegin = 1
    cnttimeend   = noutdt

    CALL unsetmessagesource()

  END SUBROUTINE pahm_init

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   P A H M  M O D E L  R U N
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE pahm_run(nTimeSTP)

    USE pahm_global, ONLY : modeltype, wvelx, wvely, wpress, times
    USE parwind, ONLY : gethollandfields
    USE pahm_netcdfio

    IMPLICIT NONE

    INTEGER, INTENT(IN), OPTIONAL :: nTimeSTP

    INTEGER :: iCnt

    CALL setmessagesource("PaHM_Run")

    IF (PRESENT(ntimestp)) THEN
      cnttimeend = cnttimebegin + ntimestp - 1
    ENDIF

    SELECT CASE (modeltype)
      CASE (1)
        DO icnt = cnttimebegin, cnttimeend
          CALL gethollandfields(icnt)
          IF (outfilenamespecified) THEN
            ! Create the output NetCDF file and fill it with the static data only
            ! Initialize all variables. This subroutine is called just once
            CALL initadcircnetcdfoutfile(outfilename)

            CALL writenetcdfrecord(outfilename, icnt)
          END IF
        END DO

      CASE DEFAULT
        WRITE(scratchmessage, '(a, i0)') &
                              'This model type is not supported: modelType = ', modeltype
        CALL logmessage(error, scratchmessage)
    END SELECT

    IF (PRESENT(ntimestp)) THEN
      cnttimebegin = cnttimeend + 1
    ENDIF

    CALL unsetmessagesource()

  END SUBROUTINE pahm_run

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   P A H M  M O D E L  F I N A L I Z E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE pahm_finalize()
```

```

   CALL setmessagesource("PaHM_Finalize")

   CALL unsetmessagesource()

   ! Close the logging facilities
   CALL closelogfile()

  END SUBROUTINE pahm_finalize

!================================================================================

END MODULE pahm_drivermod
```

9.14 global.F90 File Reference

Modules

- module pahm_global

Functions/Subroutines

- real(sz) function pahm_global::airdensity (atmT, atmP, relHum)

  This function calculates the density of the moist air.

Variables

- integer, parameter pahm_global::lun_screen = 6
- integer, parameter pahm_global::lun_ctrl = 10
- integer, parameter pahm_global::lun_inp = 14
- integer, parameter pahm_global::lun_inp1 = 15
- integer, parameter pahm_global::lun_log = 35
- integer, parameter pahm_global::lun_btrk = 22
- integer, parameter pahm_global::lun_btrk1 = 23
- integer, parameter pahm_global::lun_out = 25
- integer, parameter pahm_global::lun_out1 = 26
- real(sz), parameter pahm_global::defv_gravity = 9.80665_SZ
- real(sz), parameter pahm_global::defv_atmpress = 1013.25_SZ
- real(sz), parameter pahm_global::defv_rhoair = 1.1478_SZ
- real(sz), parameter pahm_global::defv_rhowater = 1000.0000
- real(sz), parameter pahm_global::one2ten = 0.8928_SZ
- real(sz), parameter pahm_global::ten2one = 1.0_SZ / 0.8928_SZ
- real(sz), parameter pahm_global::pi = 3.141592653589793_SZ
- real(sz), parameter pahm_global::deg2rad = PI / 180.0_SZ
- real(sz), parameter pahm_global::rad2deg = 180.0_SZ / PI
- real(sz), parameter pahm_global::basee = 2.718281828459045_SZ
- real(sz), parameter pahm_global::rearth = 6378206.4_SZ
- real(sz), parameter pahm_global::nm2m = 1852.0_SZ
- real(sz), parameter pahm_global::m2nm = 1.0_SZ / NM2M
- real(sz), parameter pahm_global::kt2ms = NM2M / 3600.0_SZ
- real(sz), parameter pahm_global::ms2kt = 1.0_SZ / KT2MS
- real(sz), parameter pahm_global::omega = 2.0_SZ ∗ PI / 86164.2_SZ

- real(sz), parameter pahm_global::mb2pa = 100.0_SZ
- real(sz), parameter pahm_global::mb2kpa = 0.1_SZ
- character(len=fnamelen) pahm_global::logfilename = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) pahm_global::controlfilename = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_control.in'
- logical pahm_global::meshfilenamespecified = .FALSE.
- character(len=fnamelen) pahm_global::meshfilename = BLANK
- character(len=64) pahm_global::meshfiletype = BLANK
- character(len=64) pahm_global::meshfileform = BLANK
- logical pahm_global::besttrackfilenamespecified = .FALSE.
- integer pahm_global::nbtrfiles = IMISSV
- character(len=fnamelen), dimension(:), allocatable pahm_global::besttrackfilename
- character(len=512) pahm_global::title = BLANK
- real(sz) pahm_global::gravity = DEFV_GRAVITY
- real(sz) pahm_global::rhowater = DEFV_RHOWATER
- real(sz) pahm_global::rhoair = DEFV_RHOAIR
- real(sz) pahm_global::backgroundatmpress = DEFV_ATMPRESS
- real(sz), parameter pahm_global::defv_bladjustfac = 0.9_SZ
- real(sz) pahm_global::windreduction = DEFV_BLADJUSTFAC
- real(sz) pahm_global::bladjustfac = DEFV_BLADJUSTFAC
- character(len=64) pahm_global::refdatetime = BLANK
- integer pahm_global::refdate = IMISSV
- integer pahm_global::reftime = IMISSV
- integer pahm_global::refyear = IMISSV
- integer pahm_global::refmonth = 0
- integer pahm_global::refday = 0
- integer pahm_global::refhour = 0
- integer pahm_global::refmin = 0
- integer pahm_global::refsec = 0
- logical pahm_global::refdatespecified = .FALSE.
- character(len=64) pahm_global::begdatetime = BLANK
- integer pahm_global::begdate = IMISSV
- integer pahm_global::begtime = IMISSV
- integer pahm_global::begyear = IMISSV
- integer pahm_global::begmonth = 0
- integer pahm_global::begday = 0
- integer pahm_global::beghour = 0
- integer pahm_global::begmin = 0
- integer pahm_global::begsec = 0
- logical pahm_global::begdatespecified = .FALSE.
- character(len=64) pahm_global::enddatetime = BLANK
- integer pahm_global::enddate = IMISSV
- integer pahm_global::endtime = IMISSV
- integer pahm_global::endyear = IMISSV
- integer pahm_global::endmonth = 0
- integer pahm_global::endday = 0
- integer pahm_global::endhour = 0
- integer pahm_global::endmin = 0
- integer pahm_global::endsec = 0
- logical pahm_global::enddatespecified = .FALSE.
- real(sz) pahm_global::begsimtime = RMISSV
- real(sz) pahm_global::endsimtime = RMISSV

- logical pahm_global::begsimspecified = .FALSE.
- logical pahm_global::endsimspecified = .FALSE.
- character(len=1) pahm_global::unittime = 'S'
- real(sz) pahm_global::outdt = RMISSV
- integer pahm_global::noutdt = IMISSV
- real(sz) pahm_global::mdoutdt = RMISSV
- real(sz) pahm_global::mdbegsimtime = RMISSV
- real(sz) pahm_global::mdendsimtime = RMISSV
- logical pahm_global::outfilenamespecified = .FALSE.
- character(len=fnamelen) pahm_global::outfilename = BLANK
- integer pahm_global::ncshuffle = 0
- integer pahm_global::ncdeflate = 0
- integer pahm_global::ncdlevel = 0
- character(len=20), parameter pahm_global::def_ncnam_pres = 'P'
- character(len=20), parameter pahm_global::def_ncnam_wndx = 'uwnd'
- character(len=20), parameter pahm_global::def_ncnam_wndy = 'vwnd'
- character(len=20) pahm_global::ncvarnam_pres = DEF_NCNAM_PRES
- character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAM_WNDX
- character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAM_WNDY
- integer pahm_global::modeltype = IMISSV
- logical pahm_global::writeparams = .FALSE.
- real(sz), dimension(:), allocatable pahm_global::wvelx
- real(sz), dimension(:), allocatable pahm_global::wvely
- real(sz), dimension(:), allocatable pahm_global::wpress
- real(sz), dimension(:), allocatable pahm_global::times

9.14.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file global.F90.

9.15 global.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                M O D U L E   G L O B A L
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_global

  USE version
  USE pahm_sizes

  IMPLICIT NONE

!################################################################
!###   BEG:: LUN NUMBERS FOR I/O OPERATIONS
!################################################################
  INTEGER, PARAMETER :: lun_screen =  6   ! I/O unit where screen output is sent
  INTEGER, PARAMETER :: lun_ctrl   = 10   ! I/O unit for the model's control file
  INTEGER, PARAMETER :: lun_inp    = 14   ! I/O unit for the input files (mesh)
```

```
  INTEGER, PARAMETER :: lun_inp1    = 15   ! I/O unit for the input files (mesh)
  INTEGER, PARAMETER :: lun_log     = 35   ! I/O unit where log output is sent
  INTEGER, PARAMETER :: lun_btrk    = 22   ! I/O unit for the best track files
  INTEGER, PARAMETER :: lun_btrk1   = 23   ! I/O unit for the best track files
  INTEGER, PARAMETER :: lun_out     = 25   ! I/O unit for the output files
  INTEGER, PARAMETER :: lun_out1    = 26   ! I/O unit for the output files
!################################################################
!###   END:: LUN NUMBERS FOR I/O OPERATIONS
!################################################################


!################################################################
!###   BEG:: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
!################################################################
  REAL(sz), PARAMETER :: defv_gravity  = 9.80665_sz    ! Default (standard) gravitational acceleration (m/s^2)
  REAL(sz), PARAMETER :: defv_atmpress = 1013.25_sz    ! Default (standard) atmospheric pressure (mb)

  REAL(sz), PARAMETER :: defv_rhoair   = 1.1478_sz      ! Default (standard) density of air at STP (kg/m^3)
                                                         ! 1.1478 (1013.25 mb, Rel. Hum  90%, 30 deg C)

 ! Water density is used in the code to convert the pressure to units of mH2O
  REAL(sz), PARAMETER :: defv_rhowater = 1000.0000    ! Default density of fresh water (kg/m^3)
                                                     !--- FRESH WATER
                                                        !  999.8900 ( 0 deg C)
                                                        ! 1000.0000 ( 4 deg C)
                                                        !  999.7025 (10 deg C)
                                                        !  999.1026 (15 deg C)
                                                        !  998.2072 (20 deg C)
                                                        !  997.0476 (25 deg C)
                                                        !  995.6495 (30 deg C)
                                                        !  994.0333 (35 deg C)
                                                        !  992.2164 (40 deg C)
                                                     !--- SEA WATER
                                                        ! 1028.0941 (35% S,  1 deg C)
                                                        ! 1027.8336 (35% S,  4 deg C)
                                                        ! 1027.0000 (35% S, 10 deg C)
                                                        ! 1026.0210 (35% S, 15 deg C)
                                                        ! 1024.8103 (35% S, 20 deg C)
                                                        ! 1023.3873 (35% S, 25 deg C)
                                                        ! 1021.7694 (35% S, 30 deg C)
                                                        ! 1019.9000 (35% S, 35 deg C)
                                                        ! 1018.0000 (35% S, 40 deg C)

  ! 1-min to 10-min wind conversion factors
  REAL(sz), PARAMETER :: one2ten = 0.8928_sz
  REAL(sz), PARAMETER :: ten2one = 1.0_sz / 0.8928_sz

  REAL(sz), PARAMETER :: pi = 3.141592653589793_sz
  REAL(sz), PARAMETER :: deg2rad = pi / 180.0_sz       ! degrees to radians
  REAL(sz), PARAMETER :: rad2deg = 180.0_sz / pi       ! radians to degrees
  REAL(sz), PARAMETER :: basee = 2.718281828459045_sz  ! mathematical constant e (natural logarithm base)

  REAL(sz), PARAMETER :: rearth   = 6378206.4_sz         ! radius of earth (m) (Clarke 1866 major spheroid radius)
  REAL(sz), PARAMETER :: nm2m     = 1852.0_sz            ! nautical miles to meters
  REAL(sz), PARAMETER :: m2nm     = 1.0_sz / nm2m        ! meters to nautical miles
  REAL(sz), PARAMETER :: kt2ms    = nm2m / 3600.0_sz     ! knots to m/s
  REAL(sz), PARAMETER :: ms2kt    = 1.0_sz / kt2ms       ! m/s to knots
  REAL(sz), PARAMETER :: omega    = 2.0_sz * pi / 86164.2_sz
  REAL(sz), PARAMETER :: mb2pa    = 100.0_sz
  REAL(sz), PARAMETER :: mb2kpa   = 0.1_sz
!################################################################
!###   END:: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
!################################################################


!################################################################
!###   BEG :: VARIABLES RELATED TO THE CONTROL FILE
!################################################################
  CHARACTER(LEN=FNAMELEN) :: logfilename = trim(adjustl(prog_name_low)) // '_model.log'

  !-------------------- Input files
  CHARACTER(FNAMELEN)      :: controlfilename = trim(adjustl(prog_name_low)) // '_control.in'  ! default value

  LOGICAL                  :: meshfilenamespecified = .false.      ! .TRUE. if the user supplied a valid filename
  CHARACTER(LEN=FNAMELEN)  :: meshfilename = blank                 ! there is no default value here
  CHARACTER(LEN=64)        :: meshfiletype = blank                 ! ADCIRC, SCHISM, FVCOM, ROMS, GENERIC
```

```
(no default)
  CHARACTER(LEN=64)        :: meshfileform = blank                ! ASCII, NETCDF (no default)

  LOGICAL                              :: besttrackfilenamespecified = .false.
  INTEGER                              :: nbtrfiles = imissv
  CHARACTER(LEN=FNAMELEN), ALLOCATABLE :: besttrackfilename(:)
  !--------------------

  !-------------------- Other parameters in the control file
  CHARACTER(LEN=512)       :: title = blank

  REAL(sz)                 :: gravity             = defv_gravity    ! m/s^2   Gravitational acceleration
  REAL(sz)                 :: rhowater            = defv_rhowater   ! kg/m^3  Mean water density
  REAL(sz)                 :: rhoair              = defv_rhoair     ! kg/m^3  Mean air density
  REAL(sz)                 :: backgroundatmpress = defv_atmpress    ! mb      Background atmospheric pressure

  ! This is for the BL reduction factor used in the Holland model
  REAL(sz), PARAMETER      :: defv_bladjustfac = 0.9_sz
  REAL(sz)                 :: windreduction    = defv_bladjustfac  ! PV BL reduction factor used in the Holland model
  REAL(sz)                 :: bladjustfac      = defv_bladjustfac   !PV same as windReduction?

  !====================
  !=== This block is for the : time/date and time stepping variables
  !====================
  !---
  ! the reference date/time for the model run YYYYMMDDhhmmss
  CHARACTER(LEN=64)        :: refdatetime      = blank
  INTEGER                  :: refdate          = imissv
  INTEGER                  :: reftime          = imissv
  INTEGER                  :: refyear          = imissv
  INTEGER                  :: refmonth         = 0
  INTEGER                  :: refday           = 0
  INTEGER                  :: refhour          = 0
  INTEGER                  :: refmin           = 0
  INTEGER                  :: refsec           = 0
  LOGICAL                  :: refdatespecified = .false.
  !---
  ! the start date/time for the model run YYYYMMDDhhmmss
  CHARACTER(LEN=64)        :: begdatetime      = blank
  INTEGER                  :: begdate          = imissv
  INTEGER                  :: begtime          = imissv
  INTEGER                  :: begyear          = imissv
  INTEGER                  :: begmonth         = 0
  INTEGER                  :: begday           = 0
  INTEGER                  :: beghour          = 0
  INTEGER                  :: begmin           = 0
  INTEGER                  :: begsec           = 0
  LOGICAL                  :: begdatespecified = .false.
  !---
  ! the stop date/time for the model run YYYYMMDDhhmmss
  CHARACTER(LEN=64)        :: enddatetime      = blank
  INTEGER                  :: enddate          = imissv
  INTEGER                  :: endtime          = imissv
  INTEGER                  :: endyear          = imissv
  INTEGER                  :: endmonth         = 0
  INTEGER                  :: endday           = 0
  INTEGER                  :: endhour          = 0
  INTEGER                  :: endmin           = 0
  INTEGER                  :: endsec           = 0
  LOGICAL                  :: enddatespecified = .false.
  !---
  ! alternative definitions for the stop date/time for the model run
  REAL(sz)                 :: begsimtime      = rmissv
  REAL(sz)                 :: endsimtime      = rmissv
  LOGICAL                  :: begsimspecified = .false.
  LOGICAL                  :: endsimspecified = .false.

  CHARACTER(LEN=1)         :: unittime = 'S'
  !====================

  !---
  ! time stepping variables for the model run
  REAL(sz)                 :: outdt        = rmissv
  INTEGER                  :: noutdt       = imissv
  REAL(sz)                 :: mdoutdt      = rmissv
  REAL(sz)                 :: mdbegsimtime = rmissv
  REAL(sz)                 :: mdendsimtime = rmissv

  LOGICAL                  :: outfilenamespecified = .false.
```

```fortran
  CHARACTER(LEN=FNAMELEN) :: outfilename = blank   ! Name of the output NetCDF file
  INTEGER                 :: ncshuffle = 0         ! Turn on the shuffle filter (>0)
  INTEGER                 :: ncdeflate = 0         ! Turn on the deflate filter (>0)
  INTEGER                 :: ncdlevel  = 0         ! Deflate level [0-9]

  ! Create a list of NetCDF variable names in the form ncYyyyVarNam = value
  ! The user can specify his/her own values in the control file (will be hidden variables)
  ! Default values
  CHARACTER(LEN=20), PARAMETER :: def_ncnam_pres = 'P',      &
                                  def_ncnam_wndx = 'uwnd',   &
                                  def_ncnam_wndy = 'vwnd'

  CHARACTER(LEN=20)            :: ncvarnam_pres = def_ncnam_pres, &
                                  ncvarnam_wndx = def_ncnam_wndx, &
                                  ncvarnam_wndy = def_ncnam_wndy

  INTEGER                 :: modeltype = imissv    ! The parametric model to use
                                                   !  0: Rankin Vortex
                                                   !  1: Holland B (1998)
                                                   !  2: Holland B (2010)
                                                   !  3: Willoughby model - NOT IMPLEMENTED YET
                                                   !  9: Assymetric vortex model (Mattocks)
                                                   ! 10: Generalized assymetric vortex Holland model
  LOGICAL                 :: writeparams = .false.
!#################################################################
!###   END :: VARIABLES RELATED TO THE CONTROL FILE
!#################################################################


!#################################################################
!###   BEG :: GLOBAL DATA ARRAYS
!#################################################################
  ! Arrays to hold the P-W fields
  !REAL(SZ), DIMENSION(:, :), ALLOCATABLE :: wVelX, wVelY, wPress
  REAL(sz), DIMENSION(:), ALLOCATABLE :: wvelx, wvely, wpress
  REAL(sz), DIMENSION(:), ALLOCATABLE :: times
!#################################################################
!###   END :: GLOBAL DATA ARRAYS
!#################################################################


  CONTAINS


  !----------------------------------------------------------------
  !  F U N C T I O N   A I R  D E N S I T Y
  !----------------------------------------------------------------
  ! >@see http://www.emd.dk/files/windpro/WindPRO_AirDensity.pdf
  !----------------------------------------------------------------
  REAL(sz) function airdensity(atmt, atmp, relhum) result(myvalout)

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: atmt      ! Surface temperature in degrees C (-50.0 <= T <= 100.0)
    REAL(sz), INTENT(IN) :: atmp      ! Atmospheric pressure (mb)
    REAL(sz), INTENT(IN) :: relhum    ! Relative humidity (0 - 100)

    ! Local variables
    REAL(hp)             :: es, p, pv, pd, rh
    REAL(hp)             :: rd, rv, temp, tempk, dens


    rh = relhum
    IF (rh < 0.01)  rh = 0.01_hp
    IF (rh > 100.0) rh = 100.0_hp

    temp = atmt
    IF (temp < -50.0) temp = -50.0_hp
    IF (temp > 100.0) temp = 100.0_hp

    rd = 287.058_hp ! specific gas constant for dry air (J/kg*K)
    rv = 461.495_hp ! specific gas constant for water vapor (J/kg*K)

    ! Convert relative humidity to %
    rh = 0.01_sz * rh

    ! Convert atmT (C) to K
    tempk = temp + 273.15_hp

    ! Calculate the saturated vapor pressure (mb)
    ! Temperature is in degrees Celcius
```

```
    p = 0.99999683e+00_hp + temp * (-0.90826951e-02_hp + temp * (0.78736169e-04_hp + temp * &
                                   (-0.61117958e-06_hp + temp * (0.43884187e-08_hp + temp * &
                                   (-0.29883885e-10_hp + temp * (0.21874425e-12_hp + temp * &
                                   (-0.17892321e-14_hp + temp * (0.11112018e-16_hp + temp * &
                                   (-0.30994571e-19_hp)))))))))
    es = 6.1078_hp / p**8   ! saturated vapour pressure (mb)

    ! Calculate the actual vapor pressure (mb)
    pv = es * rh

    ! Calculate the actual vapor pressure
    pd = atmp - pv

    ! Convert the pressures from mb to Pa
    pd = pd * 100.0_hp
    pv = pv * 100.0_hp

    ! Calculate the air density
    dens = pd / (rd * tempk) + pv / (rv * tempk)

    myvalout = dens

    RETURN

  END FUNCTION airdensity

!================================================================================

END MODULE pahm_global
```

9.16 mesh.F90 File Reference

Contains all the mesh related utilities.

Modules

- module pahm_mesh

Functions/Subroutines

- subroutine pahm_mesh::readmesh ()

  Reads an input mesh file for the specified supported model type.
- subroutine pahm_mesh::readmeshasciifort14 ()

  Reads the ADCIRC fort.14 mesh file.
- subroutine pahm_mesh::allocatenodalandelementalarrays ()

  Allocates memory to mesh arrays.

Variables

- character(len=80) pahm_mesh::agrid
- integer pahm_mesh::np = IMISSV
- integer pahm_mesh::ne = IMISSV
- integer pahm_mesh::ics
- real(sz), dimension(:), allocatable pahm_mesh::dp
- integer, dimension(:), allocatable pahm_mesh::nfn
- integer, dimension(:, :), allocatable pahm_mesh::nm
- real(sz), dimension(:), allocatable pahm_mesh::slam

- real(sz), dimension(:), allocatable pahm_mesh::sfea
- real(sz), dimension(:), allocatable pahm_mesh::xcslam
- real(sz), dimension(:), allocatable pahm_mesh::ycsfea
- real(sz) pahm_mesh::slam0 = RMISSV
- real(sz) pahm_mesh::sfea0 = RMISSV
- integer, parameter pahm_mesh::maxfacenodes = 5
- logical pahm_mesh::ismeshok = .FALSE.

9.16.1 Detailed Description

Contains all the mesh related utilities.

Created this mesh module in order to modularize mesh related data. Modularity gives us greater flexibility in reading meshes in different file formats (such as NetCDF or XDMF) or even to read meshes that were originally developed and formatted for other unstructured mesh models (such as DG ADCIRC, RiCOM, FVCOM, SUNTANS, or unstructured SWAN).

The variables and subroutines in this module were refactored out of the other parts of the code, particularly from the global module.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Note

Adopted from the ADCIRC source code.

Definition in file mesh.F90.

9.17 mesh.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!               M O D U L E   M E S H
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_mesh

  USE pahm_sizes
  USE pahm_messages

  IMPLICIT NONE

  CHARACTER(LEN=80)        :: agrid
  INTEGER                  :: np = imissv    ! number of nodes in the mesh
  INTEGER                  :: ne = imissv    ! number of elements in the mesh
  INTEGER                  :: ics            ! mesh coordinate system (1=cartesian, 2=geographic)
  REAL(sz), ALLOCATABLE    :: dp(:)          ! bathymetric depth
  INTEGER, ALLOCATABLE     :: nfn(:)         ! element number of face nodes (ne)
  INTEGER, ALLOCATABLE     :: nm(:, :)       ! element table size(ne, nfn)
  REAL(sz), ALLOCATABLE    :: slam(:)        ! longitude node locations in CPP slam(np)
  REAL(sz), ALLOCATABLE    :: sfea(:)        ! latitude node locations in CPP sfea(np)
  REAL(sz), ALLOCATABLE    :: xcslam(:)      ! x cartesian node locations xcSlam(np)
  REAL(sz), ALLOCATABLE    :: ycsfea(:)      ! y cartesian node locations ycSfea(np)

```

```fortran
REAL(sz)                 :: slam0 = rmissv ! center point of CPP spherical projection
REAL(sz)                 :: sfea0 = rmissv ! center point of CPP spherical projection

! The maximum number of faces of an element
INTEGER, PARAMETER       :: maxfacenodes = 5

! This varibale is set to .TRUE. if the mesh file read successfully
LOGICAL                  :: ismeshok = .false.


CONTAINS


!----------------------------------------------------------------
!  S U B R O U T I N E   R E A D  M E S H
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE readmesh()

  USE pahm_global, ONLY : meshfilenamespecified, meshfilename, meshfiletype, meshfileform
  USE utilities, ONLY : touppercase

  IMPLICIT NONE


  CALL setmessagesource("ReadMesh")

  IF (meshfilenamespecified .EQV. .false.) THEN
    WRITE(scratchmessage, '(a)') 'ReadMesh: First specify a valid grid filename to proceed: ' // &
                                 '[' // trim(meshfilename) // ']'
    CALL allmessage(error, scratchmessage)
    CALL terminate()
  END IF

  SELECT CASE(touppercase(meshfiletype))
    !----- ADCIRC case
    CASE('ADCIRC')
      SELECT CASE(touppercase(meshfileform))
        CASE('ASCII')
          CALL readmeshasciifort14()

        CASE('NETCDF')
          WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file type: ' // &
                                       '[' // trim(meshfiletype) // ']'
          CALL allmessage(error, scratchmessage)
          CALL terminate()

        CASE DEFAULT
          WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the mesh file type: ' // &
                                       '[' // trim(meshfiletype) // ']'
          CALL allmessage(error, scratchmessage)
          CALL terminate()
      END SELECT

    !----- SCHISM case
    CASE('SCHISM')
      SELECT CASE(touppercase(meshfileform))
        CASE('ASCII')
          CALL readmeshasciifort14()

        CASE('NETCDF')
          WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file type: ' // &
                                       '[' // trim(meshfiletype) // ']'
          CALL allmessage(error, scratchmessage)
          CALL terminate()

        CASE DEFAULT
          WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the mesh file type: ' // &
                                       '[' // trim(meshfiletype) // ']'
          CALL allmessage(error, scratchmessage)
          CALL terminate()
      END SELECT

    !----- FVCOM case
    CASE('FVCOM')
      WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
                                   '[' // trim(meshfiletype) // ']'
```

```
        CALL allmessage(error, scratchmessage)
        CALL terminate()

      !----- ROMS case
      CASE('ROMS')
        WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
                                     '[' // trim(meshfiletype) // ']'
        CALL allmessage(error, scratchmessage)
        CALL terminate()

      !----- GENERIC case
      CASE('GENERIC')
        WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
                                     '[' // trim(meshfiletype) // ']'
        CALL allmessage(error, scratchmessage)
        CALL terminate()

      CASE DEFAULT
        WRITE(scratchmessage, '(a)') 'ReadMesh: Invalid mesh file type specified: ' // &
                                     '[' // trim(meshfiletype) // ']'
        CALL allmessage(error, scratchmessage)
        CALL terminate()
    END SELECT

    CALL unsetmessagesource()

  END SUBROUTINE readmesh

!=================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   R E A D  M E S H  A S C I I  F O R T  1 4
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE readmeshasciifort14()

    USE pahm_global,  ONLY : lun_inp, meshfilename
    USE utilities   !PV specify what are we using here from utilities

    IMPLICIT NONE

    INTEGER, PARAMETER :: iUnit = lun_inp        ! LUN for read operations
    INTEGER            :: ios                    ! I/O status
    CHARACTER(LEN=512) :: fmtStr                 ! String to hold formats for I/O
    INTEGER            :: lineNum                ! Line number currently being read

    INTEGER            :: labNodes, numFNodes    ! Label and number of nodal faces for that label
    INTEGER            :: iCnt                   ! Counters


    CALL setmessagesource("ReadMeshASCIIFort14")

    CALL openfileforread(iunit, trim(meshfilename), ios)

    linenum = 1

    READ(unit=iunit, fmt='(a80)', err=10, END=20, IOSTAT=ios) agrid
    linenum = linenum + 1

    CALL logmessage(info, "Reading the mesh file: " // trim(meshfilename))
    CALL logmessage(info, "Mesh file comment line: " // trim(agrid))
    CALL logmessage(info, "Reading mesh file dimensions and coordinates.")

    READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) ne, np
    linenum = linenum + 1

    CALL allocatenodalandelementalarrays()

    ! N O D E   T A B L E
    DO icnt = 1, np
      READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, slam(iCnt), sfea(iCnt), dp(iCnt)

      ! Check for (invalid longitude, latitude) values.
      ! Currently only geographical coordinates are supported.
      IF (.NOT. ((slam(icnt) >= -180.0_sz) .AND. (slam(icnt) <= 180.0_sz)) .OR.   &
           .NOT. ((sfea(icnt) >= -90.0_sz) .AND. (sfea(icnt) <= 90.0_sz))) THEN

        fmtstr = '("Input file: ' // trim(meshfilename) // '", ", line ", i0,'
          fmtstr = trim(fmtstr) //  ' " contains invalid (lon, lat) values: ", " [", f14.4, ",  ", f14.4, "]" '
          fmtstr = trim(fmtstr) //  ' " (should be degrees east and degrees north)")'
```

```
        WRITE(scratchmessage, trim(fmtstr)) linenum, slam(icnt), sfea(icnt)

        CALL allmessage(error, scratchmessage)
        CLOSE(iunit)
        CALL terminate()
      END IF

      linenum = linenum + 1
    END DO

    ! E L E M E N T   T A B L E
    DO icnt = 1, ne
      READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, numfnodes

      ! Check if numFNodes in the line is beyond the value of parameter MAXFACENODES,
      ! to avoid out of bounds errors for the array "nm".
      IF (numfnodes > maxfacenodes) THEN
        fmtstr = '("Input file: ' // trim(meshfilename) // '", ", reading line ", i0,'
          fmtstr = trim(fmtstr) //  ' " gave a number of face nodes equal to: ", i0, '
          fmtstr = trim(fmtstr) //  ' ", which is greater than MAXFACENODES")'
        WRITE(scratchmessage, trim(fmtstr)) linenum, numfnodes

        CALL allmessage(error, scratchmessage)
        CLOSE(iunit)
        CALL terminate()
      ELSE
        backspace(unit=iunit)
      END IF

      READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, nfn(iCnt), nm(iCnt, 1:nfn(iCnt))

      linenum = linenum + 1
    END DO

    CLOSE(iunit)

    !PV Need to also check if arrays contain any missing values
    IF ((comparereals(slam0, rmissv) == 0) .OR. &
        (comparereals(sfea0, rmissv) == 0)) THEN
      slam0 = sum(slam, 1) / np
      sfea0 = sum(sfea, 1) / np
    END IF

    CALL geotocpp(sfea, slam, sfea0, slam0, xcslam, ycsfea)

    CALL logmessage(info, 'Finished reading mesh file dimensions and coordinates.')

    CALL unsetmessagesource()

    ismeshok = .true.

    RETURN

    ! Jump to here on error condition during read
    10 fmtstr = '("Reading line ", i0, " gave the following error code: ", i0, ".")'
    WRITE(scratchmessage, fmtstr) linenum, ios

    CALL allmessage(error, scratchmessage)
    CLOSE(iunit)
    CALL terminate()

    ! Jump to here on end condition during read
    20 fmtstr = '("Reached premature end of file on line ", i0, ".")'
    WRITE(scratchmessage, trim(fmtstr)) linenum

    CALL allmessage(error, scratchmessage)
    CLOSE(iunit)
    CALL terminate()

  END SUBROUTINE readmeshasciifort14

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   A L L O C A T E   N O D A L   A N D   E L E M E N T A L   A R R A Y S
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE allocatenodalandelementalarrays()

    IMPLICIT NONE

```

```
    CALL setmessagesource("AllocateNodalAndElementalArrays")

    ALLOCATE(slam(np), sfea(np), xcslam(np), ycsfea(np), dp(np))

    ALLOCATE(nfn(ne), nm(ne, maxfacenodes))

    ! Initialize to something troublesome to make it easy to spot issues
    slam   = rmissv
    sfea   = rmissv
    xcslam = rmissv
    ycsfea = rmissv
    dp     = rmissv
    nm     = imissv
    nfn    = imissv

    CALL unsetmessagesource()

  END SUBROUTINE allocatenodalandelementalarrays

!================================================================================


END MODULE pahm_mesh

```

9.18 messages.F90 File Reference

Data Types

- interface pahm_messages::logmessage
- interface pahm_messages::screenmessage
- interface pahm_messages::allmessage

Modules

- module pahm_messages

Functions/Subroutines

- subroutine pahm_messages::initlogging ()

  Initializes logging levels.
- subroutine pahm_messages::openlogfile ()

  Opens the log file for writting.
- subroutine pahm_messages::closelogfile ()

  Closes an opened log file.
- subroutine pahm_messages::screenmessage_1 (message)

  General purpose subroutine to write a message to the screen.
- subroutine pahm_messages::screenmessage_2 (level, message)
- subroutine pahm_messages::logmessage_1 (message)

  General purpose subroutine to write a message to the log file.
- subroutine pahm_messages::logmessage_2 (level, message)
- subroutine pahm_messages::allmessage_1 (message)

  General purpose subroutine to write a message to both the screen and the log file.
- subroutine pahm_messages::allmessage_2 (level, message)
- subroutine pahm_messages::setmessagesource (source)

Sets the name of the subroutine that is writing log and/or screen messages.

- subroutine pahm_messages::unsetmessagesource ()

  Removes the name of the subroutine that is no longer active.
- subroutine pahm_messages::programversion ()

  Prints on the screen the versioning information of the program.
- subroutine pahm_messages::programhelp ()

  Prints on the screen the help system of the program.
- subroutine pahm_messages::terminate ()

  Terminates the calling program when a fatal error is encountered.

Variables

- integer pahm_messages::nscreen = 1
- integer, parameter pahm_messages::debug = -1
- integer, parameter pahm_messages::echo = 0
- integer, parameter pahm_messages::info = 1
- integer, parameter pahm_messages::warning = 2
- integer, parameter pahm_messages::error = 3
- character(len=10), dimension(5) pahm_messages::loglevelnames
- character(len=50), dimension(100) pahm_messages::messagesources
- character(len=1024) pahm_messages::scratchmessage
- character(len=1024) pahm_messages::scratchformat
- integer pahm_messages::sourcenumber
- logical pahm_messages::logfileopened = .FALSE.
- logical pahm_messages::loginitcalled = .FALSE.

9.18.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Note

Adopted from the ADCIRC source code.

Definition in file messages.F90.

9.19 messages.F90

Go to the documentation of this file.

```
!------------------------------------------------------------------
!                M O D U L E   M E S S A G E S
!------------------------------------------------------------------
!------------------------------------------------------------------

MODULE pahm_messages

  USE pahm_sizes, ONLY : fnamelen
  USE pahm_global, ONLY : lun_screen, lun_log, logfilename

#ifdef __INTEL_COMPILER
  USE ifport
#endif

  IMPLICIT NONE

  INTEGER                            :: nscreen = 1       ! >= 1: write to screen, <=0 do not write to screen

  ! Logging levels
  INTEGER, PARAMETER                 :: debug   = -1     ! write all messages and echo input
  INTEGER, PARAMETER                 :: echo    =  0     ! echo input, plus write all non-debug
  INTEGER, PARAMETER                 :: info    =  1     ! don't echo input; write all non-debug
  INTEGER, PARAMETER                 :: warning =  2     ! don't echo input; write only warn/err
  INTEGER, PARAMETER                 :: error   =  3     ! don't echo input; only fatal msgs

  CHARACTER(LEN=10), DIMENSION(5)    :: loglevelnames
  CHARACTER(LEN=50), DIMENSION(100) :: messagesources    ! subroutine names
  CHARACTER(LEN=1024)                :: scratchmessage    ! used for formatted messages
  CHARACTER(LEN=1024)                :: scratchformat     ! used for Fortran format strings
  INTEGER                            :: sourcenumber      ! index into messageSources for current sub

  ! Logging flags
  LOGICAL                            :: logfileopened = .false.
  LOGICAL                            :: loginitcalled = .false.

  !----------------------------------------------------------------------
  ! I N T E R F A C E S
  !----------------------------------------------------------------------
  INTERFACE logmessage
    MODULE PROCEDURE logmessage_1
    MODULE PROCEDURE logmessage_2
  END INTERFACE logmessage

  INTERFACE screenmessage
    MODULE PROCEDURE screenmessage_1
    MODULE PROCEDURE screenmessage_2
  END INTERFACE screenmessage

  INTERFACE allmessage
    MODULE PROCEDURE allmessage_1
    MODULE PROCEDURE allmessage_2
  END INTERFACE allmessage
  !----------------------------------------------------------------------


  CONTAINS


  !-------------------------------------------------------------------
  !     S U B R O U T I N E   I N I T   L O G G I N G
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE initlogging()

    IMPLICIT NONE

    IF (loginitcalled .EQV. .false.) THEN
      sourcenumber     = 0
      loglevelnames(1) = "DEBUG"
      loglevelnames(2) = "ECHO"
      loglevelnames(3) = "INFO"
      loglevelnames(4) = "WARNING"
      loglevelnames(5) = "ERROR"

      loginitcalled = .true.
```

```

      CALL openlogfile
    END IF

  END SUBROUTINE initlogging

!===============================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E    O P E N   L O G   F I L E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE openlogfile()

    IMPLICIT NONE

    INTEGER :: errorIO   ! zero if the file opened successfully

    logfileopened = .false.

    OPEN(unit=lun_log, file=trim(adjustl(logfilename)), action='WRITE', status='REPLACE', iostat=errorio)

    IF (errorio == 0) THEN
      logfileopened = .true.
    ELSE
      WRITE(scratchmessage, '(a, i0, a, i0)')                                         &
                          'Could not open the log file = ' // trim(adjustl(logfilename)) //   &
                          ' on logical unit LUN_LOG = ', lun_log,                      &
                          '. Error code was: errorIO = ', errorio
      CALL screenmessage(error, scratchmessage)
    END IF

  END SUBROUTINE openlogfile

!===============================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E    C L O S E   L O G   F I L E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE closelogfile()

    IMPLICIT NONE

    IF (logfileopened) CLOSE(unit=lun_log)

  END SUBROUTINE closelogfile

!===============================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E    S C R E E N  M E S S A G E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE screenmessage_1(message)

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), INTENT(IN) :: message

    IF (nscreen > 0) THEN
      IF (loginitcalled) THEN
        WRITE(lun_screen, '(a)') '    --- ' // trim(adjustl(message))
      ELSE
        WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called',             &
                                                   trim(adjustl(messagesources(sourcenumber))),   &
                                                   trim(adjustl(message))
      END IF
!#ifdef FLUSH_MESSAGES
      ! In Fortran >=2003 the call is:
      ! FLUSH(LUN_LOG)
      CALL flush(lun_screen)
!#endif
      END IF

  END SUBROUTINE screenmessage_1

  SUBROUTINE screenmessage_2(level, message)

    IMPLICIT NONE
```

```

    ! Global variables
    INTEGER, INTENT(IN)           :: level
    CHARACTER(LEN=*), INTENT(IN)  :: message

    IF (nscreen > 0) THEN
      IF (loginitcalled) THEN
        WRITE(lun_screen, '(a, " :: ", a, ": ", a)') trim(adjustl(loglevelnames(level + 2))),       &
                                                     trim(adjustl(messagesources(sourcenumber))),   &
                                                     trim(adjustl(message))
      ELSE
        WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called',                      &
                                                     trim(adjustl(messagesources(sourcenumber))),   &
                                                     trim(adjustl(message))
      END IF
!#ifdef FLUSH_MESSAGES
      ! In Fortran >=2003 the call is:
      ! FLUSH(LUN_LOG)
     CALL flush(lun_screen)
!#endif
      END IF

  END SUBROUTINE screenmessage_2

!================================================================================

  !--------------------------------------------------------------------
  !     S U B R O U T I N E    L O G   M E S S A G E
  !--------------------------------------------------------------------
  !--------------------------------------------------------------------
  SUBROUTINE logmessage_1(message)

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), INTENT(IN)  :: message

    IF (logfileopened) THEN
      IF (loginitcalled) THEN
        WRITE(lun_log, '(a)') '    --- ' // trim(adjustl(message))
      ELSE
        WRITE(lun_log, '(a, " :: ", a, ": ", a)') 'InitLogging not called',                      &
                                                  trim(adjustl(messagesources(sourcenumber))),   &
                                                  trim(adjustl(message))
      END IF
!#ifdef FLUSH_MESSAGES
      ! In Fortran >=2003 the call is:
      ! FLUSH(LUN_LOG)
     CALL flush(lun_log)
!#endif
    END IF

  END SUBROUTINE logmessage_1

  SUBROUTINE logmessage_2(level, message)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN)           :: level
    CHARACTER(LEN=*), INTENT(IN)  :: message

    IF (logfileopened) THEN
      IF (loginitcalled) THEN
        WRITE(lun_log, '(a, " :: ", a, ": ", a)') trim(adjustl(loglevelnames(level + 2))),       &
                                                  trim(adjustl(messagesources(sourcenumber))),   &
                                                  trim(adjustl(message))
      ELSE
        WRITE(lun_log, '(a, " :: ", a, ": ", a)') 'InitLogging not called',                      &
                                                  trim(adjustl(messagesources(sourcenumber))),   &
                                                  trim(adjustl(message))
      END IF
!#ifdef FLUSH_MESSAGES
      ! In Fortran >=2003 the call is:
      ! FLUSH(LUN_LOG)
     CALL flush(lun_log)
!#endif
    END IF

  END SUBROUTINE logmessage_2

```

```
!================================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E   A L L   M E S S A G E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE allmessage_1(message)

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), INTENT(IN) :: message

    CALL screenmessage(message)
    CALL logmessage(message)

  END SUBROUTINE allmessage_1

  SUBROUTINE allmessage_2(level, message)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN)          :: level
    CHARACTER(LEN=*), INTENT(IN) :: message


    CALL screenmessage(level, message)
    CALL logmessage(level, message)

  END SUBROUTINE allmessage_2

!================================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E   S E T   M E S S A G E   S O U R C E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE setmessagesource(source)

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), INTENT(IN) :: source

    sourcenumber = sourcenumber + 1
    messagesources(sourcenumber) = source

  END SUBROUTINE setmessagesource

!================================================================================

  !-------------------------------------------------------------------
  !     S U B R O U T I N E   U N S E T   M E S S A G E   S O U R C E
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE unsetmessagesource()

    IMPLICIT NONE

    sourcenumber = sourcenumber - 1

  END SUBROUTINE unsetmessagesource

!================================================================================

  !----------------------------------------------------------------------
  !     S U B R O U T I N E   P R O G R A M   V E R S I O N
  !----------------------------------------------------------------------
  !----------------------------------------------------------------------
  SUBROUTINE programversion()

    USE version

    IMPLICIT NONE

    WRITE(lun_screen, '(a)') trim(prog_fullname) // ' ' // trim(prog_version) // ' ' // trim(prog_date)
!     WRITE(LUN_SCREEN, '(a)') 'NOAA/NOS/CSDL, Coastal Marine Modeling Branch.'
    WRITE(lun_screen, '(a)') '  Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/).'
    WRITE(lun_screen, '(a)') '  NOAA/NOS/CSDL (https://nauticalcharts.noaa.gov/).'
    WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - This is free software; see the source for copying
```

```
      conditions.'
    WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - There is NO warranty.'

    WRITE(lun_screen, '(a)') ''

  END SUBROUTINE programversion

!================================================================================

  !----------------------------------------------------------------------
  !     S U B R O U T I N E   P R O G R A M   H E L P
  !----------------------------------------------------------------------
  !----------------------------------------------------------------------
  SUBROUTINE programhelp()

    IMPLICIT NONE

    CALL programversion

    WRITE(lun_screen, '(a)') 'Help Screen not yet implemented'

    WRITE(lun_screen, '(a)') ''

  END SUBROUTINE programhelp

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   T E R M I N A T E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE terminate()

    USE version

    IMPLICIT NONE

    CALL setmessagesource("Terminate")

    CALL allmessage(error, trim(adjustl(prog_name)) // " Terminating.")

    CALL unsetmessagesource()

    stop

  END SUBROUTINE terminate

!================================================================================

END MODULE pahm_messages
```

9.20 netcdfio-nems.F90 File Reference

Data Types

- type pahm_netcdfio::filedata_t
- type pahm_netcdfio::timedata_t

Modules

- module pahm_netcdfio

Macros

- #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

Functions/Subroutines

- subroutine pahm_netcdfio::initadcircnetcdfoutfile (adcircOutFile)

  Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.
- subroutine pahm_netcdfio::newadcircnetcdfoutfile (ncID, adcircOutFile)

  Creates a new NetCDF data file and puts it in define mode.
- subroutine pahm_netcdfio::base_netcdfcheckerr (ierr, file, line)

  Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file.
- subroutine pahm_netcdfio::netcdfterminate ()

  Terminates the program on NetCDF error.
- subroutine pahm_netcdfio::writenetcdfrecord (adcircOutFile, timeLoc)

  Writes data to the NetCDF file.
- subroutine pahm_netcdfio::setrecordcounterandstoretime (ncID, f, t)

  Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

Variables

- type(timedata_t), save pahm_netcdfio::mytime

9.20.1 Macro Definition Documentation

9.20.1.1 NetCDFCheckErr `#define NetCDFCheckErr( arg ) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)`

9.21 netcdfio-nems.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                M O D U L E   N E T C D F  I O
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_netcdfio

  USE pahm_sizes
  USE pahm_messages
  USE pahm_global
  USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
  USE netcdf

#ifdef __INTEL_COMPILER
  USE ifport
#endif

  IMPLICIT NONE

#define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

  INTEGER, PRIVATE            :: ncFormat
```

```
INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)

INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID

TYPE :: filedata_t
  LOGICAL                 :: initialized = .false.
  INTEGER                 :: filereccounter = 0
  CHARACTER(LEN=FNAMELEN) :: filename
  LOGICAL                 :: filefound = .false.  ! .true. if the netCDF file is present
END TYPE filedata_t

TYPE :: timedata_t
  LOGICAL :: initialized = .false.
  INTEGER :: timelen = 1  ! number of time slices to write
  INTEGER :: timedimid
  INTEGER :: timeid
  INTEGER :: timedims(1)
  REAL(sz), ALLOCATABLE :: time(:)
END TYPE timedata_t

TYPE, PRIVATE :: adcirccoorddata_t
  LOGICAL               :: initialized = .false.
  REAL(sz)              :: initval
  INTEGER               :: dimid
  INTEGER               :: varid
  INTEGER               :: vardimids
  INTEGER               :: vardims
  CHARACTER(50)         :: varname
  REAL(sz), ALLOCATABLE :: var(:)
  INTEGER               :: start(1), count(1)
END TYPE adcirccoorddata_t

TYPE, PRIVATE :: adcircvardata_t
   LOGICAL               :: initialized = .false.
   REAL(sz)              :: initval
   INTEGER               :: varid
   INTEGER               :: vardimids(2)
   INTEGER               :: vardims(2)
   CHARACTER(50)         :: varname
   REAL(sz), ALLOCATABLE :: var(:, :)
   INTEGER               :: start(2), count(2)
END TYPE adcircvardata_t

TYPE, PRIVATE :: adcircvardata3d_t
   LOGICAL               :: initialized = .false.
   REAL(sz)              :: initval
   INTEGER               :: varid
   INTEGER               :: vardimids(3)
   INTEGER               :: vardims(3)
   CHARACTER(50)         :: varname
   REAL(sz), ALLOCATABLE :: var(:, :, :)
   INTEGER               :: start(3), count(3)
END TYPE adcircvardata3d_t

TYPE(filedata_t), SAVE   :: myfile
TYPE(timedata_t), SAVE   :: mytime

TYPE(adcirccoorddata_t), PRIVATE, SAVE :: crdtime
TYPE(adcirccoorddata_t), PRIVATE, SAVE :: crdlons
TYPE(adcirccoorddata_t), PRIVATE, SAVE :: crdlats
TYPE(adcirccoorddata_t), PRIVATE, SAVE :: crdxcs
TYPE(adcirccoorddata_t), PRIVATE, SAVE :: crdycs

TYPE(adcircvardata_t), PRIVATE, SAVE   :: datelements
TYPE(adcircvardata_t), PRIVATE, SAVE   :: datatmpres
TYPE(adcircvardata_t), PRIVATE, SAVE   :: datwindx
TYPE(adcircvardata_t), PRIVATE, SAVE   :: datwindy


CONTAINS


!----------------------------------------------------------------
!  S U B R O U T I N E   I N I T  A D C I R C  N E T C D F  O U T  F I L E
!----------------------------------------------------------------
!  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
!----------------------------------------------------------------
SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)

```

```fortran
USE version
USE timedateutils, ONLY : gettimeconvsec, datetime2string

IMPLICIT NONE

CHARACTER(LEN=*), INTENT(INOUT) :: adcircOutFile

INTEGER             :: ncID
CHARACTER(LEN=64)   :: refDateTimeStr, modDateTimeStr, tmpVarName
CHARACTER(LEN=128)  :: institution, source, history, comments, host, &
                       conventions, contact, references
INTEGER             :: tvals(8)
INTEGER             :: ierr ! success or failure of a netcdf call


CALL setmessagesource("InitAdcircNetCDFOutFile")

refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units = unittime)

institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
source      = ''
history     = ''
comments    = ''
host        = ''
conventions = 'UGRID-0.9.0'
contact     = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
references  = ''


! Create the NetCDF output file.
CALL newadcircnetcdfoutfile(ncid, adcircoutfile)

!====================
!===== (1) Define all the dimensions
!====================
tmpvarname = 'time'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'longitude'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'latitude'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'node'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'element'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemdimid)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'noel'
  ierr = nf90_def_dim(ncid, trim(tmpvarname), 3,  vertdimid)
    CALL netcdfcheckerr(ierr)

tmpvarname = 'mesh'
ierr = nf90_def_dim(ncid, trim(tmpvarname), 1,    meshdimid)
  CALL netcdfcheckerr(ierr)

!====================
!===== (2) Define all the variables
!====================
!----- Time variable
tmpvarname = 'time'
  crdtime%varname    = trim(tmpvarname)
  crdtime%varDimIDs = crdtime%dimID
  crdtime%varDims    = SIZE(times, 1)
  crdtime%start(1)   = 1
  crdtime%count(1)   = crdtime%varDims

  ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdtime%varID, 'long_name',     'model ' // trim(tmpvarname))
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdtime%varID, 'units',         trim(refdatetimestr))
```

```
      CALL netcdfcheckerr(ierr)

    ALLOCATE(crdtime%var(crdtime%varDims))
    crdtime%var = times * gettimeconvsec(unittime, 1)

  !----- Longitude variable
  tmpvarname = 'longitude'
    crdlons%varname = trim(tmpvarname)
    crdlons%varDimIDs = nodedimid
    ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlons%varDims)
      CALL netcdfcheckerr(ierr)
    crdlons%start(1) = 1
    crdlons%count(1) = crdlons%varDims

    ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlons%varID, 'long_name',     trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlons%varID, 'units',         'degrees_east')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue',    rmissv)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlons%varID, 'positive',      'east')
      CALL netcdfcheckerr(ierr)

    ALLOCATE(crdlons%var(crdlons%varDims))
    crdlons%var = slam

  !----- Latitude variable
  tmpvarname = 'latitude'
    crdlats%varname = trim(tmpvarname)
    crdlats%varDimIDs = nodedimid
    ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlats%varDims)
      CALL netcdfcheckerr(ierr)
    crdlats%start(1) = 1
    crdlats%count(1) = crdlats%varDims

    ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlats%varID, 'long_name',     trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlats%varID, 'units',         'degrees_north')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',    rmissv)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, crdlats%varID, 'positive',      'north')
      CALL netcdfcheckerr(ierr)

    ALLOCATE(crdlats%var(crdlats%varDims))
    crdlats%var = sfea

  !----- Element variable
  tmpvarname = 'tri'
    datelements%varname = trim(tmpvarname)
    datelements%varDimIDs(1) = vertdimid
    datelements%varDimIDs(2) = elemdimid
    ierr = nf90_inquire_dimension(ncid, vertdimid, len = datelements%varDims(1))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_inquire_dimension(ncid, elemdimid, len = datelements%varDims(2))
      CALL netcdfcheckerr(ierr)
    datelements%start(1) = 1
    datelements%count(1) = datelements%varDims(1)
    datelements%start(2) = 1
    datelements%count(2) = datelements%varDims(2)

    ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, datelements%varID,'long_name',     trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, datelements%varID,'standard_name', trim(tmpvarname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, datelements%varID, 'cf_role',      'face_node_connectivity')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, datelements%varID, 'start_index',  1)
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, datelements%varID, 'units',        'nondimensional')
      CALL netcdfcheckerr(ierr)
```

```
      ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',    imissv)
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
      datelements%var = nm

    !----- Mesh variable
    tmpvarname = 'adcirc_mesh'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, meshvarid,'long_name',               'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid,'standard_name',           'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'cf_role',                'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates',       'lon lat')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
        CALL netcdfcheckerr(ierr)

    !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
    tmpvarname = 'projection'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, projvarid,'long_name',         'equidistant cylindrical projection')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid,'standard_name',     'CPP')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'node_coordinates', 'x y')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lon0',             slam0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lat0',             sfea0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'earth_radius',     rearth)
        CALL netcdfcheckerr(ierr)

    !----- CPP CPP x-coordinates
    tmpvarname = 'x'
      crdxcs%varname = trim(tmpvarname)
      crdxcs%dimID = nodedimid
      crdxcs%varDimIDs = nodedimid
      ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
        CALL netcdfcheckerr(ierr)
      crdxcs%start(1) = 1
      crdxcs%count(1) = crdxcs%varDims

      ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',     'CPP x coordinate')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdxcs%varID, 'units',         'm')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)

      ALLOCATE(crdxcs%var(crdxcs%varDims))
      crdxcs%var = xcslam

    !----- CPP y-coordinates
    tmpvarname = 'y'
      crdycs%varname = trim(tmpvarname)
      crdycs%dimID = nodedimid
      crdycs%varDimIDs = nodedimid
      ierr = nf90_inquire_dimension(ncid, crdycs%dimID, len = crdycs%varDims)
        CALL netcdfcheckerr(ierr)
      crdycs%start(1) = 1
      crdycs%count(1) = crdycs%varDims

      ierr = nf90_def_var(ncid, trim(crdycs%varname), nf90_double, crdycs%varDimIDs, crdycs%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdycs%varID, 'long_name',     'CPP y coordinate')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdycs%varID, 'standard_name', 'cpp_y')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdycs%varID, 'units',         'm')
```

```
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdycs%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)

        ALLOCATE(crdycs%var(crdycs%varDims))
        crdycs%var = ycsfea

      !----- Atmospheric Pressure variable
      tmpvarname = trim(ncvarnam_pres)
        datatmpres%varname      = trim(tmpvarname)
        datatmpres%varDimIDs(1) = nodedimid
        datatmpres%varDimIDs(2) = crdtime%dimID
        datatmpres%varDims(1)   = SIZE(wpress, 1)
        datatmpres%varDims(2)   = SIZE(wpress, 2)
        datatmpres%start(1)     = 1
        datatmpres%count(1)     = datatmpres%varDims(1)
        datatmpres%start(2)     = 1
        datatmpres%count(2)     = datatmpres%varDims(2)

        ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
                            datatmpres%varDimIDs, datatmpres%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',     'air pressure at sea level')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'units',         'Pa')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',   'time lat lon')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'location',      'node')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',          'adcirc_mesh')
          CALL netcdfcheckerr(ierr)

        ALLOCATE(datatmpres%var(datatmpres%varDims(1), datatmpres%varDims(2)))
        datatmpres%var = wpress

      !----- Wind velocity variables
      ! Eastward
      tmpvarname = trim(ncvarnam_wndx)
        datwindx%varname      = trim(tmpvarname)
        datwindx%varDimIDs(1) = nodedimid
        datwindx%varDimIDs(2) = crdtime%dimID
        datwindx%varDims(1)   = SIZE(wvelx, 1)
        datwindx%varDims(2)   = SIZE(wvelx, 2)
        datwindx%start(1)     = 1
        datwindx%count(1)     = datwindx%varDims(1)
        datwindx%start(2)     = 1
        datwindx%count(2)     = datwindx%varDims(2)

        ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
                            datwindx%varDimIDs, datwindx%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',     '10-m eastward wind component')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'units',         'm s-1')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates',   'time lat lon')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'location',      'node')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',          'adcirc_mesh')
          CALL netcdfcheckerr(ierr)

        ALLOCATE(datwindx%var(datwindx%varDims(1), datwindx%varDims(2)))
        datwindx%var = wvelx

      ! Northward
      tmpvarname = trim(ncvarnam_wndy)
        datwindy%varname      = trim(tmpvarname)
        datwindy%varDimIDs(1) = nodedimid
        datwindy%varDimIDs(2) = crdtime%dimID
        datwindy%varDims(1)   = SIZE(wvely, 1)
        datwindy%varDims(2)   = SIZE(wvely, 2)
```

```
      datwindy%start(1)      = 1
      datwindy%count(1)      = datwindy%varDims(1)
      datwindy%start(2)      = 1
      datwindy%count(2)      = datwindy%varDims(2)

      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
                          datwindy%varDimIDs, datwindy%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',     '10-m northward wind component')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'units',         'm s-1')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates',   'time lat lon')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'location',      'node')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',          'adcirc_mesh')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datwindy%var(datwindy%varDims(1), datwindy%varDims(2)))
      datwindy%var = wvely

    !====================
    !===== (3) Set Deflate parameters if requested by the user
    !====================
#ifdef NETCDF_CAN_DEFLATE
    IF (ncformat == nc4form) THEN
      ierr = nf90_def_var_deflate(ncid, crdlons%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdlats%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdxcs%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdycs%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datatmpres%varID,  ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
    END IF
#endif

    !====================
    !===== (4) Global metadata definitions and variables
    !====================
    ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
 ')')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
      CALL netcdfcheckerr(ierr)

```

```
    CALL date_and_time(values = tvals)
    WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
    moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
 moddatetimestr)

    ierr = nf90_put_att(ncid, nf90_global,'creation_date', trim(moddatetimestr))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global,'modification_date', trim(moddatetimestr))
      CALL netcdfcheckerr(ierr)

    !----- Finalize the definitions in the NetCDF file
    ierr = nf90_enddef(ncid)
      CALL netcdfcheckerr(ierr)

    !====================
    !===== (5) Put the static data into the NetCDF file and then close it
    !====================
    ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdycs%varID, crdycs%var, crdycs%start, crdycs%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datatmpres%varID, datatmpres%var, datatmpres%start, datatmpres%count)
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datwindx%varID, datwindx%var, datwindx%start, datwindx%count)
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datwindy%varID, datwindy%var, datwindy%start, datwindy%count)
      CALL netcdfcheckerr(ierr)


    !---------- (16) Set all the "initialized" flags to .TRUE.
    crdlons%initialized      = .true.
    crdlats%initialized      = .true.
    crdxcs%initialized       = .true.
    crdycs%initialized       = .true.
    datelements%initialized  = .true.
    datatmpres%initialized   = .true.
    datwindx%initialized     = .true.
    datwindy%initialized     = .true.

    myfile%fileName    = adcircoutfile
    myfile%initialized = .true.

    !----- Close the NetCDF file
    ierr = nf90_close(ncid)
      CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE initadcircnetcdfoutfile

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   N E W A D C I R C   N E T C D F   O U T   F I L E
  !----------------------------------------------------------------
  !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
  !----------------------------------------------------------------
  SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)

    IMPLICIT NONE

    INTEGER, INTENT(OUT)              :: ncID
```

```
    CHARACTER(LEN=*), INTENT(INOUT)    :: adcircOutFile

    LOGICAL                            :: fileFound = .false.
    CHARACTER(LEN=FNAMELEN)            :: outFile, sys_cmd
    CHARACTER(LEN=14)                  :: fext, date_time
    INTEGER                            :: pos, ierr, tvals(8)


    CALL setmessagesource("NewAdcircNetCDFOutFile")

    !----------
    ! Set some variables that depend upon the type of NetCDF supported.
#if defined(HAVE_NETCDF4)
    fext = ".nc4"
    ncformat = nc4form
#else
    fext = ".nc"
    ncformat = nc3form
#endif

    !----------
    ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
    ! extension in the filename; re-define the adcircOutFile variable.
    pos = scan(trim(adcircoutfile), ".", back= .true.)
    IF (pos > 0) THEN
      adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
    ELSE
      adcircoutfile = trim(adcircoutfile) // trim(fext)
    END IF

    !----------
    ! If the adcircOutFile exists then rename it to:
    !   adcircOutFile-YYYYMMDDhhmmss.
    ! The user can remove these files afterwards.
    INQUIRE(file=adcircoutfile, exist=filefound)
    IF (filefound) THEN
      CALL date_and_time(values = tvals)
      WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
      outfile = trim(adcircoutfile) // "-" // trim(date_time)
      sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
      ierr = system(trim(sys_cmd))
      IF (ierr == 0) THEN
        WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
        CALL logmessage(info, scratchmessage)
        filefound = .false.
      ELSE
        WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' //
 trim(outfile)
        CALL logmessage(error, scratchmessage)
      END IF
    END IF

    IF (filefound) THEN
      WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adcircoutfile) // ' exists. Remove the
 file to proceed.'
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      CALL netcdfterminate
    END IF

    WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adcircoutfile) // ' and putting it in define
 mode.'
    CALL logmessage(info, scratchmessage)

    ! Create the NetCDF file
    ierr = nf90_create(adcircoutfile, ncformat, ncid)
    CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE newadcircnetcdfoutfile

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   N E T C D F  C H E C K  E R R
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
```

```
  SUBROUTINE base_netcdfcheckerr(ierr, file, line)

    IMPLICIT NONE

    INTEGER, INTENT(IN)           :: ierr
    CHARACTER(LEN=*), INTENT(IN)  :: file
    INTEGER, INTENT(IN)           :: line

    CHARACTER(LEN=1024)           :: tmpSTR

    CALL setmessagesource("NetCDFCheckErr")

    IF (ierr /= nf90_noerr) THEN
      CALL allmessage(error, nf90_strerror(ierr))
      WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
      CALL allmessage(info, tmpstr)
      CALL netcdfterminate()
    END IF

    CALL unsetmessagesource()

  END SUBROUTINE base_netcdfcheckerr

!================================================================================

  !-----------------------------------------------------------------
  ! S U B R O U T I N E   N E T C D F   T E R M I N A T E
  !-----------------------------------------------------------------
  !
  !-----------------------------------------------------------------
  SUBROUTINE netcdfterminate()

    USE version

    IMPLICIT NONE

    CALL setmessagesource("NetCDFTerminate")

    CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")

    CALL exit(1)

    CALL unsetmessagesource()

  END SUBROUTINE netcdfterminate

!================================================================================


  !-----------------------------------------------------------------
  !  S U B R O U T I N E  W R I T E  N E T C D F  R E C O R D
  !-----------------------------------------------------------------
  !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
  !-----------------------------------------------------------------
  SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)

    IMPLICIT NONE

    CHARACTER(LEN=*), INTENT(IN) :: adcircOutFile

    INTEGER :: timeLoc
    INTEGER :: ncID, ierr, nodes
    INTEGER :: start(2), kount(2)

    CALL setmessagesource("WriteNetCDFRecord")

    ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
    CALL netcdfcheckerr(ierr)

    ! Set up the 2D netcdf data extents
    ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
    start(1) = 1
    start(2) = myfile%fileRecCounter
    kount(1) = nodes
    kount(2) = mytime%timeLen

    ierr = nf90_put_var(ncid, datatmpres%varID, wpress(:, timeloc), start, kount)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datwindx%varID, wvelx(:, timeloc), start, kount)
      CALL netcdfcheckerr(ierr)
```

```
    ierr = nf90_put_var(ncid, datwindy%varID, wvely(:, timeloc), start, kount)
     CALL netcdfcheckerr(ierr)

    ! Close netCDF file
    ierr = nf90_close(ncid)
      CALL netcdfcheckerr(ierr)

    DEALLOCATE(mytime%time)

    CALL unsetmessagesource()

  END SUBROUTINE writenetcdfrecord

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E  S E T   R E C O R D  C O U N T E R  A N D  S T O R E   T I M E
  !----------------------------------------------------------------
  !  author
  !----------------------------------------------------------------
  SUBROUTINE setrecordcounterandstoretime(ncID, f, t)

    IMPLICIT NONE

    INTEGER, INTENT(IN)             :: ncID
    TYPE(filedata_t), INTENT(INOUT) :: f
    TYPE(timedata_t), INTENT(INOUT) :: t

    REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
    LOGICAL               :: timeFound      ! true if current time is in array of stored times

    INTEGER :: ndim      ! number of dimensions in the netcdf file
    INTEGER :: nvar      ! number of variables in the netcdf file
    INTEGER :: natt      ! number of attributes in the netcdf file

    INTEGER :: counti(1), starti(1)
    INTEGER :: ierr  ! success or failure of netcdf call
    INTEGER :: i     ! loop counter


    CALL setmessagesource("SetRecordCounterAndStoreTime")

    ! Inquire the time variable
    ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_inq_varid(ncid, 'time', t%timeID)
      CALL netcdfcheckerr(ierr)

    ! Determine the relationship between the current simulation time
    ! and the time array stored in the netcdf file. Set the record
    ! counter based on this relationship.
    IF (f%fileRecCounter /= 0) THEN
      ALLOCATE(storedtimes(f%fileRecCounter))
      ierr = nf90_get_var(ncid, t%timeID, storedtimes)
        CALL netcdfcheckerr(ierr)
      timefound = .false.

      DO i = 1, f%fileRecCounter
        IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
          timefound = .true.
          EXIT
        ENDIF
      END DO

      IF (timefound .EQV. .false.) THEN
        ! Increment the record counter so that we can store data at the
        ! next location in the netcdf file (i.e., all of the times
        ! in the netcdf file were found to be earlier than the current
        ! adcirc simulation time).
        f%fileRecCounter = f%fileRecCounter + 1
      ELSE
        ! set the counter at the index that reflects the
        ! current time within the netcdf file (or is between two times
        ! found in the netcdf file).
        ! WARNING: all subsequent data will remain in the file, we
        ! are just overwriting it ... if we don't overwrite all of it,
```

```
          ! the pre-existing data will still be there, which is probably
          ! not what the user intended ... but apparently there is no
          ! way to delete data from netcdf files:
          ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
          scratchformat = '("Overwriting pre-existing data in netcdf file ",a,' //   &
                          '" for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.")'
          WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
          CALL allmessage(info, scratchmessage)
          f%fileRecCounter = i
        ENDIF

        DEALLOCATE(storedtimes)
      ELSE
        ! set the counter at 1 so we can record our first time value
        f%fileRecCounter = 1
      ENDIF

      ! Store simulation time in netcdf file
      starti(1) = f%fileRecCounter
      counti(1) = t%timeLen
      ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
        CALL netcdfcheckerr(ierr)

      CALL unsetmessagesource()

    END SUBROUTINE setrecordcounterandstoretime

!================================================================================

END MODULE pahm_netcdfio
```

9.22 netcdfio-orig.F90 File Reference

Data Types

- type pahm_netcdfio::filedata_t
- type pahm_netcdfio::timedata_t

Modules

- module pahm_netcdfio

Macros

- #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

Functions/Subroutines

- subroutine pahm_netcdfio::initadcircnetcdfoutfile (adcircOutFile)

  Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.
- subroutine pahm_netcdfio::newadcircnetcdfoutfile (ncID, adcircOutFile)

  Creates a new NetCDF data file and puts it in define mode.
- subroutine pahm_netcdfio::base_netcdfcheckerr (ierr, file, line)

  Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file.
- subroutine pahm_netcdfio::netcdfterminate ()

  Terminates the program on NetCDF error.
- subroutine pahm_netcdfio::writenetcdfrecord (adcircOutFile, timeLoc)

  Writes data to the NetCDF file.
- subroutine pahm_netcdfio::setrecordcounterandstoretime (ncID, f, t)

  Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

9.22.1 Macro Definition Documentation

9.22.1.1 NetCDFCheckErr

```
#define NetCDFCheckErr(
       arg ) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
```

9.23 netcdfio-orig.F90

Go to the documentation of this file.

```
!-----------------------------------------------------------------
!                M O D U L E   N E T C D F  I O
!-----------------------------------------------------------------
!-----------------------------------------------------------------

MODULE pahm_netcdfio

  USE pahm_sizes
  USE pahm_messages
  USE pahm_global
  USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
  USE netcdf

#ifdef __INTEL_COMPILER
  USE ifport
#endif

  IMPLICIT NONE

#define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

  INTEGER, PRIVATE            :: ncFormat
  INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
  INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)

  INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
  INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID

  TYPE :: filedata_t
    LOGICAL                  :: initialized = .false.
    INTEGER                  :: fileRecCounter = 0
    CHARACTER(LEN=FNAMELEN)  :: fileName
    LOGICAL                  :: fileFound = .false.  ! .true. if the netCDF file is present
  END TYPE filedata_t

  TYPE :: timedata_t
    LOGICAL :: initialized = .false.
    INTEGER :: timeLen = 1  ! number of time slices to write
    INTEGER :: timeDimID
    INTEGER :: timeID
    INTEGER :: timeDims(1)
    REAL(SZ), ALLOCATABLE :: time(:)
  END TYPE timedata_t

  TYPE, PRIVATE :: adcirccoorddata_t
    LOGICAL                :: initialized = .false.
    REAL(SZ)               :: initVal
    INTEGER                :: dimID
    INTEGER                :: varID
    INTEGER                :: varDimIDs
    INTEGER                :: varDims
    CHARACTER(50)          :: varname
    REAL(SZ), ALLOCATABLE  :: var(:)
    INTEGER                :: start(1), count(1)
  END TYPE adcirccoorddata_t

  TYPE, PRIVATE :: adcircvardata_t
     LOGICAL                :: initialized = .false.
     REAL(SZ)               :: initVal
     INTEGER                :: varID
     INTEGER                :: varDimIDs(2)
```

```
   INTEGER               :: varDims(2)
   CHARACTER(50)         :: varname
   REAL(SZ), ALLOCATABLE :: var(:, :)
   INTEGER               :: start(2), count(2)
END TYPE adcircvardata_t

TYPE, PRIVATE :: adcircvardata3d_t
   LOGICAL               :: initialized = .false.
   REAL(SZ)              :: initVal
   INTEGER               :: varID
   INTEGER               :: varDimIDs(3)
   INTEGER               :: varDims(3)
   CHARACTER(50)         :: varname
   REAL(SZ), ALLOCATABLE :: var(:, :, :)
   INTEGER               :: start(3), count(3)
END TYPE adcircvardata3d_t

TYPE(FileData_T), SAVE   :: myFile
TYPE(TimeData_T), SAVE   :: myTime

TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdTime
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLons
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLats
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdXCs
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdYCs

TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datElements
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datAtmPres
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datWindX
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datWindY


CONTAINS


!----------------------------------------------------------------
!  S U B R O U T I N E   I N I T  A D C I R C  N E T C D F  O U T  F I L E
!----------------------------------------------------------------
!  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
!----------------------------------------------------------------
SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)

  USE version
  USE timedateutils, ONLY : gettimeconvsec, datetime2string

  IMPLICIT NONE

  CHARACTER(LEN=*), INTENT(INOUT) :: adcircOutFile

  INTEGER             :: ncID
  CHARACTER(LEN=64)   :: refDateTimeStr, modDateTimeStr, tmpVarName
  CHARACTER(LEN=128)  :: institution, source, history, comments, host, &
                         conventions, contact, references
  INTEGER             :: tvals(8)
  INTEGER             :: ierr ! success or failure of a netcdf call


  CALL setmessagesource("InitAdcircNetCDFOutFile")

  refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units = unittime)

  institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
  source      = ''
  history     = ''
  comments    = ''
  host        = ''
  conventions = 'UGRID-0.9.0'
  contact     = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
  references  = ''


  ! Create the NetCDF output file.
  CALL newadcircnetcdfoutfile(ncid, adcircoutfile)

  !====================
  !===== (1) Define all the dimensions
  !====================
  tmpvarname = 'time'
    ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
      CALL netcdfcheckerr(ierr)

```

```
    tmpvarname = 'longitude'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'latitude'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'node'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'nele'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemdimid)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'nvertex'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), 3,  vertdimid)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'mesh'
    ierr = nf90_def_dim(ncid, trim(tmpvarname), 1,    meshdimid)
      CALL netcdfcheckerr(ierr)

    !====================
    !===== (2) Define all the variables
    !====================
    !----- Time variable
    tmpvarname = 'time'
      crdtime%varname   = trim(tmpvarname)
      crdtime%varDimIDs = crdtime%dimID
      crdtime%varDims   = SIZE(times, 1)
      crdtime%start(1)  = 1
      crdtime%count(1)  = crdtime%varDims

      ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdtime%varID, 'long_name',     'model ' // trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdtime%varID, 'units',         trim(refdatetimestr))
        CALL netcdfcheckerr(ierr)

      ALLOCATE(crdtime%var(crdtime%varDims))
      crdtime%var = times * gettimeconvsec(unittime, 1)

    !----- Longitude variable
    tmpvarname = 'longitude'
      crdlons%varname = trim(tmpvarname)
      crdlons%varDimIDs = crdlons%dimID
      ierr = nf90_inquire_dimension(ncid, crdlons%dimID, len = crdlons%varDims)
        CALL netcdfcheckerr(ierr)
      crdlons%start(1) = 1
      crdlons%count(1) = crdlons%varDims

      ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlons%varID, 'long_name',     trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlons%varID, 'units',         'degrees_east')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlons%varID, 'positive',      'east')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(crdlons%var(crdlons%varDims))
      crdlons%var = slam

    !----- Latitude variable
    tmpvarname = 'latitude'
      crdlats%varname = trim(tmpvarname)
      crdlats%varDimIDs = crdlats%dimID
      ierr = nf90_inquire_dimension(ncid, crdlats%dimID, len = crdlats%varDims)
        CALL netcdfcheckerr(ierr)
      crdlats%start(1) = 1
      crdlats%count(1) = crdlats%varDims

```

```
      ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, 'long_name',     trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, 'units',         'degrees_north')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, 'positive',      'north')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(crdlats%var(crdlats%varDims))
      crdlats%var = sfea

    !----- Element variable
    tmpvarname = 'element'
      datelements%varname = trim(tmpvarname)
      datelements%varDimIDs(1) = vertdimid
      datelements%varDimIDs(2) = elemdimid
      ierr = nf90_inquire_dimension(ncid, vertdimid, len = datelements%varDims(1))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_inquire_dimension(ncid, elemdimid, len = datelements%varDims(2))
        CALL netcdfcheckerr(ierr)
      datelements%start(1) = 1
      datelements%count(1) = datelements%varDims(1)
      datelements%start(2) = 1
      datelements%count(2) = datelements%varDims(2)

      ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID,'long_name',     trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID,'standard_name', trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'cf_role',      'face_node_connectivity')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'start_index',  1)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'units',        'nondimensional')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',    imissv)
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
      datelements%var = nm

    !----- Mesh variable
    tmpvarname = 'adcirc_mesh'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, meshvarid,'long_name',               'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid,'standard_name',           'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'cf_role',                'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates',       'lon lat')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
        CALL netcdfcheckerr(ierr)

    !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
    tmpvarname = 'projection'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, projvarid,'long_name',         'equidistant cylindrical projection')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid,'standard_name',     'CPP')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'node_coordinates', 'x y')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lon0',             slam0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lat0',             sfea0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'earth_radius',     rearth)
        CALL netcdfcheckerr(ierr)
```

```

!----- CPP CPP x-coordinates
tmpvarname = 'x'
  crdxcs%varname = trim(tmpvarname)
  crdxcs%dimID = nodedimid
  crdxcs%varDimIDs = nodedimid
  ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
    CALL netcdfcheckerr(ierr)
  crdxcs%start(1) = 1
  crdxcs%count(1) = crdxcs%varDims

  ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',     'CPP x coordinate')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdxcs%varID, 'units',         'm')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',    rmissv)
    CALL netcdfcheckerr(ierr)

  ALLOCATE(crdxcs%var(crdxcs%varDims))
  crdxcs%var = xcslam

!----- CPP y-coordinates
tmpvarname = 'y'
  crdycs%varname = trim(tmpvarname)
  crdycs%dimID = nodedimid
  crdycs%varDimIDs = nodedimid
  ierr = nf90_inquire_dimension(ncid, crdycs%dimID, len = crdycs%varDims)
    CALL netcdfcheckerr(ierr)
  crdycs%start(1) = 1
  crdycs%count(1) = crdycs%varDims

  ierr = nf90_def_var(ncid, trim(crdycs%varname), nf90_double, crdycs%varDimIDs, crdycs%varID)
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdycs%varID, 'long_name',     'CPP y coordinate')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdycs%varID, 'standard_name', 'cpp_y')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdycs%varID, 'units',         'm')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, crdycs%varID, '_FillValue',    rmissv)
    CALL netcdfcheckerr(ierr)

  ALLOCATE(crdycs%var(crdycs%varDims))
  crdycs%var = ycsfea

!----- Atmospheric Pressure variable
tmpvarname = trim(ncvarnam_pres)
  datatmpres%varname      = trim(tmpvarname)
  datatmpres%varDimIDs(1) = nodedimid
  datatmpres%varDimIDs(2) = crdtime%dimID
  datatmpres%varDims(1)   = SIZE(wpress, 1)
  datatmpres%varDims(2)   = SIZE(wpress, 2)
  datatmpres%start(1)     = 1
  datatmpres%count(1)     = datatmpres%varDims(1)
  datatmpres%start(2)     = 1
  datatmpres%count(2)     = datatmpres%varDims(2)

  ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
                      datatmpres%varDimIDs, datatmpres%varID)
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',     'air pressure at sea level')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'units',         'Pa')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',   'time lat lon')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'location',      'node')
    CALL netcdfcheckerr(ierr)
  ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',          'adcirc_mesh')
    CALL netcdfcheckerr(ierr)

  ALLOCATE(datatmpres%var(datatmpres%varDims(1), datatmpres%varDims(2)))
  datatmpres%var = wpress
```

```fortran

    !----- Wind velocity variables
    ! Eastward
    tmpvarname = trim(ncvarnam_wndx)
      datwindx%varname      = trim(tmpvarname)
      datwindx%varDimIDs(1) = nodedimid
      datwindx%varDimIDs(2) = crdtime%dimID
      datwindx%varDims(1)   = SIZE(wvelx, 1)
      datwindx%varDims(2)   = SIZE(wvelx, 2)
      datwindx%start(1)     = 1
      datwindx%count(1)     = datwindx%varDims(1)
      datwindx%start(2)     = 1
      datwindx%count(2)     = datwindx%varDims(2)

      ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
                          datwindx%varDimIDs, datwindx%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',     '10-m eastward wind component')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'units',         'm s-1')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates',   'time lat lon')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'location',      'node')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',          'adcirc_mesh')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datwindx%var(datwindx%varDims(1), datwindx%varDims(2)))
      datwindx%var = wvelx

    ! Northward
    tmpvarname = trim(ncvarnam_wndy)
      datwindy%varname      = trim(tmpvarname)
      datwindy%varDimIDs(1) = nodedimid
      datwindy%varDimIDs(2) = crdtime%dimID
      datwindy%varDims(1)   = SIZE(wvely, 1)
      datwindy%varDims(2)   = SIZE(wvely, 2)
      datwindy%start(1)     = 1
      datwindy%count(1)     = datwindy%varDims(1)
      datwindy%start(2)     = 1
      datwindy%count(2)     = datwindy%varDims(2)

      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
                          datwindy%varDimIDs, datwindy%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',     '10-m northward wind component')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'units',         'm s-1')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',    rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates',   'time lat lon')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'location',      'node')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',          'adcirc_mesh')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datwindy%var(datwindy%varDims(1), datwindy%varDims(2)))
      datwindy%var = wvely

    !====================
    !===== (3) Set Deflate parameters if requested by the user
    !====================
#ifdef NETCDF_CAN_DEFLATE
    IF (ncformat == nc4form) THEN
      ierr = nf90_def_var_deflate(ncid, crdlons%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdlats%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdxcs%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, crdycs%varID,     ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
```

```
      ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datatmpres%varID,  ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, ncdlevel)
        CALL netcdfcheckerr(ierr)
    END IF
#endif

    !====================
    !===== (4) Global metadata definitions and variables
    !====================
    ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
 ')')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
      CALL netcdfcheckerr(ierr)

    CALL date_and_time(values = tvals)
    WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
    moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
 moddatetimestr)

    ierr = nf90_put_att(ncid, nf90_global,'creation_date', trim(moddatetimestr))
      CALL netcdfcheckerr(ierr)
    ierr = nf90_put_att(ncid, nf90_global,'modification_date', trim(moddatetimestr))
      CALL netcdfcheckerr(ierr)

    !----- Finalize the definitions in the NetCDF file
    ierr = nf90_enddef(ncid)
      CALL netcdfcheckerr(ierr)

    !====================
    !===== (5) Put the static data into the NetCDF file and then close it
    !====================
    ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, crdycs%varID, crdycs%var, crdycs%start, crdycs%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datatmpres%varID, datatmpres%var, datatmpres%start, datatmpres%count)
```

```
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datwindx%varID, datwindx%var, datwindx%start, datwindx%count)
      CALL netcdfcheckerr(ierr)

     ierr = nf90_put_var(ncid, datwindy%varID, datwindy%var, datwindy%start, datwindy%count)
      CALL netcdfcheckerr(ierr)


    !---------- (16) Set all the "initialized" flags to .TRUE.
    crdlons%initialized      = .true.
    crdlats%initialized      = .true.
    crdxcs%initialized       = .true.
    crdycs%initialized       = .true.
    datelements%initialized  = .true.
    datatmpres%initialized   = .true.
    datwindx%initialized     = .true.
    datwindy%initialized     = .true.

    myfile%fileName    = adcircoutfile
    myfile%initialized = .true.

    !----- Close the NetCDF file
    ierr = nf90_close(ncid)
      CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE initadcircnetcdfoutfile

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   N E W  A D C I R C  N E T C D F  O U T  F I L E
  !----------------------------------------------------------------
  !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
  !----------------------------------------------------------------
  SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)

    IMPLICIT NONE

    INTEGER, INTENT(OUT)              :: ncID
    CHARACTER(LEN=*), INTENT(INOUT)   :: adcircOutFile

    LOGICAL                           :: fileFound = .false.
    CHARACTER(LEN=FNAMELEN)           :: outFile, sys_cmd
    CHARACTER(LEN=14)                 :: fext, date_time
    INTEGER                           :: pos, ierr, tvals(8)


    CALL setmessagesource("NewAdcircNetCDFOutFile")

    !----------
    ! Set some variables that depend upon the type of NetCDF supported.
#if defined(HAVE_NETCDF4)
    fext = ".nc4"
    ncformat = nc4form
#else
    fext = ".nc"
    ncformat = nc3form
#endif

    !----------
    ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
    ! extension in the filename; re-define the adcircOutFile variable.
    pos = scan(trim(adcircoutfile), ".", back= .true.)
    IF (pos > 0) THEN
      adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
    ELSE
      adcircoutfile = trim(adcircoutfile) // trim(fext)
    END IF

    !----------
    ! If the adcircOutFile exists then rename it to:
    !   adcircOutFile-YYYYMMDDhhmmss.
    ! The user can remove these files afterwards.
    INQUIRE(file=adcircoutfile, exist=filefound)
    IF (filefound) THEN
      CALL date_and_time(values = tvals)
      WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
      outfile = trim(adcircoutfile) // "-" // trim(date_time)
```

```
   sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
   ierr = system(trim(sys_cmd))
   IF (ierr == 0) THEN
    WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
    CALL logmessage(info, scratchmessage)
    filefound = .false.
   ELSE
    WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' // trim(outfile)
    CALL logmessage(error, scratchmessage)
   END IF
  END IF

  IF (filefound) THEN
   WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adcircoutfile) // ' exists. Remove the file to proceed.'
   CALL allmessage(error, scratchmessage)

   CALL unsetmessagesource()

   CALL netcdfterminate
  END IF

  WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adcircoutfile) // ' and putting it in define mode.'
  CALL logmessage(info, scratchmessage)

  ! Create the NetCDF file
  ierr = nf90_create(adcircoutfile, ncformat, ncid)
  CALL netcdfcheckerr(ierr)

  CALL unsetmessagesource()

 END SUBROUTINE newadcircnetcdfoutfile

!================================================================================

 !----------------------------------------------------------------
 ! S U B R O U T I N E   N E T C D F   C H E C K   E R R
 !----------------------------------------------------------------
 !
 !----------------------------------------------------------------
 SUBROUTINE base_netcdfcheckerr(ierr, file, line)

  IMPLICIT NONE

  INTEGER, INTENT(IN)           :: ierr
  CHARACTER(LEN=*), INTENT(IN)  :: file
  INTEGER, INTENT(IN)           :: line

  CHARACTER(LEN=1024)           :: tmpSTR

  CALL setmessagesource("NetCDFCheckErr")

  IF (ierr /= nf90_noerr) THEN
   CALL allmessage(error, nf90_strerror(ierr))
   WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
   CALL allmessage(info, tmpstr)
   CALL netcdfterminate()
  END IF

  CALL unsetmessagesource()

 END SUBROUTINE base_netcdfcheckerr

!================================================================================

 !----------------------------------------------------------------
 ! S U B R O U T I N E   N E T C D F   T E R M I N A T E
 !----------------------------------------------------------------
 !
 !----------------------------------------------------------------
 SUBROUTINE netcdfterminate()

  USE version

  IMPLICIT NONE

  CALL setmessagesource("NetCDFTerminate")

  CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")
```

```
    CALL exit(1)

    CALL unsetmessagesource()

  END SUBROUTINE netcdfterminate

!================================================================================


  !----------------------------------------------------------------
  !  S U B R O U T I N E   W R I T E   N E T C D F   R E C O R D
  !----------------------------------------------------------------
  !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
  !----------------------------------------------------------------
  SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)

    IMPLICIT NONE

    CHARACTER(LEN=*), INTENT(IN) :: adcircOutFile

    INTEGER :: timeLoc
    INTEGER :: ncID, ierr, nodes
    INTEGER :: start(2), kount(2)

    CALL setmessagesource("WriteNetCDFRecord")

    ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
    CALL netcdfcheckerr(ierr)

    ! Set up the 2D netcdf data extents
    ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
    start(1) = 1
    start(2) = myfile%fileRecCounter
    kount(1) = nodes
    kount(2) = mytime%timeLen

    ierr = nf90_put_var(ncid, datatmpres%varID, wpress(:, timeloc), start, kount)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datwindx%varID, wvelx(:, timeloc), start, kount)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datwindy%varID, wvely(:, timeloc), start, kount)
     CALL netcdfcheckerr(ierr)

    ! Close netCDF file
    ierr = nf90_close(ncid)
      CALL netcdfcheckerr(ierr)

    DEALLOCATE(mytime%time)

    CALL unsetmessagesource()

  END SUBROUTINE writenetcdfrecord

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E  S E T   R E C O R D  C O U N T E R  A N D  S T O R E   T I M E
  !----------------------------------------------------------------
  !  author
  !----------------------------------------------------------------
  SUBROUTINE setrecordcounterandstoretime(ncID, f, t)

    IMPLICIT NONE

    INTEGER, INTENT(IN)              :: ncID
    TYPE(FileData_T), INTENT(INOUT) :: f
    TYPE(TimeData_T), INTENT(INOUT) :: t

    REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
    LOGICAL              :: timeFound      ! true if current time is in array of stored times

    INTEGER :: ndim      ! number of dimensions in the netcdf file
    INTEGER :: nvar      ! number of variables in the netcdf file
    INTEGER :: natt      ! number of attributes in the netcdf file

    INTEGER :: counti(1), starti(1)
    INTEGER :: ierr  ! success or failure of netcdf call
    INTEGER :: i     ! loop counter
```

```


    CALL setmessagesource("SetRecordCounterAndStoreTime")

    ! Inquire the time variable
    ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_inq_varid(ncid, 'time', t%timeID)
      CALL netcdfcheckerr(ierr)

    ! Determine the relationship between the current simulation time
    ! and the time array stored in the netcdf file. Set the record
    ! counter based on this relationship.
    IF (f%fileRecCounter /= 0) THEN
      ALLOCATE(storedtimes(f%fileRecCounter))
      ierr = nf90_get_var(ncid, t%timeID, storedtimes)
        CALL netcdfcheckerr(ierr)
      timefound = .false.

      DO i = 1, f%fileRecCounter
        IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
          timefound = .true.
          EXIT
        ENDIF
      END DO

      IF (timefound .EQV. .false.) THEN
        ! Increment the record counter so that we can store data at the
        ! next location in the netcdf file (i.e., all of the times
        ! in the netcdf file were found to be earlier than the current
        ! adcirc simulation time).
        f%fileRecCounter = f%fileRecCounter + 1
      ELSE
        ! set the counter at the index that reflects the
        ! current time within the netcdf file (or is between two times
        ! found in the netcdf file).
        ! WARNING: all subsequent data will remain in the file, we
        ! are just overwriting it ... if we don't overwrite all of it,
        ! the pre-existing data will still be there, which is probably
        ! not what the user intended ... but apparently there is no
        ! way to delete data from netcdf files:
        ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
        scratchformat = '("Overwriting pre-existing data in netcdf file ",a,' //   &
                        '" for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.")'
        WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
        CALL allmessage(info, scratchmessage)
        f%fileRecCounter = i
      ENDIF

      DEALLOCATE(storedtimes)
    ELSE
      ! set the counter at 1 so we can record our first time value
      f%fileRecCounter = 1
    ENDIF

    ! Store simulation time in netcdf file
    starti(1) = f%fileRecCounter
    counti(1) = t%timeLen
    ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
      CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE setrecordcounterandstoretime

!================================================================================

END MODULE pahm_netcdfio
```

9.24 netcdfio.F90 File Reference

Data Types

- type pahm_netcdfio::filedata_t

- type pahm_netcdfio::timedata_t

Modules

- module pahm_netcdfio

Macros

- #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

Functions/Subroutines

- subroutine pahm_netcdfio::initadcircnetcdfoutfile (adcircOutFile)

  Initializes a new NetCDF data file and puts it in define mode. Sets up netCDF dimensions and variables.
- subroutine pahm_netcdfio::newadcircnetcdfoutfile (ncID, adcircOutFile)

  Creates a new NetCDF data file and puts it in define mode.
- subroutine pahm_netcdfio::base_netcdfcheckerr (ierr, file, line)

  Checks the return value from netCDF calls; if there was an error, it writes the error message to the screen and to the log file.
- subroutine pahm_netcdfio::netcdfterminate ()

  Terminates the program on NetCDF error.
- subroutine pahm_netcdfio::writenetcdfrecord (adcircOutFile, timeLoc)

  Writes data to the NetCDF file.
- subroutine pahm_netcdfio::setrecordcounterandstoretime (ncID, f, t)

  Compares the current simulation time with the array of output times in the file, and if the simulation time is before the end of the file, it sets the record counter to the right place within the existing data. Data that occur after the inserted data will remain, due to the inability of netcdf to delete data from files.

9.24.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file netcdfio.F90.

9.24.2 Macro Definition Documentation

9.24.2.1 NetCDFCheckErr

```
#define NetCDFCheckErr(
    arg ) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
```

9.25 netcdfio.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                M O D U L E   N E T C D F  I O
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_netcdfio

  USE pahm_sizes
  USE pahm_messages
  USE pahm_global
  USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
  USE netcdf

#ifdef __INTEL_COMPILER
  USE ifport
#endif

  IMPLICIT NONE

#define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

  INTEGER, PRIVATE            :: ncFormat
  INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
  INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)

  INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
  INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID

  TYPE :: filedata_t
    LOGICAL                 :: initialized = .false.
    INTEGER                 :: fileRecCounter = 0
    CHARACTER(LEN=FNAMELEN) :: fileName
    LOGICAL                 :: fileFound = .false.  ! .true. if the netCDF file is present
  END TYPE filedata_t

  TYPE :: timedata_t
    LOGICAL :: initialized = .false.
    INTEGER :: timeLen = 1  ! number of time slices to write
    INTEGER :: timeDimID
    INTEGER :: timeID
    INTEGER :: timeDims(1)
    REAL(SZ), ALLOCATABLE :: time(:)
  END TYPE timedata_t

  TYPE, PRIVATE :: adcirccoorddata_t
    LOGICAL               :: initialized = .false.
    REAL(SZ)              :: initVal
    INTEGER               :: dimID
    INTEGER               :: varID
    INTEGER               :: varDimIDs
    INTEGER               :: varDims
    CHARACTER(50)         :: varname
    REAL(SZ), ALLOCATABLE :: var(:)
    INTEGER               :: start(1), count(1)
  END TYPE adcirccoorddata_t

  TYPE, PRIVATE :: adcircvardata_t
     LOGICAL               :: initialized = .false.
     REAL(SZ)              :: initVal
     INTEGER               :: varID
     INTEGER               :: varDimIDs(2)
     INTEGER               :: varDims(2)
     CHARACTER(50)         :: varname
     REAL(SZ), ALLOCATABLE :: var(:, :)
     INTEGER               :: start(2), count(2)
  END TYPE adcircvardata_t

  TYPE, PRIVATE :: adcircvardata3d_t
     LOGICAL               :: initialized = .false.
     REAL(SZ)              :: initVal
     INTEGER               :: varID
     INTEGER               :: varDimIDs(3)
     INTEGER               :: varDims(3)
     CHARACTER(50)         :: varname
     REAL(SZ), ALLOCATABLE :: var(:, :, :)
     INTEGER               :: start(3), count(3)
```

```
END TYPE adcircvardata3d_t

TYPE(FileData_T), SAVE   :: myFile
TYPE(TimeData_T), SAVE   :: myTime

TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdTime
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLons
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLats
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdXCs
TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdYCs

TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datElements
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datAtmPres
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datWindX
TYPE(AdcircVarData_T), PRIVATE, SAVE   :: datWindY


CONTAINS


!----------------------------------------------------------------
!  S U B R O U T I N E   I N I T  A D C I R C  N E T C D F  O U T  F I L E
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)

  USE version
  USE timedateutils, ONLY : gettimeconvsec, datetime2string

  IMPLICIT NONE

  CHARACTER(LEN=*), INTENT(INOUT) :: adcircOutFile

  INTEGER             :: ncID
  CHARACTER(LEN=64)   :: refDateTimeStr, modDateTimeStr, tmpVarName
  CHARACTER(LEN=128)  :: institution, source, history, comments, host, &
                         conventions, contact, references
  INTEGER             :: tvals(8)
  INTEGER             :: ierr ! success or failure of a netcdf call
  INTEGER             :: iCnt, jCnt

  LOGICAL, SAVE                        :: firstCall = .true.


  IF (firstcall) THEN
    firstcall = .false.

    CALL setmessagesource("InitAdcircNetCDFOutFile")


    refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units = unittime)

    institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
    source      = ''
    history     = ''
    comments    = ''
    host        = ''
    conventions = 'UGRID-0.9.0'
    contact     = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
    references  = ''


    ! Create the NetCDF output file.
    CALL newadcircnetcdfoutfile(ncid, adcircoutfile)

    !====================
    !===== (1) Define all the dimensions
    !====================
    tmpvarname = 'time'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'longitude'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
        CALL netcdfcheckerr(ierr)

    tmpvarname = 'latitude'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
        CALL netcdfcheckerr(ierr)

```

```
      tmpvarname = 'node'
        ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
          CALL netcdfcheckerr(ierr)

      tmpvarname = 'element'
        ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemdimid)
          CALL netcdfcheckerr(ierr)

      tmpvarname = 'noel'
        ierr = nf90_def_dim(ncid, trim(tmpvarname), 3,  vertdimid)
          CALL netcdfcheckerr(ierr)

      tmpvarname = 'mesh'
      ierr = nf90_def_dim(ncid, trim(tmpvarname), 1,    meshdimid)
        CALL netcdfcheckerr(ierr)

      !====================
      !===== (2) Define all the variables
      !====================
      !----- Time variable
      tmpvarname = 'time'
        crdtime%varname    = trim(tmpvarname)
        crdtime%varDimIDs = crdtime%dimID
        crdtime%varDims    = SIZE(times, 1)
        crdtime%start(1)   = 1
        crdtime%count(1)   = crdtime%varDims

        ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdtime%varID, 'long_name',     'model ' // trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdtime%varID, 'units',         trim(refdatetimestr))
          CALL netcdfcheckerr(ierr)

        ALLOCATE(crdtime%var(crdtime%varDims))
        crdtime%var = times * gettimeconvsec(unittime, 1)

      !----- Longitude variable
      tmpvarname = 'longitude'
        crdlons%varname = trim(tmpvarname)
        crdlons%varDimIDs = nodedimid
        ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlons%varDims)
          CALL netcdfcheckerr(ierr)
        crdlons%start(1) = 1
        crdlons%count(1) = crdlons%varDims

        ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlons%varID, 'long_name',     trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlons%varID, 'units',         'degrees_east')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlons%varID, 'positive',      'east')
          CALL netcdfcheckerr(ierr)

        ALLOCATE(crdlons%var(crdlons%varDims))
        crdlons%var = slam

      !----- Latitude variable
      tmpvarname = 'latitude'
        crdlats%varname = trim(tmpvarname)
        crdlats%varDimIDs = nodedimid
        ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlats%varDims)
          CALL netcdfcheckerr(ierr)
        crdlats%start(1) = 1
        crdlats%count(1) = crdlats%varDims

        ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlats%varID, 'long_name',     trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdlats%varID, 'units',         'degrees_north')
          CALL netcdfcheckerr(ierr)
```

```
      ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',     rmissv)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, crdlats%varID, 'positive',       'north')
        CALL netcdfcheckerr(ierr)

      ALLOCATE(crdlats%var(crdlats%varDims))
      crdlats%var = sfea

    !----- Element variable
    !----- We need to switch the order in array for NetCDF
    !----- It should be: elements(nf, icnt) and NOT elements(icnt, nf)
    tmpvarname = 'tri'
      datelements%varname = trim(tmpvarname)
      datelements%varDimIDs(1) = vertdimid
      datelements%varDimIDs(2) = elemdimid
      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(1), len = datelements%varDims(1))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(2), len = datelements%varDims(2))
        CALL netcdfcheckerr(ierr)
      datelements%start(1) = 1
      datelements%count(1) = datelements%varDims(1)
      datelements%start(2) = 1
      datelements%count(2) = datelements%varDims(2)

      ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID,'long_name',     trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID,'standard_name', trim(tmpvarname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'cf_role',      'face_node_connectivity')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'start_index',  1)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, 'units',        'nondimensional')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',    imissv)
        CALL netcdfcheckerr(ierr)

      ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
      DO icnt = 1, datelements%varDims(2)
        DO jcnt = 1, datelements%varDims(1)
          datelements%var(jcnt, icnt) = nm(icnt, jcnt)
        END DO
      END DO

    !----- Mesh variable
    tmpvarname = 'adcirc_mesh'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, meshvarid,'long_name',               'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid,'standard_name',           'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'cf_role',                'mesh_topology')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates',       'lon lat')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
        CALL netcdfcheckerr(ierr)

    !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
    tmpvarname = 'projection'
      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_att(ncid, projvarid,'long_name',         'equidistant cylindrical projection')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid,'standard_name',     'CPP')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'node_coordinates', 'x y')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lon0',             slam0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'lat0',             sfea0)
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, projvarid, 'earth_radius',     rearth)
        CALL netcdfcheckerr(ierr)

    !----- CPP CPP x-coordinates
```

```
      tmpvarname = 'x'
        crdxcs%varname = trim(tmpvarname)
        crdxcs%dimID = nodedimid
        crdxcs%varDimIDs = nodedimid
        ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
          CALL netcdfcheckerr(ierr)
        crdxcs%start(1) = 1
        crdxcs%count(1) = crdxcs%varDims

        ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',     'CPP x coordinate')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdxcs%varID, 'units',         'm')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)

        ALLOCATE(crdxcs%var(crdxcs%varDims))
        crdxcs%var = xcslam

      !----- CPP y-coordinates
      tmpvarname = 'y'
        crdycs%varname = trim(tmpvarname)
        crdycs%dimID = nodedimid
        crdycs%varDimIDs = nodedimid
        ierr = nf90_inquire_dimension(ncid, crdycs%dimID, len = crdycs%varDims)
          CALL netcdfcheckerr(ierr)
        crdycs%start(1) = 1
        crdycs%count(1) = crdycs%varDims

        ierr = nf90_def_var(ncid, trim(crdycs%varname), nf90_double, crdycs%varDimIDs, crdycs%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdycs%varID, 'long_name',     'CPP y coordinate')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdycs%varID, 'standard_name', 'cpp_y')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdycs%varID, 'units',         'm')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, crdycs%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)

        ALLOCATE(crdycs%var(crdycs%varDims))
        crdycs%var = ycsfea

      !----- Atmospheric Pressure variable
      tmpvarname = trim(ncvarnam_pres)
        datatmpres%varname       = trim(tmpvarname)
        datatmpres%varDimIDs(1) = nodedimid
        datatmpres%varDimIDs(2) = crdtime%dimID
        datatmpres%varDims(1)   = SIZE(wpress, 1)
        datatmpres%varDims(2)   = crdtime%varDims
        datatmpres%start(1)     = 1
        datatmpres%count(1)     = datatmpres%varDims(1)
        datatmpres%start(2)     = 1
        datatmpres%count(2)     = datatmpres%varDims(2)

        ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
                            datatmpres%varDimIDs, datatmpres%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',     'air pressure at sea level')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'units',         'Pa')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',   'time lat lon')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'location',      'node')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',          'adcirc_mesh')
          CALL netcdfcheckerr(ierr)

!PV     ALLOCATE(datAtmPres%var(datAtmPres%varDims(1), datAtmPres%varDims(2)))
!PV     datAtmPres%var = wPress

      !----- Wind velocity variables
```

```
      ! Eastward
      tmpvarname = trim(ncvarnam_wndx)
        datwindx%varname       = trim(tmpvarname)
        datwindx%varDimIDs(1) = nodedimid
        datwindx%varDimIDs(2) = crdtime%dimID
        datwindx%varDims(1)    = SIZE(wvelx, 1)
        datwindx%varDims(2)    = crdtime%varDims
        datwindx%start(1)      = 1
        datwindx%count(1)      = datwindx%varDims(1)
        datwindx%start(2)      = 1
        datwindx%count(2)      = datwindx%varDims(2)

        ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
                          datwindx%varDimIDs, datwindx%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',     '10-m eastward wind component')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'units',         'm s-1')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates',   'time lat lon')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'location',      'node')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',          'adcirc_mesh')
          CALL netcdfcheckerr(ierr)

!PV    ALLOCATE(datWindX%var(datWindX%varDims(1), datWindX%varDims(2)))
!PV    datWindX%var = wVelX

      ! Northward
      tmpvarname = trim(ncvarnam_wndy)
        datwindy%varname       = trim(tmpvarname)
        datwindy%varDimIDs(1) = nodedimid
        datwindy%varDimIDs(2) = crdtime%dimID
        datwindy%varDims(1)    = SIZE(wvely, 1)
        datwindy%varDims(2)    = crdtime%varDims
        datwindy%start(1)      = 1
        datwindy%count(1)      = datwindy%varDims(1)
        datwindy%start(2)      = 1
        datwindy%count(2)      = datwindy%varDims(2)

        ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
                          datwindy%varDimIDs, datwindy%varID)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',     '10-m northward wind component')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'units',         'm s-1')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',    rmissv)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates',   'time lat lon')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'location',      'node')
          CALL netcdfcheckerr(ierr)
        ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',          'adcirc_mesh')
          CALL netcdfcheckerr(ierr)

!PV    ALLOCATE(datWindY%var(datWindY%varDims(1), datWindY%varDims(2)))
!PV    datWindY%var = wVelY

      !====================
      !===== (3) Set Deflate parameters if requested by the user
      !====================
#ifdef NETCDF_CAN_DEFLATE
      IF (ncformat == nc4form) THEN
        ierr = nf90_def_var_deflate(ncid, crdlons%varID,     ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, crdlats%varID,     ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, crdxcs%varID,     ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, crdycs%varID,     ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
```

```
        ierr = nf90_def_var_deflate(ncid, datatmpres%varID,  ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
        ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, ncdlevel)
          CALL netcdfcheckerr(ierr)
      END IF
#endif

      !====================
      !===== (4) Global metadata definitions and variables
      !====================
      ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
   ')')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
        CALL netcdfcheckerr(ierr)

      CALL date_and_time(values = tvals)
      WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
      moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
   moddatetimestr)

      ierr = nf90_put_att(ncid, nf90_global,'creation_date', trim(moddatetimestr))
        CALL netcdfcheckerr(ierr)
      ierr = nf90_put_att(ncid, nf90_global,'modification_date', trim(moddatetimestr))
        CALL netcdfcheckerr(ierr)

      !----- Finalize the definitions in the NetCDF file
      ierr = nf90_enddef(ncid)
        CALL netcdfcheckerr(ierr)

      !====================
      !===== (5) Put the static data into the NetCDF file and then close it
      !====================
      ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_var(ncid, crdycs%varID, crdycs%var, crdycs%start, crdycs%count)
        CALL netcdfcheckerr(ierr)

      ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
        CALL netcdfcheckerr(ierr)

 !PV    ierr = NF90_PUT_VAR(ncID, datElements%varID, datElements%var, datElements%start,
   datElements%count)
 !PV      CALL NetCDFCheckErr(ierr)

 !PV     ierr = NF90_PUT_VAR(ncID, datAtmPres%varID, datAtmPres%var, datAtmPres%start, datAtmPres%count)
 !PV      CALL NetCDFCheckErr(ierr)
```

```
  !PV     ierr = NF90_PUT_VAR(ncID, datWindX%varID, datWindX%var, datWindX%start, datWindX%count)
  !PV      CALL NetCDFCheckErr(ierr)

  !PV     ierr = NF90_PUT_VAR(ncID, datWindY%varID, datWindY%var, datWindY%start, datWindY%count)
  !PV      CALL NetCDFCheckErr(ierr)


      !---------- (16) Set all the "initialized" flags to .TRUE.
      crdlons%initialized      = .true.
      crdlats%initialized      = .true.
      crdxcs%initialized       = .true.
      crdycs%initialized       = .true.
      datelements%initialized  = .true.
      datatmpres%initialized   = .true.
      datwindx%initialized     = .true.
      datwindy%initialized     = .true.

      myfile%fileName    = adcircoutfile
      myfile%initialized = .true.

      !----- Close the NetCDF file
      ierr = nf90_close(ncid)
        CALL netcdfcheckerr(ierr)

      CALL unsetmessagesource()

    END IF !firstCall

  END SUBROUTINE initadcircnetcdfoutfile

!=================================================================================

  !-----------------------------------------------------------------
  !  S U B R O U T I N E   N E W  A D C I R C  N E T C D F  O U T  F I L E
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)

    IMPLICIT NONE

    INTEGER, INTENT(OUT)              :: ncID
    CHARACTER(LEN=*), INTENT(INOUT)   :: adcircOutFile

    LOGICAL                           :: fileFound = .false.
    CHARACTER(LEN=FNAMELEN)           :: outFile, sys_cmd
    CHARACTER(LEN=14)                 :: fext, date_time
    INTEGER                           :: pos, ierr, tvals(8)


    CALL setmessagesource("NewAdcircNetCDFOutFile")

    !----------
    ! Set some variables that depend upon the type of NetCDF supported.
#if defined(HAVE_NETCDF4)
    fext = ".nc4"
    ncformat = nc4form
#else
    fext = ".nc"
    ncformat = nc3form
#endif

    !----------
    ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
    ! extension in the filename; re-define the adcircOutFile variable.
    pos = scan(trim(adcircoutfile), ".", back= .true.)
    IF (pos > 0) THEN
      adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
    ELSE
      adcircoutfile = trim(adcircoutfile) // trim(fext)
    END IF

    !----------
    ! If the adcircOutFile exists then rename it to:
    !   adcircOutFile-YYYYMMDDhhmmss.
    ! The user can remove these files afterwards.
    INQUIRE(file=adcircoutfile, exist=filefound)
    IF (filefound) THEN
      CALL date_and_time(values = tvals)
      WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
      outfile = trim(adcircoutfile) // "-" // trim(date_time)
```

```
      sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
      ierr = system(trim(sys_cmd))
      IF (ierr == 0) THEN
        WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
        CALL logmessage(info, scratchmessage)
        filefound = .false.
      ELSE
        WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' // trim(outfile)
        CALL logmessage(error, scratchmessage)
      END IF
    END IF

    IF (filefound) THEN
      WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adcircoutfile) // ' exists. Remove the file to proceed.'
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      CALL netcdfterminate
    END IF

    WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adcircoutfile) // ' and putting it in define mode.'
    CALL logmessage(info, scratchmessage)

    ! Create the NetCDF file
    ierr = nf90_create(adcircoutfile, ncformat, ncid)
    CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE newadcircnetcdfoutfile

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   N E T C D F   C H E C K   E R R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE base_netcdfcheckerr(ierr, file, line)

    IMPLICIT NONE

    INTEGER, INTENT(IN)           :: ierr
    CHARACTER(LEN=*), INTENT(IN)  :: file
    INTEGER, INTENT(IN)           :: line

    CHARACTER(LEN=1024)           :: tmpSTR

    CALL setmessagesource("NetCDFCheckErr")

    IF (ierr /= nf90_noerr) THEN
      CALL allmessage(error, nf90_strerror(ierr))
      WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
      CALL allmessage(info, tmpstr)
      CALL netcdfterminate()
    END IF

    CALL unsetmessagesource()

  END SUBROUTINE base_netcdfcheckerr

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   N E T C D F   T E R M I N A T E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE netcdfterminate()

    USE version

    IMPLICIT NONE

    CALL setmessagesource("NetCDFTerminate")

    CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")

    CALL exit(1)
```

```

    CALL unsetmessagesource()

  END SUBROUTINE netcdfterminate

!================================================================================


  !-----------------------------------------------------------------
  !  S U B R O U T I N E   W R I T E   N E T C D F   R E C O R D
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)

    USE timedateutils, ONLY : gettimeconvsec

    IMPLICIT NONE

    CHARACTER(LEN=*), INTENT(IN) :: adcircOutFile

    INTEGER :: timeLoc
    INTEGER :: ncID, ierr, nodes
    INTEGER :: start(2), kount(2)


    CALL setmessagesource("WriteNetCDFRecord")

    ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
    CALL netcdfcheckerr(ierr)

    ! Set up the 2D netcdf data extents
    ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
    start(1) = 1
    start(2) = timeloc
    kount(1) = nodes
    kount(2) = 1

    ierr = nf90_put_var(ncid, datatmpres%varID, wpress, start, kount)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datwindx%varID, wvelx, start, kount)
      CALL netcdfcheckerr(ierr)

    ierr = nf90_put_var(ncid, datwindy%varID, wvely, start, kount)
     CALL netcdfcheckerr(ierr)

    ! Close netCDF file
    ierr = nf90_close(ncid)
      CALL netcdfcheckerr(ierr)

    CALL unsetmessagesource()

  END SUBROUTINE writenetcdfrecord

!================================================================================


  !-----------------------------------------------------------------
  !  S U B R O U T I N E   S E T   R E C O R D   C O U N T E R   A N D   S T O R E   T I M E
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  SUBROUTINE setrecordcounterandstoretime(ncID, f, t)

    IMPLICIT NONE

    INTEGER, INTENT(IN)             :: ncID
    TYPE(FileData_T), INTENT(INOUT) :: f
    TYPE(TimeData_T), INTENT(INOUT) :: t

    REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
    LOGICAL               :: timeFound      ! true if current time is in array of stored times

    INTEGER :: ndim      ! number of dimensions in the netcdf file
    INTEGER :: nvar      ! number of variables in the netcdf file
    INTEGER :: natt      ! number of attributes in the netcdf file

    INTEGER :: counti(1), starti(1)
    INTEGER :: ierr  ! success or failure of netcdf call
    INTEGER :: i     ! loop counter


    CALL setmessagesource("SetRecordCounterAndStoreTime")
```

```

  ! Inquire the time variable
  ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
    CALL netcdfcheckerr(ierr)

  ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
    CALL netcdfcheckerr(ierr)

  ierr = nf90_inq_varid(ncid, 'time', t%timeID)
    CALL netcdfcheckerr(ierr)

  ! Determine the relationship between the current simulation time
  ! and the time array stored in the netcdf file. Set the record
  ! counter based on this relationship.
  IF (f%fileRecCounter /= 0) THEN
    ALLOCATE(storedtimes(f%fileRecCounter))
    ierr = nf90_get_var(ncid, t%timeID, storedtimes)
      CALL netcdfcheckerr(ierr)
    timefound = .false.

    DO i = 1, f%fileRecCounter
      IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
        timefound = .true.
        EXIT
      ENDIF
    END DO

    IF (timefound .EQV. .false.) THEN
      ! Increment the record counter so that we can store data at the
      ! next location in the netcdf file (i.e., all of the times
      ! in the netcdf file were found to be earlier than the current
      ! adcirc simulation time).
      f%fileRecCounter = f%fileRecCounter + 1
    ELSE
      ! set the counter at the index that reflects the
      ! current time within the netcdf file (or is between two times
      ! found in the netcdf file).
      ! WARNING: all subsequent data will remain in the file, we
      ! are just overwriting it ... if we don't overwrite all of it,
      ! the pre-existing data will still be there, which is probably
      ! not what the user intended ... but apparently there is no
      ! way to delete data from netcdf files:
      ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
      scratchformat = '("Overwriting pre-existing data in netcdf file ",a,' //   &
                      '" for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.")'
      WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
      CALL allmessage(info, scratchmessage)
      f%fileRecCounter = i
    ENDIF

    DEALLOCATE(storedtimes)
  ELSE
    ! set the counter at 1 so we can record our first time value
    f%fileRecCounter = 1
  ENDIF

  ! Store simulation time in netcdf file
  starti(1) = f%fileRecCounter
  counti(1) = t%timeLen
  ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
    CALL netcdfcheckerr(ierr)

  CALL unsetmessagesource()

 END SUBROUTINE setrecordcounterandstoretime

!================================================================================

END MODULE pahm_netcdfio
```

9.26 pahm.F90 File Reference

Main PaHM program, calls Init, Run and Finalize procedures.

Functions/Subroutines

- program pahm

9.26.1 Detailed Description

Main PaHM program, calls Init, Run and Finalize procedures.

1) Initialize PaHM by establishing the logging facilities and calling the subroutine "GetProgramCmdlArgs" to get possible command line arguments and set the defaults. During the initialization stage, PaHM reads the mandatory input control file (defaults to pahm_control.in) to read in the definitions of different variables used in PaHM. At this stage we read the mesh/grid of the domain or the generic mesh/grid input file and the list of best track files supplied by the user.

2) Start the PaHM run (timestepping).

3) Finalize the PaHM run and exit the program.

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file pahm.F90.

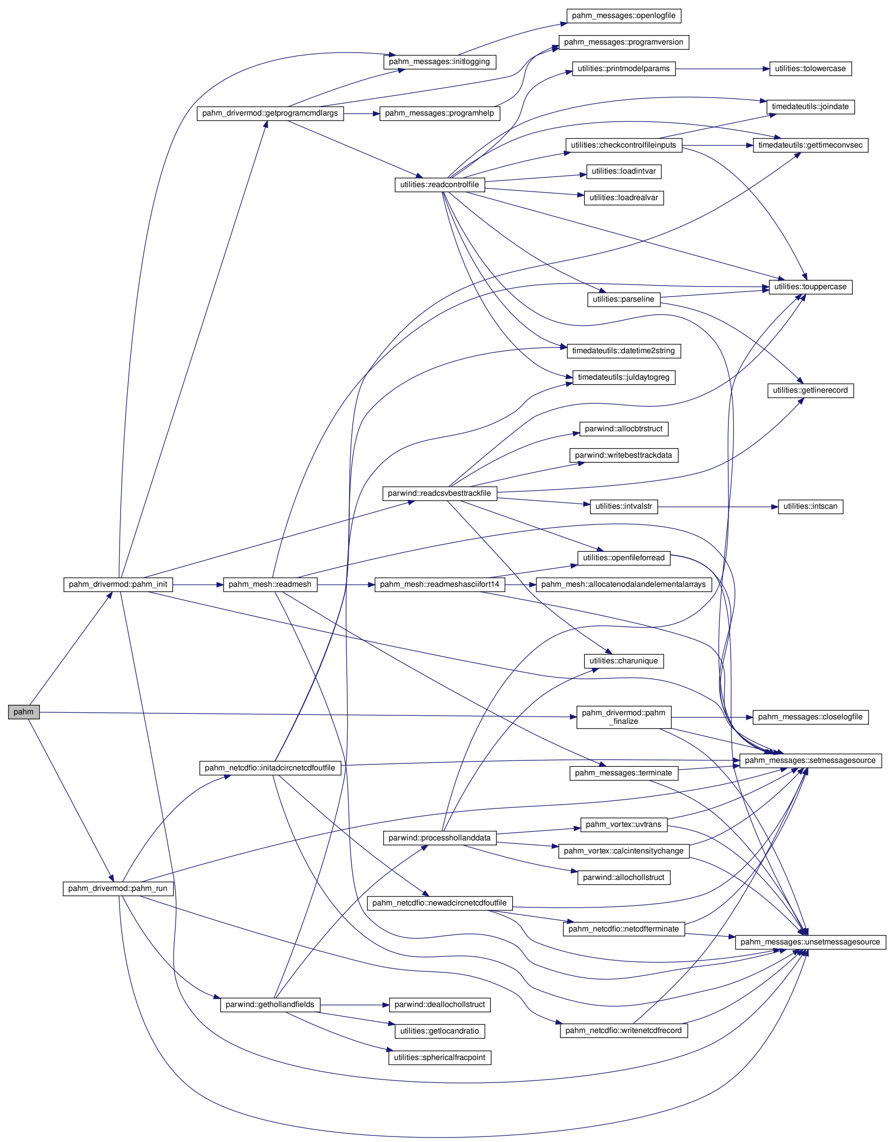
9.26.2 Function/Subroutine Documentation

9.26.2.1 pahm() `program pahm`

Definition at line 26 of file pahm.F90.

References pahm_drivermod::pahm_finalize(), pahm_drivermod::pahm_init(), and pahm_drivermod::pahm_run().

Here is the call graph for this function:



9.27 pahm.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                 P R O G R A M   P A H M
!----------------------------------------------------------------
!----------------------------------------------------------------

PROGRAM pahm

  USE pahm_drivermod, ONLY : pahm_init, pahm_run, pahm_finalize

  IMPLICIT NONE

  CALL pahm_init()
```

```
  CALL pahm_run()

  CALL pahm_finalize()

END PROGRAM pahm
```

9.28 parwind-orig.F90 File Reference

Data Types

- type parwind::besttrackdata_t
- type parwind::hollanddata_t

Modules

- module parwind

Functions/Subroutines

- subroutine parwind::readbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine parwind::readcsvbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine parwind::processhollanddata (idTrFile, strOut, status)

  Subroutine to support the Holland model (GetHolland). Gets the next line from the file, skipping lines that are time repeats.
- subroutine parwind::gethollandfields ()

  Calculate wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.
- subroutine parwind::writebesttrackdata (inpFile, btrStruc, suffix)

  Writes the best track data (adjusted or not) to the "adjusted" best track output file.
- subroutine parwind::allocbtrstruct (str, nRec)

  Subroutine to allocate memory for a best track structure.
- subroutine parwind::deallocbtrstruct (str)

  Subroutine to deallocate the memory allocated for a best track structure.
- subroutine parwind::allochollstruct (str, nRec)

  Subroutine to allocate memory for a holland structure.
- subroutine parwind::deallochollstruct (str)

  Subroutine to deallocate memory of an allocated holland structure.

Variables

- real(sz) parwind::windreftime
- type(besttrackdata_t), dimension(:), allocatable, target parwind::besttrackdata

9.29 parwind-orig.F90

Go to the documentation of this file.

```
!-----------------------------------------------------------------
!                M O D U L E   P A R W I N D
!-----------------------------------------------------------------
!-----------------------------------------------------------------

MODULE parwind

  USE pahm_sizes
  USE pahm_messages

  ! switch to turn on or off geostrophic balance in GAHM
  ! on (default): Coriolis term included, phiFactors will be calculated before being used
  ! off         : parameter is set to 'TRUE', phiFactors will be set to constant 1
  !LOGICAL :: geostrophicSwitch = .TRUE.
  !INTEGER :: geoFactor = 1               !turn on or off gostrophic balance

  REAL(sz) :: windreftime !jgf46.29 seconds since beginning of year, this          !PV check
                          !corresponds to time=0 of the simulation

  !-----------------------------------------------------------------
  ! The BestTrackData_T structure holds all data read from the best track files(s)
  ! in ATCF format (a-deck/b-deck)
  !-----------------------------------------------------------------
  TYPE besttrackdata_t
    CHARACTER(LEN=FNAMELEN)           :: filename         ! full path to the best track file
    CHARACTER(LEN=10)                 :: thisstorm        ! the name of the "named" storm
    LOGICAL                           :: loaded = .false. ! .TRUE. if we have loaded the data from file
    INTEGER                           :: numrec           ! number of records in the structure

    !----- input data from best track file (ATCF format)
    CHARACTER(LEN=2), ALLOCATABLE     :: basin(:)         ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
    INTEGER, ALLOCATABLE              :: cynum(:)         ! annual cyclone number: 1 - 99
    CHARACTER(LEN=10), ALLOCATABLE    :: dtg(:)           ! warning Date-Time-Group (DTG), YYYYMMDDHH
    INTEGER, ALLOCATABLE              :: technum(:)       ! objective technique sorting number, minutes for best track: 00 - 99
    CHARACTER(LEN=4), ALLOCATABLE     :: tech(:)          ! acronym for each objective technique or CARQ or WRNG,
                                                          ! BEST for best track, up to 4 chars.
    INTEGER, ALLOCATABLE              :: tau(:)           ! forecast period: -24 through 240 hours, 0 for best-track,
                                                          ! negative taus used for CARQ and WRNG records.
    INTEGER, ALLOCATABLE              :: intlat(:)        ! latitude for the DTG: 0 - 900 tenths of degrees
    INTEGER, ALLOCATABLE              :: intlon(:)        ! latitude for the DTG: 0 - 900 tenths of degrees
    CHARACTER(LEN=1), ALLOCATABLE     :: ew(:)            ! E/W
    CHARACTER(LEN=1), ALLOCATABLE     :: ns(:)            ! N/S

    INTEGER, ALLOCATABLE              :: intvmax(:)       ! maximum sustained wind speed in knots: 0 - 300 kts
    INTEGER, ALLOCATABLE              :: intmslp(:)       ! minimum sea level pressure, 850 - 1050 mb
    CHARACTER(LEN=2), ALLOCATABLE     :: ty(:)            ! Highest level of tc development:
                                                          !   DB - disturbance,
                                                          !   TD - tropical depression,
                                                          !   TS - tropical storm,
                                                          !   TY - typhoon,
                                                          !   ST - super typhoon,
                                                          !   TC - tropical cyclone,
                                                          !   HU - hurricane,
                                                          !   SD - subtropical depression,
                                                          !   SS - subtropical storm,
                                                          !   EX - extratropical systems,
                                                          !   PT - post tropical,
                                                          !   IN - inland,
                                                          !   DS - dissipating,
                                                          !   LO - low,
                                                          !   WV - tropical wave,
                                                          !   ET - extrapolated,
                                                          !   MD - monsoon depression,
                                                          !   XX - unknown.
    INTEGER, ALLOCATABLE              :: rad(:)           ! wind intensity for the radii defined in this record: 34, 50 or 64 kt
    CHARACTER(LEN=3), ALLOCATABLE     :: windcode(:)      ! radius code:
                                                          !   AAA - full circle
                                                          !   NEQ, SEQ, SWQ, NWQ - quadrant
    INTEGER, ALLOCATABLE              :: intrad1(:)       ! if full circle, radius of specified wind intensity, or radius of
                                                          ! first quadrant wind intensity as specified by WINDCODE.  0 - 999 n mi
```

```
   INTEGER, ALLOCATABLE             :: intrad2(:)       ! if full circle this field not used, or radius of 2nd quadrant wind
                                                        ! intensity as specified by WINDCODE.  0 - 999 n mi
   INTEGER, ALLOCATABLE             :: intrad3(:)       ! if full circle this field not used, or radius of 3rd quadrant wind
                                                        ! intensity as specified by WINDCODE.  0 - 999 n mi
   INTEGER, ALLOCATABLE             :: intrad4(:)       ! if full circle this field not used, or radius of 4th quadrant wind
                                                        ! intensity as specified by WINDCODE.  0 - 999 n mi
   INTEGER, ALLOCATABLE             :: intpouter(:)     ! pressure in millibars of the last closed isobar, 900 - 1050 mb
   INTEGER, ALLOCATABLE             :: introuter(:)     ! radius of the last closed isobar, 0 - 999 n mi
   INTEGER, ALLOCATABLE             :: intrmw(:)        ! radius of max winds, 0 - 999 n mi
   INTEGER, ALLOCATABLE             :: gusts(:)         ! gusts, 0 - 999 kt
   INTEGER, ALLOCATABLE             :: eye(:)           ! eye diameter, 0 - 120 n mi
   CHARACTER(LEN=3), ALLOCATABLE    :: subregion(:)     ! subregion code: W,A,B,S,P,C,E,L,Q
                                                        !   A - Arabian Sea
                                                        !   B - Bay of Bengal
                                                        !   C - Central Pacific
                                                        !   E - Eastern Pacific
                                                        !   L - Atlantic
                                                        !   P - South Pacific (135E - 120W)
                                                        !   Q - South Atlantic
                                                        !   S - South IO (20E - 135E)
                                                        !   W - Western Pacific
   INTEGER, ALLOCATABLE             :: maxseas(:)       ! max seas: 0 - 999 ft
   CHARACTER(LEN=3), ALLOCATABLE    :: initials(:)      ! forecaster's initials used for tau 0 WRNG or OFCL, up to 3 chars
   INTEGER, ALLOCATABLE             :: dir(:)           ! storm direction, 0 - 359 degrees
   INTEGER, ALLOCATABLE             :: intspeed(:)      ! storm speed, 0 - 999 kts
   CHARACTER(LEN=10), ALLOCATABLE   :: stormname(:)     ! literal storm name, number, NONAME or INVEST, or TCcyx where:
                                                        !   cy = Annual cyclone number 01 - 99
                                                        !   x  = Subregion code: W,A,B,S,P,C,E,L,Q.
   INTEGER, ALLOCATABLE             :: cyclenum(:)      ! the cycle number !PV check if this is OK

!    !----- converted data from the above values (if needed)
   INTEGER, DIMENSION(:), ALLOCATABLE  :: year, month, day, hour
   REAL(sz), DIMENSION(:), ALLOCATABLE :: lat, lon
  END TYPE besttrackdata_t

  ! Array of info about the best track data (extension to use multiple storms)
  TYPE(besttrackdata_t), ALLOCATABLE, TARGET :: besttrackdata(:)

  !----------------------------------------------------------------
  ! The HollandData_T structure holds all required data for the Holland model
  ! The data are filtered to only include unique DTGs
  !----------------------------------------------------------------
  TYPE hollanddata_t
   CHARACTER(LEN=FNAMELEN)             :: filename         ! full path to the best track file
   CHARACTER(LEN=10)                   :: thisstorm        ! the name of the "named" storm
   LOGICAL                             :: loaded = .false. ! .TRUE. if we have loaded the data from file
   INTEGER                             :: numrec           ! number of records in the structure

   CHARACTER(LEN=2),       ALLOCATABLE :: basin(:)         ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
   INTEGER, ALLOCATABLE                :: stormnumber(:)   ! annual cyclone number: 1 - 99
   CHARACTER(LEN=10),      ALLOCATABLE :: dtg(:)           ! warning Date-Time-Group (DTG), YYYYMMDDHH
   INTEGER, DIMENSION(:),  ALLOCATABLE :: year, month, day, hour
   REAL(sz), ALLOCATABLE               :: casttime(:)      ! converted to decimal E/N (lon, lat)
   CHARACTER(LEN=4),       ALLOCATABLE :: casttype(:)      ! BEST, OFCL, CALM, ...
   INTEGER,                ALLOCATABLE :: fcstinc(:)       ! forecast period: -24 through 240 hours, 0 for best-track

   INTEGER, DIMENSION(:),  ALLOCATABLE :: ilat, ilon       ! latitude, longitude for the GTD
   REAL(sz), DIMENSION(:), ALLOCATABLE :: lat, lon         ! converted to decimal E/N (lon, lat)

   INTEGER,                ALLOCATABLE :: ispeed(:)        ! maximum sustained wind speed in knots: 0 - 300 kts
   REAL(sz),               ALLOCATABLE :: speed(:)         ! converted from kts to m/s

   INTEGER,                ALLOCATABLE :: icpress(:)       ! minimum sea level pressure, 850 - 1050 mb
   REAL(sz),               ALLOCATABLE :: cpress(:)        ! converted to Pa

   INTEGER,                ALLOCATABLE :: irrp(:)          ! radius of the last closed isobar, 0 - 999 n mi
   REAL(sz),               ALLOCATABLE :: rrp(:)           ! converted from nm to m

   INTEGER,                ALLOCATABLE :: irmw(:)          ! radius of max winds, 0 - 999 n mi
```

```
   REAL(sz),               ALLOCATABLE :: rmw(:)            ! converted from nm to m

   REAL(sz), DIMENSION(:), ALLOCATABLE :: cprdt             ! central pressure intensity change (Pa / h)
   REAL(sz), DIMENSION(:), ALLOCATABLE :: trvx, trvy        ! translational velocity components (x, y) of the
                                                            ! moving hurricane (m/s)
 END TYPE hollanddata_t


 CONTAINS


 !----------------------------------------------------------------
 !  S U B R O U T I N E   R E A D  B E S T  T R A C K  F I L E
 !----------------------------------------------------------------
 !----------------------------------------------------------------
 SUBROUTINE readbesttrackfile()

   USE pahm_global, ONLY : lun_btrk, lun_btrk1, nbtrfiles, besttrackfilename
   USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
   USE sortutils, ONLY : arth, indexx, arrayequal

   IMPLICIT NONE

   CHARACTER(LEN=FNAMELEN)        :: inpFile
   CHARACTER(LEN=512)             :: inpLine, line
   CHARACTER(LEN=512)             :: fmtStr

   INTEGER                        :: lenLine
   INTEGER                        :: nLines                 ! Number of lines counter
   INTEGER                        :: iFile, iCnt            ! loop counters
   INTEGER                        :: iUnit, errIO, ios, status

   CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
   INTEGER, ALLOCATABLE           :: idxArrStr(:)
   INTEGER                        :: nUnique, maxCnt, kCnt, kMax

   INTEGER, ALLOCATABLE           :: idx0(:), idx1(:)

   !---------- Initialize variables
   iunit = lun_btrk
   errio = 0

   fmtstr = '(a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x, a2, '
     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i4, 2x, i3, 2x, i4, 2x, i3, '
     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, i3, 2x, i3, 2x, a11, 2x, i3)'
   !----------

   CALL setmessagesource("ReadBestTrackFile")

   ! Allocate the best track structure array. This structure holds all the
   ! input values for the storm track as read in from the track input file
   ! (a-deck, b-deck ATCF format) as well as the converted best track variables
   ! (as appropriate).
   ALLOCATE(besttrackdata(nbtrfiles))

   ! This is the main loop. We loop through all the best track files
   ! (user input)
   DO ifile = 1, nbtrfiles
     inpfile = besttrackfilename(ifile)

     CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)

     IF (errio /= 0) THEN
       WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
       CALL allmessage(error, scratchmessage)

       CALL unsetmessagesource()

       CALL terminate()
     ELSE
       WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
       CALL logmessage(info, scratchmessage)
     END IF

     besttrackdata(ifile)%fileName  = trim(adjustl(inpfile))
     besttrackdata(ifile)%thisStorm = ""
     besttrackdata(ifile)%loaded    = .false.
     besttrackdata(ifile)%numRec    = -1

```

```
      ! Count the number of non-empty or commented out lines in the file.
      ! Comments are are considered those lines with the first non-blank character of "!" or "#"
      nlines = 0
      DO
        READ(unit=iunit, fmt='(a)', err=10, END=5, IOSTAT=errIO) inpline

        lenline = getlinerecord(inpline, line)
        IF (lenline /= 0) nlines = nlines + 1
      END DO
      5 rewind(unit=iunit)

      ! Array allocation in the structure bestTrackData
      CALL allocbtrstruct(besttrackdata(ifile), nlines)

      icnt = 0
      DO WHILE (.true.)
        READ(unit=iunit, fmt='(a)', err=10, END=20, IOSTAT=errIO) inpline

        lenline = getlinerecord(inpline, line)

        IF (lenline /= 0) THEN
          icnt = icnt + 1
          READ(line, fmt=fmtstr, err=11, iostat=ios)                                    &
            besttrackdata(ifile)%basin(icnt),     besttrackdata(ifile)%cyNum(icnt),      &
            besttrackdata(ifile)%dtg(icnt),       besttrackdata(ifile)%techNum(icnt),    &
            besttrackdata(ifile)%tech(icnt),      besttrackdata(ifile)%tau(icnt),        &
            besttrackdata(ifile)%intLat(icnt),    besttrackdata(ifile)%ns(icnt),         &
            besttrackdata(ifile)%intLon(icnt),    besttrackdata(ifile)%ew(icnt),         &
            besttrackdata(ifile)%intVMax(icnt),   besttrackdata(ifile)%intMslp(icnt),    &
            besttrackdata(ifile)%ty(icnt),        besttrackdata(ifile)%rad(icnt),        &
            besttrackdata(ifile)%windCode(icnt),  besttrackdata(ifile)%intRad1(icnt),    &
            besttrackdata(ifile)%intRad2(icnt),   besttrackdata(ifile)%intRad3(icnt),    &
            besttrackdata(ifile)%intRad4(icnt),   besttrackdata(ifile)%intPOuter(icnt),  &
            besttrackdata(ifile)%intROuter(icnt), besttrackdata(ifile)%intRmw(icnt),     &
            besttrackdata(ifile)%gusts(icnt),     besttrackdata(ifile)%eye(icnt),        &
            besttrackdata(ifile)%subregion(icnt), besttrackdata(ifile)%maxseas(icnt),    &
            besttrackdata(ifile)%initials(icnt),  besttrackdata(ifile)%dir(icnt),        &
            besttrackdata(ifile)%intSpeed(icnt),  besttrackdata(ifile)%stormName(icnt),  &
            besttrackdata(ifile)%cycleNum(icnt)

          !---------- Convert lat/lon values to S/N and W/E notations
          IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
            besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
          ELSE
            besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
          END IF

          IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
            besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
          ELSE
            besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
          END IF
          !----------

          !---------- Get the year, month, day, hour from the DGT string
          READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios) besttrackdata(ifile)%year(icnt)
            IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%month(icnt)
            IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%day(icnt)
            IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%hour(icnt)
            IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
          !----------
        END IF
      END DO

      10 IF (errio /= 0) THEN
           WRITE(scratchmessage, '(a)') 'Error in file: '  // trim(adjustl(inpfile)) // &
                                        ', while processing line: ' // trim(adjustl(inpline))
           CALL allmessage(error, scratchmessage)

           CLOSE(iunit)

           CALL unsetmessagesource()

           CALL terminate()
```

```
   END IF

11 IF (ios /= 0) THEN
     WRITE(scratchmessage, '(a)') 'Error in file: '  // trim(adjustl(inpfile)) // &
                                  ', while processing line: ' // trim(adjustl(line))
     CALL allmessage(error, scratchmessage)

     CLOSE(iunit)

     CALL unsetmessagesource()

     CALL terminate()
   END IF

20 CLOSE(iunit)

besttrackdata(ifile)%thisStorm = ''
besttrackdata(ifile)%loaded    = .true.
besttrackdata(ifile)%numRec    = nlines


!------------------------------------------------------------
! Get the unique storm name and store it in the thisStorm string
ALLOCATE(chkarrstr(nlines))
ALLOCATE(idxarrstr(nlines))

nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)

maxcnt = -1
DO kcnt = 1, nunique
  kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
  IF (kmax > maxcnt) THEN
    maxcnt = kmax
    besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
  END IF
END DO

DEALLOCATE(chkarrstr)
DEALLOCATE(idxarrstr)
!------------------------------------------------------------

!------------------------------------------------------------
! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
! stored in ascending order
ALLOCATE(idx0(besttrackdata(ifile)%numRec))
ALLOCATE(idx1(besttrackdata(ifile)%numRec))

CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)

IF (status /= 0) THEN
  CALL unsetmessagesource()

  CALL terminate()
END IF

! Create the index array to be used in the comparison below
idx0 = arth(1, 1, besttrackdata(ifile)%numRec)

IF (.NOT. arrayequal(idx0, idx1)) THEN
  besttrackdata(ifile)%basin     =  besttrackdata(ifile)%basin(idx1)
  besttrackdata(ifile)%cyNum     =  besttrackdata(ifile)%cyNum(idx1)
  besttrackdata(ifile)%dtg       =  besttrackdata(ifile)%dtg(idx1)
  besttrackdata(ifile)%techNum   =  besttrackdata(ifile)%techNum(idx1)
  besttrackdata(ifile)%tech      =  besttrackdata(ifile)%tech(idx1)
  besttrackdata(ifile)%tau       =  besttrackdata(ifile)%tau(idx1)
  besttrackdata(ifile)%intLat    =  besttrackdata(ifile)%intLat(idx1)
  besttrackdata(ifile)%ns        =  besttrackdata(ifile)%ns(idx1)
  besttrackdata(ifile)%intLon    =  besttrackdata(ifile)%intLon(idx1)
  besttrackdata(ifile)%ew        =  besttrackdata(ifile)%ew(idx1)
  besttrackdata(ifile)%intVMax   =  besttrackdata(ifile)%intVMax(idx1)
  besttrackdata(ifile)%intMslp   =  besttrackdata(ifile)%intMslp(idx1)
  besttrackdata(ifile)%ty        =  besttrackdata(ifile)%ty(idx1)
  besttrackdata(ifile)%rad       =  besttrackdata(ifile)%rad(idx1)
  besttrackdata(ifile)%windCode  =  besttrackdata(ifile)%windCode(idx1)
  besttrackdata(ifile)%intRad1   =  besttrackdata(ifile)%intRad1(idx1)
  besttrackdata(ifile)%intRad2   =  besttrackdata(ifile)%intRad2(idx1)
  besttrackdata(ifile)%intRad3   =  besttrackdata(ifile)%intRad3(idx1)
  besttrackdata(ifile)%intRad4   =  besttrackdata(ifile)%intRad4(idx1)
  besttrackdata(ifile)%intPOuter =  besttrackdata(ifile)%intPOuter(idx1)
  besttrackdata(ifile)%intROuter =  besttrackdata(ifile)%intROuter(idx1)
  besttrackdata(ifile)%intRmw    =  besttrackdata(ifile)%intRmw(idx1)
```

```
      besttrackdata(ifile)%gusts     =  besttrackdata(ifile)%gusts(idx1)
      besttrackdata(ifile)%eye       =  besttrackdata(ifile)%eye(idx1)
      besttrackdata(ifile)%subregion =  besttrackdata(ifile)%subregion(idx1)
      besttrackdata(ifile)%maxseas   =  besttrackdata(ifile)%maxseas(idx1)
      besttrackdata(ifile)%initials  =  besttrackdata(ifile)%initials(idx1)
      besttrackdata(ifile)%dir       =  besttrackdata(ifile)%dir(idx1)
      besttrackdata(ifile)%intSpeed  =  besttrackdata(ifile)%intSpeed(idx1)
      besttrackdata(ifile)%stormName =  besttrackdata(ifile)%stormName(idx1)
      besttrackdata(ifile)%cycleNum  =  besttrackdata(ifile)%cycleNum(idx1)
    END IF

    DEALLOCATE(idx0)
    DEALLOCATE(idx1)
    !------------------------------------------------------------

    CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
  END DO ! End of "iFile" loop

  CALL unsetmessagesource()

END SUBROUTINE readbesttrackfile

SUBROUTINE readcsvbesttrackfile()

  USE pahm_global, ONLY : nbtrfiles, besttrackfilename
  USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique, &
                        intvalstr
  USE sortutils, ONLY : arth, indexx, arrayequal
  USE csv_module

  IMPLICIT NONE

  TYPE(csv_file)                 :: f
  CHARACTER(LEN=64), ALLOCATABLE :: sval2D(:, :)
  LOGICAL                        :: statusOK

  CHARACTER(LEN=FNAMELEN)        :: inpFile
  CHARACTER(LEN=512)             :: line
  CHARACTER(LEN=64)              :: tmpStr

  INTEGER                        :: iFile, nLines, lenLine
  INTEGER                        :: iCnt, jCnt, kCnt, kMax       ! loop counters
  INTEGER                        :: ios, status

  CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
  INTEGER, ALLOCATABLE           :: idxArrStr(:)
  INTEGER                        :: nUnique, maxCnt

  INTEGER, ALLOCATABLE           :: idx0(:), idx1(:)


  CALL setmessagesource("ReadCsvBestTrackFile")

  ! Allocate the best track structure array. This structure holds all the
  ! input values for the storm track as read in from the track input file
  ! (a-deck, b-deck ATCF format) as well as the converted best track variables
  ! (as appropriate).
  ALLOCATE(besttrackdata(nbtrfiles))

  ! This is the main loop. We loop through all the best track files
  ! (user input)
  DO ifile = 1, nbtrfiles
    inpfile = besttrackfilename(ifile)

    besttrackdata(ifile)%fileName  = trim(adjustl(inpfile))
    besttrackdata(ifile)%thisStorm = ""
    besttrackdata(ifile)%loaded    = .false.
    besttrackdata(ifile)%numRec    = -1

    CALL f%Read(trim(adjustl(inpfile)), status_ok=statusok)
    CALL f%Get(sval2d, status_ok=statusok)

    ! Array allocation in the structure bestTrackData
    nlines = f%n_rows
    CALL allocbtrstruct(besttrackdata(ifile), nlines)

    DO icnt = 1, nlines
      DO jcnt = 1 , f%n_cols
        line = line // trim(adjustl(sval2d(icnt, jcnt)))
      END DO
      jcnt = 0
```

```
      lenline = len_trim(adjustl(line))

      IF (lenline /= 0) THEN
        !--- col:  1
        besttrackdata(ifile)%basin(icnt)      = trim(adjustl(sval2d(icnt, 1)))
        !--- col:  2
        besttrackdata(ifile)%cyNum(icnt)      = intvalstr(trim(adjustl(sval2d(icnt, 2))))
        !--- col:  3
        besttrackdata(ifile)%dtg(icnt)        = trim(adjustl(sval2d(icnt, 3)))
        !--- col:  4
        besttrackdata(ifile)%techNum(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 4))))
        !--- col:  5
        besttrackdata(ifile)%tech(icnt)       = trim(adjustl(sval2d(icnt, 5)))
        !--- col:  6
        besttrackdata(ifile)%tau(icnt)        = intvalstr(trim(adjustl(sval2d(icnt, 6))))
        !--- col:  7
        tmpstr = trim(adjustl(sval2d(icnt, 7)))
        READ(tmpstr, '(i3, a1)') &
             besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
        !--- col:  8
        tmpstr = trim(adjustl(sval2d(icnt, 8)))
        READ(tmpstr, '(i3, a1)') &
             besttrackdata(ifile)%intLon(icnt), besttrackdata(ifile)%ew(icnt)
        !--- col:  9
        besttrackdata(ifile)%intVMax(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 9))))
        !--- col: 10
        besttrackdata(ifile)%intMslp(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 10))))
        !--- col: 11
        besttrackdata(ifile)%ty(icnt)         = trim(adjustl(sval2d(icnt, 11)))
        !--- col: 12
        besttrackdata(ifile)%rad(icnt)        = intvalstr(trim(adjustl(sval2d(icnt, 12))))
        !--- col: 13
        besttrackdata(ifile)%windCode(icnt)   = trim(adjustl(sval2d(icnt, 13)))
        !--- col: 14
        besttrackdata(ifile)%intRad1(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 14))))
        !--- col: 15
        besttrackdata(ifile)%intRad2(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 15))))
        !--- col: 16
        besttrackdata(ifile)%intRad3(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 16))))
        !--- col: 17
        besttrackdata(ifile)%intRad4(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 17))))
        !--- col: 18
        besttrackdata(ifile)%intPOuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 18))))
        !--- col: 19
        besttrackdata(ifile)%intROuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 19))))
        !--- col: 20
        besttrackdata(ifile)%intRmw(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 20))))
        !--- col: 21
        besttrackdata(ifile)%gusts(icnt)      = intvalstr(trim(adjustl(sval2d(icnt, 21))))
        !--- col: 22
        besttrackdata(ifile)%eye(icnt)        = intvalstr(trim(adjustl(sval2d(icnt, 22))))
        !--- col: 23
        besttrackdata(ifile)%subregion(icnt) = trim(adjustl(sval2d(icnt, 23)))
        !--- col: 24
        besttrackdata(ifile)%maxseas(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 24))))
        !--- col: 25
         besttrackdata(ifile)%initials(icnt) = trim(adjustl(sval2d(icnt, 25)))
        !--- col: 26
        besttrackdata(ifile)%dir(icnt)        = intvalstr(trim(adjustl(sval2d(icnt, 26))))
        !--- col: 27
        besttrackdata(ifile)%intSpeed(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 27))))
        !--- col: 28
        besttrackdata(ifile)%stormName(icnt) = trim(adjustl(sval2d(icnt, 28)))

        ! This is for the cycleNum, the last column we consider
        IF (icnt == 1) THEN
          kcnt = icnt
          besttrackdata(ifile)%cycleNum(icnt) = icnt
        ELSE
          kcnt = kcnt + 1
          IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
            besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
            kcnt = kcnt - 1
          ELSE
            besttrackdata(ifile)%cycleNum(icnt) = kcnt
          END IF
        END IF

        !---------- Convert lat/lon values to S/N and W/E notations
        IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
```

```
            besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
          ELSE
            besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
          END IF

          IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
            besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
          ELSE
            besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
          END IF
          !----------

          !---------- Get the year, month, day, hour from the DGT string
          READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios) besttrackdata(ifile)%year(icnt)
            IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%month(icnt)
            IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%day(icnt)
            IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%hour(icnt)
            IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
          !----------

        END IF
      END DO

      besttrackdata(ifile)%thisStorm = ''
      besttrackdata(ifile)%loaded    = .true.
      besttrackdata(ifile)%numRec    = nlines

      !------------------------------------------------------------
      ! Get the unique storm name and store it in the thisStorm string
      ALLOCATE(chkarrstr(nlines))
      ALLOCATE(idxarrstr(nlines))

      nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)

      maxcnt = -1
      DO kcnt = 1, nunique
        kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
        IF (kmax > maxcnt) THEN
          maxcnt = kmax
          besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
        END IF
      END DO

      DEALLOCATE(chkarrstr)
      DEALLOCATE(idxarrstr)
      !------------------------------------------------------------

      !------------------------------------------------------------
      ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
      ! stored in ascending order
      ALLOCATE(idx0(besttrackdata(ifile)%numRec))
      ALLOCATE(idx1(besttrackdata(ifile)%numRec))

      CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)

      IF (status /= 0) THEN
        CALL unsetmessagesource()

        CALL terminate()
      END IF

      ! Create the index array to be used in the comparison below
      idx0 = arth(1, 1, besttrackdata(ifile)%numRec)

      IF (.NOT. arrayequal(idx0, idx1)) THEN
        besttrackdata(ifile)%basin     =  besttrackdata(ifile)%basin(idx1)
        besttrackdata(ifile)%cyNum     =  besttrackdata(ifile)%cyNum(idx1)
        besttrackdata(ifile)%dtg       =  besttrackdata(ifile)%dtg(idx1)
        besttrackdata(ifile)%techNum   =  besttrackdata(ifile)%techNum(idx1)
        besttrackdata(ifile)%tech      =  besttrackdata(ifile)%tech(idx1)
        besttrackdata(ifile)%tau       =  besttrackdata(ifile)%tau(idx1)
        besttrackdata(ifile)%intLat    =  besttrackdata(ifile)%intLat(idx1)
        besttrackdata(ifile)%ns        =  besttrackdata(ifile)%ns(idx1)
        besttrackdata(ifile)%intLon    =  besttrackdata(ifile)%intLon(idx1)
```

```
      besttrackdata(ifile)%ew         =  besttrackdata(ifile)%ew(idx1)
      besttrackdata(ifile)%intVMax    =  besttrackdata(ifile)%intVMax(idx1)
      besttrackdata(ifile)%intMslp    =  besttrackdata(ifile)%intMslp(idx1)
      besttrackdata(ifile)%ty         =  besttrackdata(ifile)%ty(idx1)
      besttrackdata(ifile)%rad        =  besttrackdata(ifile)%rad(idx1)
      besttrackdata(ifile)%windCode   =  besttrackdata(ifile)%windCode(idx1)
      besttrackdata(ifile)%intRad1    =  besttrackdata(ifile)%intRad1(idx1)
      besttrackdata(ifile)%intRad2    =  besttrackdata(ifile)%intRad2(idx1)
      besttrackdata(ifile)%intRad3    =  besttrackdata(ifile)%intRad3(idx1)
      besttrackdata(ifile)%intRad4    =  besttrackdata(ifile)%intRad4(idx1)
      besttrackdata(ifile)%intPOuter  =  besttrackdata(ifile)%intPOuter(idx1)
      besttrackdata(ifile)%intROuter  =  besttrackdata(ifile)%intROuter(idx1)
      besttrackdata(ifile)%intRmw     =  besttrackdata(ifile)%intRmw(idx1)
      besttrackdata(ifile)%gusts      =  besttrackdata(ifile)%gusts(idx1)
      besttrackdata(ifile)%eye        =  besttrackdata(ifile)%eye(idx1)
      besttrackdata(ifile)%subregion  =  besttrackdata(ifile)%subregion(idx1)
      besttrackdata(ifile)%maxseas    =  besttrackdata(ifile)%maxseas(idx1)
      besttrackdata(ifile)%initials   =  besttrackdata(ifile)%initials(idx1)
      besttrackdata(ifile)%dir        =  besttrackdata(ifile)%dir(idx1)
      besttrackdata(ifile)%intSpeed   =  besttrackdata(ifile)%intSpeed(idx1)
      besttrackdata(ifile)%stormName  =  besttrackdata(ifile)%stormName(idx1)
      besttrackdata(ifile)%cycleNum   =  besttrackdata(ifile)%cycleNum(idx1)
    END IF

    DEALLOCATE(idx0)
    DEALLOCATE(idx1)
    !------------------------------------------------------------

    CALL f%Destroy()

    CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')

  END DO ! End of "iFile" loop

  CALL unsetmessagesource()

END SUBROUTINE readcsvbesttrackfile

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   P R O C E S S   H O L L A N D   D A T A
!----------------------------------------------------------------
!  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
!----------------------------------------------------------------
SUBROUTINE processhollanddata(idTrFile, strOut, status)

  USE pahm_global, ONLY : nm2m, kt2ms, nbtrfiles
  USE utilities, ONLY : touppercase, charunique
  USE timedateutils, ONLY : timeconv
  USE pahm_vortex, ONLY : calcintensitychange, uvtrans

  IMPLICIT NONE

  INTEGER, INTENT(IN)                :: idTrFile
  TYPE(hollanddata_t), INTENT(OUT)   :: strOut
  INTEGER, INTENT(OUT)               :: status ! error status

  ! numUniqRec, outDTG, idxDTG are used to identify the unique DTG elements in the input structure
  INTEGER                            :: numUniqRec
  CHARACTER(LEN=10), ALLOCATABLE     :: outDTG(:)
  INTEGER, ALLOCATABLE               :: idxDTG(:)

  INTEGER                            :: plIdx            ! populated index for Holland Data array
  INTEGER                            :: iCnt             ! loop counters

  CHARACTER(LEN=4)                   :: castType         !hindcast,forecast
  REAL(SZ), ALLOCATABLE              :: castTime(:)      ! seconds since start of year

  REAL(SZ)                           :: spdVal, pressVal, rrpVal, rmwVal

  status = 0  ! no error


  CALL setmessagesource("ProcessHollandData")

  IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
    IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
      status = 2

      WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ',
```

```
idtrfile
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      RETURN
    END IF
  ELSE
    status = 1

    WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
                                             ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ',
nbtrfiles
    CALL allmessage(error, scratchmessage)

    CALL unsetmessagesource()

    RETURN
  END IF

  WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
  CALL logmessage(info, scratchmessage)

  ! Most likely the array size will be larger if repeated times are found
  ! in the best track structure.
  ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
  ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))

  ! Get unique lines that represent new points in time.
  ! Repeated time points occur in hindcasts for the purpose of
  ! describing winds in the quadrants of the storm. We don't use the
  ! quadrant-by-quadrant wind data. Repeated time data occur in the
  ! forecast because the time data is just the time that the forecast
  ! was made. The important parameter in the forecast file is the
  ! forecast increment.
  numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)

  !--------------------
  ! Populate the Holland structure
  !--------------------
  CALL allocholstruct(strout, numuniqrec)

  ALLOCATE(casttime(numuniqrec))

  strout%fileName  = besttrackdata(idtrfile)%fileName
  strout%thisStorm = besttrackdata(idtrfile)%thisStorm
  strout%loaded    = .true.
  strout%numRec    = numuniqrec

  WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
  CALL logmessage(info, scratchmessage)

  DO icnt = 1, numuniqrec
    plidx = idxdtg(icnt)

    casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(plidx))))

    ! Convert speeds from knots to m/s
    spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)

    ! Convert pressure(s) from mbar to Pa
    pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)

    ! Convert all distances from nm to km/m
    rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
    rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx)    ! in m

    strout%basin(icnt)       = besttrackdata(idtrfile)%basin(plidx)
    strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)
    strout%dtg(icnt)         = besttrackdata(idtrfile)%dtg(plidx)
    strout%year(icnt)        = besttrackdata(idtrfile)%year(plidx)
    strout%month(icnt)       = besttrackdata(idtrfile)%month(plidx)
    strout%day(icnt)         = besttrackdata(idtrfile)%day(plidx)
    strout%hour(icnt)        = besttrackdata(idtrfile)%hour(plidx)
    strout%castType(icnt)    = besttrackdata(idtrfile)%tech(plidx)
    strout%fcstInc(icnt)     = besttrackdata(idtrfile)%tau(plidx)
    strout%iLat(icnt)        = besttrackdata(idtrfile)%intLat(plidx)
    strout%lat(icnt)         = besttrackdata(idtrfile)%lat(plidx)
    strout%iLon(icnt)        = besttrackdata(idtrfile)%intLon(plidx)
    strout%lon(icnt)         = besttrackdata(idtrfile)%lon(plidx)

```

```
      strout%iSpeed(icnt)       = besttrackdata(idtrfile)%intVMax(plidx)
      strout%speed(icnt)        = spdval
      strout%iCPress(icnt)      = besttrackdata(idtrfile)%intMslp(plidx)
      strout%cPress(icnt)       = pressval
      strout%iRrp(icnt)         = besttrackdata(idtrfile)%intROuter(plidx)
      strout%rrp(icnt)          = rrpval
      strout%iRmw(icnt)         = besttrackdata(idtrfile)%intRmw(plidx)
      strout%rmw(icnt)          = rmwval

      ! PV check if this SELECT code is actually needed. Need to check the different format
      ! of input files.
      SELECT CASE(casttype)
        CASE("BEST")      ! nowcast/hindcast
          ! PV check if this is needed
          CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0, 0.0_sz, casttime(icnt))

        CASE("OFCL")      ! forecast
          ! PV check if this is needed
          IF (icnt > 1) THEN
            IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1))) cycle
          END IF

          IF (strout%fcstInc(icnt) == 0) THEN
            CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
                          strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
          ELSE
           casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) * 3600.0_sz)
          END IF

          IF ((strout%iCPress(icnt) == 0) .OR. (strout%iRmw(icnt) == 0)) THEN
            CALL allmessage(error,                                                         &
                            'The storm hindcast/forecast input file ' // trim(strout%fileName) //  &
                            ' contains invalid data for central pressure or rMax.')
            CALL terminate()
          END IF

        ! Adding a new type to allow the analyst to add lines
        ! that do nothing but produce zero winds and background barometric
        ! pressure. These lines can have a date/time like a BEST line or
        ! a date/time and forecast period like an OFCL line.
        CASE("CALM")
          ! PV check if this is needed
          WRITE(scratchmessage, '(a)') 'The file: ' // trim(strout%fileName) // ' contains at least one "CALM" line.'
          CALL logmessage(echo, scratchmessage)

          IF (icnt > 1) THEN
            IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1))) cycle
          END IF

          IF (strout%fcstInc(icnt) == 0) THEN
            CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
                          strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
          ELSE
            casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) * 3600.0_sz)
          END IF

        CASE DEFAULT        ! unrecognized
          WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of ' // &
                                       trim(adjustl(strout%fileName))
          CALL allmessage(error, scratchmessage)

          CALL terminate()
      END SELECT

      strout%castTime(icnt) = casttime(icnt)
    END DO   ! numUniqRec

    ! Calculate the cPress intensity change (dP/dt)
    CALL calcintensitychange(strout%cPress, casttime, strout%cPrDt, status, 2)

    ! Calculate storm translation velocities based on change in position,
    ! approximate u and v translation velocities
    CALL uvtrans(strout%lat, strout%lon, casttime, strout%trVx, strout%trVy, status, 2)

```

```
   DEALLOCATE(casttime)
!--------------------

   DEALLOCATE(outdtg)
   DEALLOCATE(idxdtg)

   CALL unsetmessagesource()

 END SUBROUTINE processhollanddata

!================================================================================

 !----------------------------------------------------------------
 ! S U B R O U T I N E   G E T  H O L L A N D  F I E L D S
 !----------------------------------------------------------------
 !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
 !----------------------------------------------------------------
 SUBROUTINE gethollandfields()

   USE pahm_mesh, ONLY : slam, sfea, xcslam, ycsfea, np, ismeshok
   USE pahm_global, ONLY : gravity, rhowater, rhoair,                        &
                           backgroundatmpress, bladjustfac, one2ten,      &
                           deg2rad, rad2deg, basee, omega, mb2pa, mb2kpa, &
                           nbtrfiles, besttrackfilename,                  &
                           noutdt, mdbegsimtime, mdendsimtime, mdoutdt,   &
                           wvelx, wvely, wpress, times
   USE utilities, ONLY : sphericaldistance, sphericalfracpoint, getlocandratio
   USE timedateutils, ONLY : juldaytogreg, gregtojulday
   USE pahm_netcdfio

   IMPLICIT NONE

   TYPE(hollanddata_t), ALLOCATABLE     :: holStru(:)          ! array of Holland data structures
   INTEGER                              :: stormNumber         ! storm identification number
   REAL(SZ)                             :: hlB                 ! Holland B parameter
   REAL(SZ)                             :: rrp                 ! radius of the last closed isobar (m)
   REAL(SZ)                             :: rmw                 ! radius of max winds (m)
   REAL(SZ)                             :: speed               ! maximum sustained wind speed (m/s)
   REAL(SZ)                             :: cPress              ! central pressure (Pa)
   REAL(SZ)                             :: cPressDef           ! pressure deficit: Ambient Press - cPress
 (Pa)
   REAL(SZ)                             :: trVX, trVY, trSPD   ! storm translation velocities (m/s)
   REAL(SZ)                             :: trSpdX, trSpdY      ! adjusted translation velocities (m/s)
   REAL(SZ)                             :: lon, lat            ! current eye location

   REAL(SZ), ALLOCATABLE                :: rad(:)              ! distance of nodal points from the eye
 location
   INTEGER, ALLOCATABLE                 :: radIDX(:)           ! indices of nodal points duch that rad <=
 rrp
   INTEGER                              :: maxRadIDX           ! total number of radIDX elements
   REAL(SZ)                             :: windMultiplier      ! for storm 2 in lpfs ensemble DO WE NEED
 THIS?
   REAL(SZ)                             :: dx, dy, theta
   REAL(SZ)                             :: wtRatio
   REAL(SZ)                             :: coriolis

   REAL(SZ)                             :: sfPress             ! calculated surface MSL pressure (Pa)
   REAL(SZ)                             :: grVel               ! wind speed (m/s) at gradient level (top
 of ABL)
   REAL(SZ)                             :: sfVelX, sfVelY      ! calculated surface (10-m above ground)
 wind velocities (m/s)

   INTEGER                              :: iCnt, stCnt, npCnt
   INTEGER                              :: i, jl1, jl2
   INTEGER                              :: status

   CHARACTER(LEN=64)                    :: tmpTimeStr, tmpStr1, tmpStr2


   CALL setmessagesource("GetHollandFields")

   ! Check if the mash variables are set and that nOutDT is greater than zero.
   IF (.NOT. ismeshok) THEN
     WRITE(scratchmessage, '(a)') 'The mesh variables are not established properly. ' // &
                                  'Call subroutine ReadMesh to read/create the mesh topology first.'
     CALL allmessage(error, scratchmessage)

     CALL unsetmessagesource()

     CALL terminate()
   ELSE
```

```
     IF ((np <= 0) .OR. (noutdt <= 0)) THEN
       WRITE(tmpstr1, '(a, i0)') 'np = ', np
       WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
       WRITE(scratchmessage, '(a)') 'Variables "np" or "nOutDT" are not defined properly: ' // &
                                    trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
       CALL allmessage(error, scratchmessage)

       CALL unsetmessagesource()

       CALL terminate()
     END IF
   END IF

   !------------------------------
   ! Allocate storage for required arrays and set the Times array
   ! that contains the output times.
   ALLOCATE(times(noutdt))
   ALLOCATE(wvelx(np, noutdt), wvely(np, noutdt), wpress(np, noutdt))

   !wVelX = RMISSV
   !wVelY = wVelX; wPress = wVelX
   wvelx  = 0.0_sz
   wvely  = wvelx
   wpress = backgroundatmpress * mb2pa
   DO icnt = 1, noutdt
     times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
   END DO
   !------------------------------

   !------------------------------
   ! ALLOCATE THE HOLLAND DATA STRUCTURES AND STORE THE HOLLAND
   ! DATA INTO THE DATA STRUCTURE ARRAY FOR SUBSEQUENT USE
   !------------------------------
   !
   ! Allocate the array of Holland data structures. The Holland
   ! structures are allocated by calling the ProcessHollandData
   ! subroutine.
   ALLOCATE(holstru(nbtrfiles))

   ! Process and store the "best track" data into the array of Holland structures
   ! for subsequent use. All required data to generate the P-W model wind fields
   ! are contained in these structures. We take into consideration that might be
   ! more than one "best track" file for the simulation period.
   DO stcnt = 1, nbtrfiles
     CALL processhollanddata(stcnt, holstru(stcnt), status)

     IF (.NOT. holstru(stcnt)%loaded) THEN
       WRITE(scratchmessage, '(a)') 'There was an error loading the Holland data structure for the best track file: ' // &
                                    trim(adjustl(besttrackfilename(stcnt)))
       CALL allmessage(error, scratchmessage)

       CALL deallochollstruct(holstru(stcnt))
       DEALLOCATE(holstru)

       CALL unsetmessagesource()

       CALL terminate()
     ELSE IF (status /= 0) THEN
       WRITE(scratchmessage, '(a)') 'There was an error processing the Holland data structure for the best track file: ' // &
                                    trim(adjustl(besttrackfilename(stcnt)))
       CALL allmessage(error, scratchmessage)

       CALL deallochollstruct(holstru(stcnt))
       DEALLOCATE(holstru)

       CALL unsetmessagesource()

       CALL terminate()
     ELSE
       WRITE(scratchmessage, '(a)') 'Processing the Holland data structure for the best track file: ' // &
                                    trim(adjustl(besttrackfilename(stcnt)))
       CALL logmessage(info, scratchmessage)
     END IF
   END DO
   !------------------------------

   !------------------------------
   ! THIS IS THE MAIN TIME LOOP
```

```
    !------------------------------
    WRITE(scratchmessage, '(a)') 'Start of the main time loop'
    CALL allmessage(info, scratchmessage)
    DO icnt = 1, noutdt
        WRITE(tmpstr1, '(i5)') icnt
        WRITE(tmpstr2, '(i5)') noutdt
      tmpstr1 = '(' // trim(tmpstr1) // '/' // trim(adjustl(tmpstr2)) // ')'
        WRITE(tmptimestr, '(f20.3)') times(icnt)
      WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(tmpstr1)) // " " // trim(adjustl(tmptimestr))
      CALL allmessage(scratchmessage)

      DO stcnt = 1, nbtrfiles
        ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
        ! factor for the subsequent linear interpolation in time. In order for this to
        ! work, the array holStru%castTime should be ordered in ascending order.
        CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, jl1, jl2, wtratio)

        ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
        ! by exiting this loop
        IF ((jl1 <= 0) .OR. (jl2 <= 0)) THEN
          WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(tmptimestr)) // &
                                       ', skipping generating data for this time'
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        ! Perform linear interpolation in time
        stormnumber = holstru(stcnt)%stormNumber(jl1)

        CALL sphericalfracpoint(holstru(stcnt)%lat(jl1), holstru(stcnt)%lon(jl1), &
                                holstru(stcnt)%lat(jl2), holstru(stcnt)%lon(jl2), &
                                wtratio, lat, lon)
        !lat    = holStru(stCnt)%lat(jl1) + &
        !         wtRatio * (holStru(stCnt)%lat(jl2) - holStru(stCnt)%lat(jl1))
        !lon    = holStru(stCnt)%lon(jl1) + &
        !         wtRatio * (holStru(stCnt)%lon(jl2) - holStru(stCnt)%lon(jl1))

        ! Radius of the last closed isobar
        rrp = holstru(stcnt)%rrp(jl1) + &
                wtratio * (holstru(stcnt)%rrp(jl2) - holstru(stcnt)%rrp(jl1))

        ! Radius of maximum winds
        rmw = holstru(stcnt)%rmw(jl1) + &
                wtratio * (holstru(stcnt)%rmw(jl2) - holstru(stcnt)%rmw(jl1))

        ! Get all the distances of the mesh nodes from (lat, lon)
        rad    = sphericaldistance(sfea, slam, lat, lon)
        ! ... and the indices of the nodal points where rad <= rrp
        radidx = pack([(i, i = 1, np)], rad <= rrp)
        maxradidx = SIZE(radidx)

        ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
        IF (maxradidx == 0) THEN
          WRITE(tmpstr1, '(f20.3)') rrp
            tmpstr1 = '(rrp = ' // trim(adjustl(tmpstr1)) // ' m)'
          WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar ' // &
                                       trim(adjustl(tmpstr1)) // ' for storm: ' // &
                                       trim(adjustl(holstru(stcnt)%thisStorm))
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        speed  = holstru(stcnt)%speed(jl1) + &
                  wtratio * (holstru(stcnt)%speed(jl2) - holstru(stcnt)%speed(jl1))

        cpress = holstru(stcnt)%cPress(jl1) + &
                  wtratio * (holstru(stcnt)%cPress(jl2) - holstru(stcnt)%cPress(jl1))

        trvx   = holstru(stcnt)%trVx(jl1) + &
                wtratio * (holstru(stcnt)%trVx(jl2) - holstru(stcnt)%trVx(jl1))
        trvy   = holstru(stcnt)%trVy(jl1) + &
                wtratio * (holstru(stcnt)%trVy(jl2) - holstru(stcnt)%trVy(jl1))

        ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
        ! background pressure and return. PV: check if this is actually needed
        IF (cpress < 0.0_sz) THEN
          wpress(:, icnt) = backgroundatmpress * mb2pa
```

```
          wvelx(:, icnt)  = 0.0_sz
          wvely(:, icnt)  = 0.0_sz

          WRITE(scratchmessage, '(a)') 'Calm period found, generating zero atmospheric fields for this
 time'
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        ! Calculate and limit central pressure deficit; some track files (e.g., Charley 2004)
        ! may have a central pressure greater than the ambient pressure that this subroutine assumes
        cpressdef = backgroundatmpress * mb2pa - cpress
        IF (cpressdef < 100.0_sz) cpressdef = 100.0_sz

        ! Subtract the translational speed of the storm from the observed max wind speed to avoid
        ! distortion in the Holland curve fit. The translational speed will be added back later.
        trspd = sqrt(trvx * trvx + trvy * trvy)
        speed = speed - trspd

        ! Convert wind speed from 10 meter altitude (which is what the
        ! NHC forecast contains) to wind speed at the top of the atmospheric
        ! boundary layer (which is what the Holland curve fit requires).
        speed = speed / bladjustfac

        ! Calculate Holland parameters and limit the result to its appropriate range.
        hlb = rhoair * basee * (speed**2) / cpressdef
        IF (hlb < 1.0_sz) hlb = 1.0_sz
        IF (hlb > 2.5_sz) hlb = 2.5_sz

        ! If we are running storm 2 in the Lake Pontchartrain !PV Do we need this?
        ! Forecast System ensemble, the final wind speeds should be multiplied by 1.2.
        windmultiplier = 1.0_sz
        IF (stormnumber == 2) windmultiplier = 1.2_sz

        DO npcnt = 1, maxradidx
          i = radidx(npcnt)

          dx    = sphericaldistance(lat, lon, lat, slam(i))
          dy    = sphericaldistance(lat, lon, sfea(i), lon)
          theta = atan2(dy, dx)

          ! Compute coriolis
          coriolis = 2.0_sz * omega * sin(sfea(i) * deg2rad)

          ! Compute the pressure (Pa) at a distance rad(i); all distances are in meters
          sfpress = cpress + cpressdef * exp(-(rmw / rad(i))**hlb)

          ! Compute wind speed (speed - trSPD) at gradient level (m/s) and at a distance rad(i);
          ! all distances are in meters. Using absolute value for coriolis for Southern Hempisphere
          grvel = sqrt(speed**2 * (rmw / rad(i))**hlb * exp(1.0_sz - (rmw / rad(i))**hlb) +   &
                      (rad(i) * abs(coriolis) / 2.0_sz)**2) -                                &
                  rad(i) * abs(coriolis) / 2.0_sz

          ! Determine translation speed that should be added to final !PV CHECK ON THIS
          ! storm wind speed. This is tapered to zero as the storm wind tapers
          ! to zero toward the eye of the storm and at long distances from the storm.
          trspdx = (abs(grvel) / speed ) * trvx
          trspdy = (abs(grvel) / speed ) * trvy

          ! Apply mutliplier for Storm #2 in LPFS ensemble.
          grvel = grvel * windmultiplier

          ! Find the wind velocity components.
          sfvelx = -grvel * sin(theta)
          sfvely =  grvel * cos(theta)
          !print *, sfVelX, sfVelY
          ! Convert wind velocity from the gradient level (top of atmospheric boundary layer)
          ! which, is what the Holland curve fit produces, to 10-m wind velocity.
          sfvelx = sfvelx * bladjustfac
          sfvely = sfvely * bladjustfac
          !print *, sfVelX, sfVelY
          ! Convert from 1-minute averaged winds to 10-minute averaged winds.
          sfvelx = sfvelx * one2ten
          sfvely = sfvely * one2ten
          !print *, sfVelX, sfVelY
          ! Add back the storm translation speed.
          sfvelx = sfvelx + trspdx
          sfvely = sfvely + trspdy

          !print *, sfVelX, sfVelY, wVelX(i, iCnt), wVelY(i, iCnt)
```

```
        !PV Need to interpolate between storms if this nodal point
        !   is affected by more than on storm
        wpress(i, icnt) = sfpress
        wvelx(i, icnt)  = sfvelx
        wvely(i, icnt)  = sfvely

        !print *, sfVelX, sfVelY, wVelX(i, iCnt), wVelY(i, iCnt)
        !print *, '-------------------------------------'
      END DO ! npCnt = 1, maxRadIDX

    END DO ! stCnt = 1, nBTrFiles
  END DO ! iCnt = 1, nOutDT
  WRITE(scratchmessage, '(a)') 'End of the main time loop'
  CALL allmessage(info, scratchmessage)

  !---------- Deallocate the arrays
  IF (ALLOCATED(rad)) DEALLOCATE(rad)
  IF (ALLOCATED(radidx)) DEALLOCATE(radidx)
  DO icnt = 1, nbtrfiles
    CALL deallochollstruct(holstru(icnt))
  END DO
  DEALLOCATE(holstru)
  !----------

  CALL unsetmessagesource()

END SUBROUTINE gethollandfields

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   W R I T E  B E S T  T R A C K  D A T A
!----------------------------------------------------------------
!  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
!----------------------------------------------------------------
SUBROUTINE writebesttrackdata(inpFile, btrStruc, suffix)

  USE pahm_global, ONLY : lun_btrk, lun_btrk1

  IMPLICIT NONE

  ! Global variables
  CHARACTER(LEN=*)                          :: inpFile
  TYPE(besttrackdata_t), INTENT(IN)         :: btrStruc
  CHARACTER(LEN=*), OPTIONAL, INTENT(IN)    :: suffix

  ! Local variables
  CHARACTER(LEN=FNAMELEN)                   :: outFile
  CHARACTER(LEN=64)                         :: fSuf
  INTEGER                                   :: iCnt
  INTEGER                                   :: iUnit, errIO
  CHARACTER(LEN=512)                        :: fmtStr


  !---------- Initialize variables
  iunit  = lun_btrk1
  errio  = 0

  fmtstr = '(a2, ",", 1x, i2.2, ",", 1x, a10, ",", 1x, i2, ",", 1x, a4, ",", 1x, i3, ",", 1x, i3, a1, ",", 1x, i4, a1, ",", '
    fmtstr = trim(fmtstr) // ' 1x, i3, ",", 1x, i4, ",", 1x, a2, ",", 1x, i3, ",", 1x, a3, ",", '
    fmtstr = trim(fmtstr) // ' 4(1x, i4, ","), 1x, i4, ",", 1x, i4, ",", 1x, i3, ",", 1x, i4, ",", 1x, i3, ",", '
    fmtstr = trim(fmtstr) // ' 1x, a3,",", 1x, i3,",", 1x, a3, ",", i3,",", 1x, i3,",", 1x, a11,",", 1x, i3, ",")'
  !----------

  fsuf = '_adj'
  IF (PRESENT(suffix)) fsuf = adjustl(suffix)

  CALL setmessagesource("WriteBestTrackData")

  IF (.NOT. btrstruc%loaded) THEN
    WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be written."
    CALL allmessage(info, scratchmessage)

    RETURN
  END IF

  outfile = trim(adjustl(inpfile)) // trim(fsuf)
```

```

    WRITE(scratchmessage, '(a)') 'Writting the "adjusted" best track data to: ' // trim(adjustl(outfile))
    CALL logmessage(info, scratchmessage)

    OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)

    IF (errio /= 0) THEN
      WRITE(scratchmessage, '(a)') 'Error opening the outFile: '  // trim(outfile) // &
                                   ', skip writting the "adjusted" best track fields'
      CALL allmessage(error, scratchmessage)

      RETURN
    END IF

    DO icnt = 1, btrstruc%numRec
      WRITE(iunit, fmtstr)                                         &
          btrstruc%basin(icnt),     btrstruc%cyNum(icnt),          &
          btrstruc%dtg(icnt),       btrstruc%techNum(icnt),        &
          btrstruc%tech(icnt),      btrstruc%tau(icnt),            &
          btrstruc%intLat(icnt),    btrstruc%ns(icnt),             &
          btrstruc%intLon(icnt),    btrstruc%ew(icnt),             &
          btrstruc%intVMax(icnt),   btrstruc%intMslp(icnt),        &
          btrstruc%ty(icnt),        btrstruc%rad(icnt),            &
          btrstruc%windCode(icnt),  btrstruc%intRad1(icnt),        &
          btrstruc%intRad2(icnt),   btrstruc%intRad3(icnt),        &
          btrstruc%intRad4(icnt),   btrstruc%intPOuter(icnt),      &
          btrstruc%intROuter(icnt), btrstruc%intRmw(icnt),         &
          btrstruc%gusts(icnt),     btrstruc%eye(icnt),            &
          btrstruc%subregion(icnt), btrstruc%maxseas(icnt),        &
          btrstruc%initials(icnt),  btrstruc%dir(icnt),            &
          btrstruc%intSpeed(icnt),  btrstruc%stormName(icnt),      &
          btrstruc%cycleNum(icnt)
    END DO

    CLOSE(iunit)

    CALL unsetmessagesource()

  END SUBROUTINE writebesttrackdata

!================================================================================

  !-----------------------------------------------------------------
  ! S U B R O U T I N E   A L L O C   B T R   S T R U C T
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  SUBROUTINE allocbtrstruct(str, nRec)

    IMPLICIT NONE

    TYPE(besttrackdata_t) :: str
    INTEGER, INTENT(IN)   :: nRec

    str%numRec = nrec
    str%loaded = .false.

    !----- Input parameters
    IF (.NOT. ALLOCATED(str%basin))      ALLOCATE(str%basin(nrec))
    IF (.NOT. ALLOCATED(str%cyNum))      ALLOCATE(str%cyNum(nrec))
    IF (.NOT. ALLOCATED(str%dtg))        ALLOCATE(str%dtg(nrec))
    IF (.NOT. ALLOCATED(str%techNum))    ALLOCATE(str%techNum(nrec))
    IF (.NOT. ALLOCATED(str%tech))       ALLOCATE(str%tech(nrec))
    IF (.NOT. ALLOCATED(str%tau))        ALLOCATE(str%tau(nrec))
    IF (.NOT. ALLOCATED(str%intLat))     ALLOCATE(str%intLat(nrec))
    IF (.NOT. ALLOCATED(str%intLon))     ALLOCATE(str%intLon(nrec))
    IF (.NOT. ALLOCATED(str%ew))         ALLOCATE(str%ew(nrec))
    IF (.NOT. ALLOCATED(str%ns))         ALLOCATE(str%ns(nrec))
    IF (.NOT. ALLOCATED(str%intVMax))    ALLOCATE(str%intVMax(nrec))
    IF (.NOT. ALLOCATED(str%intMslp))    ALLOCATE(str%intMslp(nrec))
    IF (.NOT. ALLOCATED(str%ty))         ALLOCATE(str%ty(nrec))
    IF (.NOT. ALLOCATED(str%rad))        ALLOCATE(str%rad(nrec))
    IF (.NOT. ALLOCATED(str%windCode))   ALLOCATE(str%windCode(nrec))
    IF (.NOT. ALLOCATED(str%intRad1))    ALLOCATE(str%intRad1(nrec))
    IF (.NOT. ALLOCATED(str%intRad2))    ALLOCATE(str%intRad2(nrec))
    IF (.NOT. ALLOCATED(str%intRad3))    ALLOCATE(str%intRad3(nrec))
    IF (.NOT. ALLOCATED(str%intRad4))    ALLOCATE(str%intRad4(nrec))
    IF (.NOT. ALLOCATED(str%intPOuter))  ALLOCATE(str%intPOuter(nrec))
    IF (.NOT. ALLOCATED(str%intROuter))  ALLOCATE(str%intROuter(nrec))
    IF (.NOT. ALLOCATED(str%intRmw))     ALLOCATE(str%intRmw(nrec))
    IF (.NOT. ALLOCATED(str%gusts))      ALLOCATE(str%gusts(nrec))
    IF (.NOT. ALLOCATED(str%eye))        ALLOCATE(str%eye(nrec))
```

```
    IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
    IF (.NOT. ALLOCATED(str%maxseas))   ALLOCATE(str%maxseas(nrec))
    IF (.NOT. ALLOCATED(str%initials))  ALLOCATE(str%initials(nrec))
    IF (.NOT. ALLOCATED(str%dir))       ALLOCATE(str%dir(nrec))
    IF (.NOT. ALLOCATED(str%intSpeed))  ALLOCATE(str%intSpeed(nrec))
    IF (.NOT. ALLOCATED(str%stormName)) ALLOCATE(str%stormName(nrec))
    IF (.NOT. ALLOCATED(str%cycleNum))  ALLOCATE(str%cycleNum(nrec))

    !----- Converted parameters
    IF (.NOT. ALLOCATED(str%year))      ALLOCATE(str%year(nrec))
    IF (.NOT. ALLOCATED(str%month))     ALLOCATE(str%month(nrec))
    IF (.NOT. ALLOCATED(str%day))       ALLOCATE(str%day(nrec))
    IF (.NOT. ALLOCATED(str%hour))      ALLOCATE(str%hour(nrec))
    IF (.NOT. ALLOCATED(str%lat))       ALLOCATE(str%lat(nrec))
    IF (.NOT. ALLOCATED(str%lon))       ALLOCATE(str%lon(nrec))

  END SUBROUTINE allocbtrstruct

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   D E A L L O C  B T R  S T R U C T
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE deallocbtrstruct(str)

    IMPLICIT NONE

    TYPE(besttrackdata_t) :: str

    str%numRec = -1
    str%loaded = .false.

    !----- Input parameters
    IF (ALLOCATED(str%basin))     DEALLOCATE(str%basin)
    IF (ALLOCATED(str%cyNum))     DEALLOCATE(str%cyNum)
    IF (ALLOCATED(str%dtg))       DEALLOCATE(str%dtg)
    IF (ALLOCATED(str%techNum))   DEALLOCATE(str%techNum)
    IF (ALLOCATED(str%tech))      DEALLOCATE(str%tech)
    IF (ALLOCATED(str%tau))       DEALLOCATE(str%tau)
    IF (ALLOCATED(str%intLat))    DEALLOCATE(str%intLat)
    IF (ALLOCATED(str%intLon))    DEALLOCATE(str%intLon)
    IF (ALLOCATED(str%ew))        DEALLOCATE(str%ew)
    IF (ALLOCATED(str%ns))        DEALLOCATE(str%ns)
    IF (ALLOCATED(str%intVMax))   DEALLOCATE(str%intVMax)
    IF (ALLOCATED(str%intMslp))   DEALLOCATE(str%intMslp)
    IF (ALLOCATED(str%ty))        DEALLOCATE(str%ty)
    IF (ALLOCATED(str%rad))       DEALLOCATE(str%rad)
    IF (ALLOCATED(str%windCode))  DEALLOCATE(str%windCode)
    IF (ALLOCATED(str%intRad1))   DEALLOCATE(str%intRad1)
    IF (ALLOCATED(str%intRad2))   DEALLOCATE(str%intRad2)
    IF (ALLOCATED(str%intRad3))   DEALLOCATE(str%intRad3)
    IF (ALLOCATED(str%intRad4))   DEALLOCATE(str%intRad4)
    IF (ALLOCATED(str%intPOuter)) DEALLOCATE(str%intPOuter)
    IF (ALLOCATED(str%intROuter)) DEALLOCATE(str%intROuter)
    IF (ALLOCATED(str%intRmw))    DEALLOCATE(str%intRmw)
    IF (ALLOCATED(str%gusts))     DEALLOCATE(str%gusts)
    IF (ALLOCATED(str%eye))       DEALLOCATE(str%eye)
    IF (ALLOCATED(str%subregion)) DEALLOCATE(str%subregion)
    IF (ALLOCATED(str%maxseas))   DEALLOCATE(str%maxseas)
    IF (ALLOCATED(str%initials))  DEALLOCATE(str%initials)
    IF (ALLOCATED(str%dir))       DEALLOCATE(str%dir)
    IF (ALLOCATED(str%intSpeed))  DEALLOCATE(str%intSpeed)
    IF (ALLOCATED(str%stormName)) DEALLOCATE(str%stormName)
    IF (ALLOCATED(str%cycleNum))  DEALLOCATE(str%cycleNum)

     !----- Converted parameters
    IF (ALLOCATED(str%year))      DEALLOCATE(str%year)
    IF (ALLOCATED(str%month))     DEALLOCATE(str%month)
    IF (ALLOCATED(str%day))       DEALLOCATE(str%day)
    IF (ALLOCATED(str%hour))      DEALLOCATE(str%hour)
    IF (ALLOCATED(str%lat))       DEALLOCATE(str%lat)
    IF (ALLOCATED(str%lon))       DEALLOCATE(str%lon)

  END SUBROUTINE deallocbtrstruct

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   A L L O C  H O L L  S T R U C T
  !----------------------------------------------------------------
```

```
!----------------------------------------------------------------
SUBROUTINE allochollstruct(str, nRec)

  IMPLICIT NONE

  TYPE(hollanddata_t) :: str
  INTEGER, INTENT(IN) :: nRec

  str%numRec = nrec
  str%loaded = .false.

  !----- Input parameters
  IF (.NOT. ALLOCATED(str%basin))       ALLOCATE(str%basin(nrec))

  IF (.NOT. ALLOCATED(str%dtg))         ALLOCATE(str%dtg(nrec))
  IF (.NOT. ALLOCATED(str%stormNumber)) ALLOCATE(str%stormNumber(nrec))
  IF (.NOT. ALLOCATED(str%year))        ALLOCATE(str%year(nrec))
  IF (.NOT. ALLOCATED(str%month))       ALLOCATE(str%month(nrec))
  IF (.NOT. ALLOCATED(str%day))         ALLOCATE(str%day(nrec))
  IF (.NOT. ALLOCATED(str%hour))        ALLOCATE(str%hour(nrec))

  IF (.NOT. ALLOCATED(str%castTime))    ALLOCATE(str%castTime(nrec))
  IF (.NOT. ALLOCATED(str%castType))    ALLOCATE(str%castType(nrec))
  IF (.NOT. ALLOCATED(str%fcstInc))     ALLOCATE(str%fcstInc(nrec))

  IF (.NOT. ALLOCATED(str%iLat))        ALLOCATE(str%iLat(nrec))
  IF (.NOT. ALLOCATED(str%lat))         ALLOCATE(str%lat(nrec))
  IF (.NOT. ALLOCATED(str%iLon))        ALLOCATE(str%iLon(nrec))
  IF (.NOT. ALLOCATED(str%lon))         ALLOCATE(str%lon(nrec))

  IF (.NOT. ALLOCATED(str%iSpeed))      ALLOCATE(str%iSpeed(nrec))
  IF (.NOT. ALLOCATED(str%speed))       ALLOCATE(str%speed(nrec))

  IF (.NOT. ALLOCATED(str%iCPress))     ALLOCATE(str%iCPress(nrec))
  IF (.NOT. ALLOCATED(str%cPress))      ALLOCATE(str%cPress(nrec))

  IF (.NOT. ALLOCATED(str%iRrp))        ALLOCATE(str%iRrp(nrec))
  IF (.NOT. ALLOCATED(str%rrp))         ALLOCATE(str%rrp(nrec))

  IF (.NOT. ALLOCATED(str%iRmw))        ALLOCATE(str%iRmw(nrec))
  IF (.NOT. ALLOCATED(str%rmw))         ALLOCATE(str%rmw(nrec))

  IF (.NOT. ALLOCATED(str%cPrDt))       ALLOCATE(str%cPrDt(nrec))

  IF (.NOT. ALLOCATED(str%trVx))        ALLOCATE(str%trVx(nrec))
  IF (.NOT. ALLOCATED(str%trVy))        ALLOCATE(str%trVy(nrec))

END SUBROUTINE allochollstruct

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   D E A L L O C   H O L L   S T R U C T
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE deallochollstruct(str)

  IMPLICIT NONE

  TYPE(hollanddata_t), INTENT(OUT) :: str

  str%numRec = -1
  str%loaded = .false.

  !----- Input parameters
  IF (ALLOCATED(str%basin))       DEALLOCATE(str%basin)

  IF (ALLOCATED(str%dtg))         DEALLOCATE(str%dtg)
  IF (ALLOCATED(str%stormNumber)) DEALLOCATE(str%stormNumber)
  IF (ALLOCATED(str%year))        DEALLOCATE(str%year)
  IF (ALLOCATED(str%month))       DEALLOCATE(str%month)
  IF (ALLOCATED(str%day))         DEALLOCATE(str%day)
  IF (ALLOCATED(str%hour))        DEALLOCATE(str%hour)

  IF (ALLOCATED(str%castTime))    DEALLOCATE(str%castTime)
  IF (ALLOCATED(str%castType))    DEALLOCATE(str%castType)
  IF (ALLOCATED(str%fcstInc))     DEALLOCATE(str%fcstInc)

  IF (ALLOCATED(str%iLat))        DEALLOCATE(str%iLat)
  IF (ALLOCATED(str%lat))         DEALLOCATE(str%lat)
  IF (ALLOCATED(str%iLon))        DEALLOCATE(str%iLon)
```

```
    IF (ALLOCATED(str%lon))           DEALLOCATE(str%lon)

    IF (ALLOCATED(str%iSpeed))        DEALLOCATE(str%iSpeed)
    IF (ALLOCATED(str%speed))         DEALLOCATE(str%speed)

    IF (ALLOCATED(str%iCPress))       DEALLOCATE(str%iCPress)
    IF (ALLOCATED(str%cPress))        DEALLOCATE(str%cPress)

    IF (ALLOCATED(str%iRrp))          DEALLOCATE(str%iRrp)
    IF (ALLOCATED(str%rrp))           DEALLOCATE(str%rrp)

    IF (ALLOCATED(str%iRmw))          DEALLOCATE(str%iRmw)
    IF (ALLOCATED(str%rmw))           DEALLOCATE(str%rmw)

    IF (ALLOCATED(str%cPrDt))         DEALLOCATE(str%cPrDt)

    IF (ALLOCATED(str%trVx))          DEALLOCATE(str%trVx)
    IF (ALLOCATED(str%trVy))          DEALLOCATE(str%trVy)

  END SUBROUTINE deallochollstruct

!================================================================================

END MODULE parwind
```

9.30 parwind.F90 File Reference

Data Types

- type parwind::besttrackdata_t
- type parwind::hollanddata_t

Modules

- module parwind

Functions/Subroutines

- subroutine parwind::readbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine parwind::readcsvbesttrackfile ()

  Subroutine to read all a-deck/b-deck best track files (ATCF format).
- subroutine parwind::processhollanddata (idTrFile, strOut, status)

  Subroutine to support the Holland model (GetHolland). Gets the next line from the file, skipping lines that are time repeats.
- subroutine parwind::gethollandfields (timeIDX)

  Calculates wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.
- subroutine parwind::writebesttrackdata (inpFile, btrStruc, suffix)

  Writes the best track data (adjusted or not) to the "adjusted" best track output file.
- subroutine parwind::allocbtrstruct (str, nRec)

  Subroutine to allocate memory for a best track structure.
- subroutine parwind::deallocbtrstruct (str)

  Subroutine to deallocate the memory allocated for a best track structure.
- subroutine parwind::allochollstruct (str, nRec)

  Subroutine to allocate memory for a holland structure.
- subroutine parwind::deallochollstruct (str)

  Subroutine to deallocate memory of an allocated holland structure.

9.30.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file parwind.F90.

9.31 parwind.F90

Go to the documentation of this file.

```
!-----------------------------------------------------------------
!                M O D U L E   P A R W I N D
!-----------------------------------------------------------------
!-----------------------------------------------------------------

MODULE parwind

  USE pahm_sizes
  USE pahm_messages

  ! switch to turn on or off geostrophic balance in GAHM
  ! on (default): Coriolis term included, phiFactors will be calculated before being used
  ! off         : parameter is set to 'TRUE', phiFactors will be set to constant 1
  !LOGICAL :: geostrophicSwitch = .TRUE.
  !INTEGER :: geoFactor = 1               !turn on or off gostrophic balance

  REAL(SZ) :: WindRefTime !jgf46.29 seconds since beginning of year, this          !PV check
                          !corresponds to time=0 of the simulation

  !-----------------------------------------------------------------
  ! The BestTrackData_T structure holds all data read from the best track files(s)
  ! in ATCF format (a-deck/b-deck)
  !-----------------------------------------------------------------
  TYPE besttrackdata_t
    CHARACTER(LEN=FNAMELEN)          :: fileName         ! full path to the best track file
    CHARACTER(LEN=10)                :: thisStorm        ! the name of the "named" storm
    LOGICAL                          :: loaded = .false. ! .TRUE. if we have loaded the data from file
    INTEGER                          :: numRec           ! number of records in the structure

    !----- input data from best track file (ATCF format)
    CHARACTER(LEN=2), ALLOCATABLE    :: basin(:)         ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
    INTEGER, ALLOCATABLE             :: cyNum(:)         ! annual cyclone number: 1 - 99
    CHARACTER(LEN=10), ALLOCATABLE   :: dtg(:)           ! warning Date-Time-Group (DTG), YYYYMMDDHH
    INTEGER, ALLOCATABLE             :: techNum(:)       ! objective technique sorting number, minutes for best track: 00 - 99
    CHARACTER(LEN=4), ALLOCATABLE    :: tech(:)          ! acronym for each objective technique or CARQ or WRNG,
                                                         ! BEST for best track, up to 4 chars.
    INTEGER, ALLOCATABLE             :: tau(:)           ! forecast period: -24 through 240 hours, 0 for best-track,
                                                         ! negative taus used for CARQ and WRNG records.
    INTEGER, ALLOCATABLE             :: intLat(:)        ! latitude for the DTG: 0 - 900 tenths of degrees
    INTEGER, ALLOCATABLE             :: intLon(:)        ! latitude for the DTG: 0 - 900 tenths of degrees
    CHARACTER(LEN=1), ALLOCATABLE    :: ew(:)            ! E/W
    CHARACTER(LEN=1), ALLOCATABLE    :: ns(:)            ! N/S

    INTEGER, ALLOCATABLE             :: intVMax(:)       ! maximum sustained wind speed in knots: 0 - 300 kts
    INTEGER, ALLOCATABLE             :: intMslp(:)       ! minimum sea level pressure, 850 - 1050 mb
    CHARACTER(LEN=2), ALLOCATABLE    :: ty(:)            ! Highest level of tc development:
                                                         !   DB - disturbance,
                                                         !   TD - tropical depression,
                                                         !   TS - tropical storm,
                                                         !   TY - typhoon,
                                                         !   ST - super typhoon,
                                                         !   TC - tropical cyclone,
                                                         !   HU - hurricane,
                                                         !   SD - subtropical depression,
                                                         !   SS - subtropical storm,
                                                         !   EX - extratropical systems,
                                                         !   PT - post tropical,
```

```
                                                          !   IN - inland,
                                                          !   DS - dissipating,
                                                          !   LO - low,
                                                          !   WV - tropical wave,
                                                          !   ET - extrapolated,
                                                          !   MD - monsoon depression,
                                                          !   XX - unknown.
    INTEGER, ALLOCATABLE             :: rad(:)            ! wind intensity for the radii defined in this record: 34, 50 or 64 kt
    CHARACTER(LEN=3), ALLOCATABLE    :: windCode(:)       ! radius code:
                                                          !   AAA - full circle
                                                          !   NEQ, SEQ, SWQ, NWQ - quadrant
    INTEGER, ALLOCATABLE             :: intRad1(:)        ! if full circle, radius of specified wind intensity, or radius of
                                                          ! first quadrant wind intensity as specified by WINDCODE.  0 - 999 n mi
    INTEGER, ALLOCATABLE             :: intRad2(:)        ! if full circle this field not used, or radius of 2nd quadrant wind
                                                          ! intensity as specified by WINDCODE.  0 - 999 n mi
    INTEGER, ALLOCATABLE             :: intRad3(:)        ! if full circle this field not used, or radius of 3rd quadrant wind
                                                          ! intensity as specified by WINDCODE.  0 - 999 n mi
    INTEGER, ALLOCATABLE             :: intRad4(:)        ! if full circle this field not used, or radius of 4th quadrant wind
                                                          ! intensity as specified by WINDCODE.  0 - 999 n mi
    INTEGER, ALLOCATABLE             :: intPOuter(:)      ! pressure in millibars of the last closed isobar, 900 - 1050 mb
    INTEGER, ALLOCATABLE             :: intROuter(:)      ! radius of the last closed isobar, 0 - 999 n mi
    INTEGER, ALLOCATABLE             :: intRmw(:)         ! radius of max winds, 0 - 999 n mi
    INTEGER, ALLOCATABLE             :: gusts(:)          ! gusts, 0 - 999 kt
    INTEGER, ALLOCATABLE             :: eye(:)            ! eye diameter, 0 - 120 n mi
    CHARACTER(LEN=3), ALLOCATABLE    :: subregion(:)      ! subregion code: W,A,B,S,P,C,E,L,Q
                                                          !   A - Arabian Sea
                                                          !   B - Bay of Bengal
                                                          !   C - Central Pacific
                                                          !   E - Eastern Pacific
                                                          !   L - Atlantic
                                                          !   P - South Pacific (135E - 120W)
                                                          !   Q - South Atlantic
                                                          !   S - South IO (20E - 135E)
                                                          !   W - Western Pacific
    INTEGER, ALLOCATABLE             :: maxseas(:)        ! max seas: 0 - 999 ft
    CHARACTER(LEN=3), ALLOCATABLE    :: initials(:)       ! forecaster's initials used for tau 0 WRNG or OFCL, up to 3 chars
    INTEGER, ALLOCATABLE             :: dir(:)            ! storm direction, 0 - 359 degrees
    INTEGER, ALLOCATABLE             :: intSpeed(:)       ! storm speed, 0 - 999 kts
    CHARACTER(LEN=10), ALLOCATABLE   :: stormName(:)      ! literal storm name, number, NONAME or INVEST, or TCcyx where:
                                                          !   cy = Annual cyclone number 01 - 99
                                                          !   x  = Subregion code: W,A,B,S,P,C,E,L,Q.
    INTEGER, ALLOCATABLE             :: cycleNum(:)       ! the cycle number !PV check if this is OK

!    !----- converted data from the above values (if needed)
    INTEGER, DIMENSION(:), ALLOCATABLE  :: year, month, day, hour
    REAL(SZ), DIMENSION(:), ALLOCATABLE :: lat, lon
  END TYPE besttrackdata_t

  ! Array of info about the best track data (extension to use multiple storms)
  TYPE(BestTrackData_T), ALLOCATABLE, TARGET :: bestTrackData(:)

  !----------------------------------------------------------------
  ! The HollandData_T structure holds all required data for the Holland model
  ! The data are filtered to only include unique DTGs
  !----------------------------------------------------------------
  TYPE hollanddata_t
    CHARACTER(LEN=FNAMELEN)              :: fileName         ! full path to the best track file
    CHARACTER(LEN=10)                    :: thisStorm        ! the name of the "named" storm
    LOGICAL                              :: loaded = .false. ! .TRUE. if we have loaded the data from file
    INTEGER                              :: numRec           ! number of records in the structure

    CHARACTER(LEN=2),       ALLOCATABLE :: basin(:)          ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
    INTEGER, ALLOCATABLE                :: stormNumber(:)    ! annual cyclone number: 1 - 99
    CHARACTER(LEN=10),      ALLOCATABLE :: dtg(:)            ! warning Date-Time-Group (DTG), YYYYMMDDHH
    INTEGER, DIMENSION(:),  ALLOCATABLE :: year, month, day, hour
    REAL(SZ), ALLOCATABLE               :: castTime(:)       ! converted to decimal E/N (lon, lat)
    CHARACTER(LEN=4),       ALLOCATABLE :: castType(:)       ! BEST, OFCL, CALM, ...
    INTEGER,                ALLOCATABLE :: fcstInc(:)        ! forecast period: -24 through 240 hours, 0 for best-track
```

```
   INTEGER, DIMENSION(:),  ALLOCATABLE :: iLat, iLon        ! latitude, longitude for the GTD
   REAL(SZ), DIMENSION(:), ALLOCATABLE :: lat, lon          ! converted to decimal E/N (lon, lat)

   INTEGER,                ALLOCATABLE :: iSpeed(:)         ! maximum sustained wind speed in knots: 0 - 300 kts
   REAL(SZ),               ALLOCATABLE :: speed(:)          ! converted from kts to m/s

   INTEGER,                ALLOCATABLE :: iCPress(:)        ! minimum sea level pressure, 850 - 1050 mb
   REAL(SZ),               ALLOCATABLE :: cPress(:)         ! converted to Pa

   INTEGER,                ALLOCATABLE :: iRrp(:)           ! radius of the last closed isobar, 0 - 999 n mi
   REAL(SZ),               ALLOCATABLE :: rrp(:)            ! converted from nm to m

   INTEGER,                ALLOCATABLE :: iRmw(:)           ! radius of max winds, 0 - 999 n mi
   REAL(SZ),               ALLOCATABLE :: rmw(:)            ! converted from nm to m

   REAL(SZ), DIMENSION(:), ALLOCATABLE :: cPrDt             ! central pressure intensity change (Pa / h)
   REAL(SZ), DIMENSION(:), ALLOCATABLE :: trVx, trVy        ! translational velocity components (x, y) of the
                                                            ! moving hurricane (m/s)
 END TYPE hollanddata_t


 CONTAINS


 !----------------------------------------------------------------
 !  S U B R O U T I N E   R E A D  B E S T  T R A C K  F I L E
 !----------------------------------------------------------------
 !----------------------------------------------------------------
 SUBROUTINE readbesttrackfile()

   USE pahm_global, ONLY : lun_btrk, lun_btrk1, nbtrfiles, besttrackfilename
   USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
   USE sortutils, ONLY : arth, indexx, arrayequal

   IMPLICIT NONE

   CHARACTER(LEN=FNAMELEN)          :: inpFile
   CHARACTER(LEN=512)               :: inpLine, line
   CHARACTER(LEN=512)               :: fmtStr

   INTEGER                          :: lenLine
   INTEGER                          :: nLines                 ! Number of lines counter
   INTEGER                          :: iFile, iCnt            ! loop counters
   INTEGER                          :: iUnit, errIO, ios, status

   CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
   INTEGER, ALLOCATABLE             :: idxArrStr(:)
   INTEGER                          :: nUnique, maxCnt, kCnt, kMax

   INTEGER, ALLOCATABLE             :: idx0(:), idx1(:)

   !---------- Initialize variables
   iunit = lun_btrk
   errio = 0

   fmtstr = '(a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x, a2, '
     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i4, 2x, i3, 2x, i4, 2x, i3, '
     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, i3, 2x, i3, 2x, a11, 2x, i3)'
   !----------

   CALL setmessagesource("ReadBestTrackFile")

   ! Allocate the best track structure array. This structure holds all the
   ! input values for the storm track as read in from the track input file
   ! (a-deck, b-deck ATCF format) as well as the converted best track variables
   ! (as appropriate).
   ALLOCATE(besttrackdata(nbtrfiles))

   ! This is the main loop. We loop through all the best track files
   ! (user input)
   DO ifile = 1, nbtrfiles
     inpfile = besttrackfilename(ifile)

     CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)

     IF (errio /= 0) THEN
```

```
        WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
        CALL allmessage(error, scratchmessage)

        CALL unsetmessagesource()

        CALL terminate()
      ELSE
        WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
        CALL logmessage(info, scratchmessage)
      END IF

      besttrackdata(ifile)%fileName  = trim(adjustl(inpfile))
      besttrackdata(ifile)%thisStorm = ""
      besttrackdata(ifile)%loaded    = .false.
      besttrackdata(ifile)%numRec    = -1

      ! Count the number of non-empty or commented out lines in the file.
      ! Comments are are considered those lines with the first non-blank character of "!" or "#"
      nlines = 0
      DO
        READ(unit=iunit, fmt='(a)', err=10, END=5, IOSTAT=errIO) inpline

        lenline = getlinerecord(inpline, line)
        IF (lenline /= 0) nlines = nlines + 1
      END DO
      5 rewind(unit=iunit)

      ! Array allocation in the structure bestTrackData
      CALL allocbtrstruct(besttrackdata(ifile), nlines)

      icnt = 0
      DO WHILE (.true.)
        READ(unit=iunit, fmt='(a)', err=10, END=20, IOSTAT=errIO) inpline

        lenline = getlinerecord(inpline, line)

        IF (lenline /= 0) THEN
          icnt = icnt + 1
          READ(line, fmt=fmtstr, err=11, iostat=ios)                                       &
            besttrackdata(ifile)%basin(icnt),     besttrackdata(ifile)%cyNum(icnt),        &
            besttrackdata(ifile)%dtg(icnt),       besttrackdata(ifile)%techNum(icnt),      &
            besttrackdata(ifile)%tech(icnt),      besttrackdata(ifile)%tau(icnt),          &
            besttrackdata(ifile)%intLat(icnt),    besttrackdata(ifile)%ns(icnt),           &
            besttrackdata(ifile)%intLon(icnt),    besttrackdata(ifile)%ew(icnt),           &
            besttrackdata(ifile)%intVMax(icnt),   besttrackdata(ifile)%intMslp(icnt),      &
            besttrackdata(ifile)%ty(icnt),        besttrackdata(ifile)%rad(icnt),          &
            besttrackdata(ifile)%windCode(icnt),  besttrackdata(ifile)%intRad1(icnt),      &
            besttrackdata(ifile)%intRad2(icnt),   besttrackdata(ifile)%intRad3(icnt),      &
            besttrackdata(ifile)%intRad4(icnt),   besttrackdata(ifile)%intPOuter(icnt),    &
            besttrackdata(ifile)%intROuter(icnt), besttrackdata(ifile)%intRmw(icnt),       &
            besttrackdata(ifile)%gusts(icnt),     besttrackdata(ifile)%eye(icnt),          &
            besttrackdata(ifile)%subregion(icnt), besttrackdata(ifile)%maxseas(icnt),      &
            besttrackdata(ifile)%initials(icnt),  besttrackdata(ifile)%dir(icnt),          &
            besttrackdata(ifile)%intSpeed(icnt),  besttrackdata(ifile)%stormName(icnt),    &
            besttrackdata(ifile)%cycleNum(icnt)

          !---------- Convert lat/lon values to S/N and W/E notations
          IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
            besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
          ELSE
            besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
          END IF

          IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
            besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
          ELSE
            besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
          END IF
          !----------

          !---------- Get the year, month, day, hour from the DGT string
          READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios) besttrackdata(ifile)%year(icnt)
            IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%month(icnt)
            IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%day(icnt)
            IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
          READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
```

```
besttrackdata(ifile)%hour(icnt)
          IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
        !----------
      END IF
    END DO

    10 IF (errio /= 0) THEN
         WRITE(scratchmessage, '(a)') 'Error in file: '  // trim(adjustl(inpfile)) // &
                                 ', while processing line: ' // trim(adjustl(inpline))
         CALL allmessage(error, scratchmessage)

         CLOSE(iunit)

         CALL unsetmessagesource()

         CALL terminate()
       END IF

    11 IF (ios /= 0) THEN
         WRITE(scratchmessage, '(a)') 'Error in file: '  // trim(adjustl(inpfile)) // &
                                 ', while processing line: ' // trim(adjustl(line))
         CALL allmessage(error, scratchmessage)

         CLOSE(iunit)

         CALL unsetmessagesource()

         CALL terminate()
       END IF

    20 CLOSE(iunit)

    besttrackdata(ifile)%thisStorm = ''
    besttrackdata(ifile)%loaded    = .true.
    besttrackdata(ifile)%numRec    = nlines


    !------------------------------------------------------------
    ! Get the unique storm name and store it in the thisStorm string
    ALLOCATE(chkarrstr(nlines))
    ALLOCATE(idxarrstr(nlines))

    nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)

    maxcnt = -1
    DO kcnt = 1, nunique
      kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
      IF (kmax > maxcnt) THEN
        maxcnt = kmax
        besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
      END IF
    END DO

    DEALLOCATE(chkarrstr)
    DEALLOCATE(idxarrstr)
    !------------------------------------------------------------

    !------------------------------------------------------------
    ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
    ! stored in ascending order
    ALLOCATE(idx0(besttrackdata(ifile)%numRec))
    ALLOCATE(idx1(besttrackdata(ifile)%numRec))

    CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)

    IF (status /= 0) THEN
      CALL unsetmessagesource()

      CALL terminate()
    END IF

    ! Create the index array to be used in the comparison below
    idx0 = arth(1, 1, besttrackdata(ifile)%numRec)

    IF (.NOT. arrayequal(idx0, idx1)) THEN
      besttrackdata(ifile)%basin     =  besttrackdata(ifile)%basin(idx1)
      besttrackdata(ifile)%cyNum     =  besttrackdata(ifile)%cyNum(idx1)
      besttrackdata(ifile)%dtg       =  besttrackdata(ifile)%dtg(idx1)
      besttrackdata(ifile)%techNum   =  besttrackdata(ifile)%techNum(idx1)
      besttrackdata(ifile)%tech      =  besttrackdata(ifile)%tech(idx1)
      besttrackdata(ifile)%tau       =  besttrackdata(ifile)%tau(idx1)
```

```
        besttrackdata(ifile)%intLat     =  besttrackdata(ifile)%intLat(idx1)
        besttrackdata(ifile)%ns         =  besttrackdata(ifile)%ns(idx1)
        besttrackdata(ifile)%intLon     =  besttrackdata(ifile)%intLon(idx1)
        besttrackdata(ifile)%ew         =  besttrackdata(ifile)%ew(idx1)
        besttrackdata(ifile)%intVMax    =  besttrackdata(ifile)%intVMax(idx1)
        besttrackdata(ifile)%intMslp    =  besttrackdata(ifile)%intMslp(idx1)
        besttrackdata(ifile)%ty         =  besttrackdata(ifile)%ty(idx1)
        besttrackdata(ifile)%rad        =  besttrackdata(ifile)%rad(idx1)
        besttrackdata(ifile)%windCode   =  besttrackdata(ifile)%windCode(idx1)
        besttrackdata(ifile)%intRad1    =  besttrackdata(ifile)%intRad1(idx1)
        besttrackdata(ifile)%intRad2    =  besttrackdata(ifile)%intRad2(idx1)
        besttrackdata(ifile)%intRad3    =  besttrackdata(ifile)%intRad3(idx1)
        besttrackdata(ifile)%intRad4    =  besttrackdata(ifile)%intRad4(idx1)
        besttrackdata(ifile)%intPOuter =  besttrackdata(ifile)%intPOuter(idx1)
        besttrackdata(ifile)%intROuter =  besttrackdata(ifile)%intROuter(idx1)
        besttrackdata(ifile)%intRmw     =  besttrackdata(ifile)%intRmw(idx1)
        besttrackdata(ifile)%gusts      =  besttrackdata(ifile)%gusts(idx1)
        besttrackdata(ifile)%eye        =  besttrackdata(ifile)%eye(idx1)
        besttrackdata(ifile)%subregion =  besttrackdata(ifile)%subregion(idx1)
        besttrackdata(ifile)%maxseas    =  besttrackdata(ifile)%maxseas(idx1)
        besttrackdata(ifile)%initials   =  besttrackdata(ifile)%initials(idx1)
        besttrackdata(ifile)%dir        =  besttrackdata(ifile)%dir(idx1)
        besttrackdata(ifile)%intSpeed   =  besttrackdata(ifile)%intSpeed(idx1)
        besttrackdata(ifile)%stormName =  besttrackdata(ifile)%stormName(idx1)
        besttrackdata(ifile)%cycleNum   =  besttrackdata(ifile)%cycleNum(idx1)
      END IF

      DEALLOCATE(idx0)
      DEALLOCATE(idx1)
      !------------------------------------------------------------

      CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
    END DO ! End of "iFile" loop

    CALL unsetmessagesource()

  END SUBROUTINE readbesttrackfile

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   R E A D  C S V  B E S T  T R A C K F I L E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE readcsvbesttrackfile()

    USE pahm_global, ONLY : nbtrfiles, besttrackfilename
    USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique, &
                          intvalstr
    USE sortutils, ONLY : arth, indexx, arrayequal
    USE csv_module

    IMPLICIT NONE

    TYPE(csv_file)                 :: f
    CHARACTER(LEN=64), ALLOCATABLE :: sval2D(:, :)
    LOGICAL                        :: statusOK

    CHARACTER(LEN=FNAMELEN)        :: inpFile
    CHARACTER(LEN=512)             :: line
    CHARACTER(LEN=64)              :: tmpStr

    INTEGER                        :: iFile, nLines, lenLine
    INTEGER                        :: iCnt, jCnt, kCnt, kMax       ! loop counters
    INTEGER                        :: ios, status

    CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
    INTEGER, ALLOCATABLE           :: idxArrStr(:)
    INTEGER                        :: nUnique, maxCnt

    INTEGER, ALLOCATABLE           :: idx0(:), idx1(:)


    CALL setmessagesource("ReadCsvBestTrackFile")

    ! Allocate the best track structure array. This structure holds all the
    ! input values for the storm track as read in from the track input file
    ! (a-deck, b-deck ATCF format) as well as the converted best track variables
    ! (as appropriate).
    ALLOCATE(besttrackdata(nbtrfiles))

```

```
    ! This is the main loop. We loop through all the best track files
    ! (user input)
    DO ifile = 1, nbtrfiles
      inpfile = besttrackfilename(ifile)

      besttrackdata(ifile)%fileName  = trim(adjustl(inpfile))
      besttrackdata(ifile)%thisStorm = ""
      besttrackdata(ifile)%loaded    = .false.
      besttrackdata(ifile)%numRec    = -1

      CALL f%Read(trim(adjustl(inpfile)), status_ok=statusok)
      CALL f%Get(sval2d, status_ok=statusok)

      ! Array allocation in the structure bestTrackData
      nlines = f%n_rows
      CALL allocbtrstruct(besttrackdata(ifile), nlines)

      DO icnt = 1, nlines
        DO jcnt = 1 , f%n_cols
          line = line // trim(adjustl(sval2d(icnt, jcnt)))
        END DO
        jcnt = 0

        lenline = len_trim(adjustl(line))

        IF (lenline /= 0) THEN
          !--- col:  1
          besttrackdata(ifile)%basin(icnt)     = trim(adjustl(sval2d(icnt, 1)))
          !--- col:  2
          besttrackdata(ifile)%cyNum(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 2))))
          !--- col:  3
          besttrackdata(ifile)%dtg(icnt)       = trim(adjustl(sval2d(icnt, 3)))
          !--- col:  4
          besttrackdata(ifile)%techNum(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 4))))
          !--- col:  5
          besttrackdata(ifile)%tech(icnt)      = trim(adjustl(sval2d(icnt, 5)))
          !--- col:  6
          besttrackdata(ifile)%tau(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 6))))
          !--- col:  7
          tmpstr = trim(adjustl(sval2d(icnt, 7)))
          READ(tmpstr, '(i3, a1)') &
               besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
          !--- col:  8
          tmpstr = trim(adjustl(sval2d(icnt, 8)))
          READ(tmpstr, '(i3, a1)') &
               besttrackdata(ifile)%intLon(icnt), besttrackdata(ifile)%ew(icnt)
          !--- col:  9
          besttrackdata(ifile)%intVMax(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 9))))
          !--- col: 10
          besttrackdata(ifile)%intMslp(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 10))))
          !--- col: 11
          besttrackdata(ifile)%ty(icnt)        = trim(adjustl(sval2d(icnt, 11)))
          !--- col: 12
          besttrackdata(ifile)%rad(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 12))))
          !--- col: 13
          besttrackdata(ifile)%windCode(icnt)  = trim(adjustl(sval2d(icnt, 13)))
          !--- col: 14
          besttrackdata(ifile)%intRad1(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 14))))
          !--- col: 15
          besttrackdata(ifile)%intRad2(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 15))))
          !--- col: 16
          besttrackdata(ifile)%intRad3(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 16))))
          !--- col: 17
          besttrackdata(ifile)%intRad4(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 17))))
          !--- col: 18
          besttrackdata(ifile)%intPOuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 18))))
          !--- col: 19
          besttrackdata(ifile)%intROuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 19))))
          !--- col: 20
          besttrackdata(ifile)%intRmw(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 20))))
          !--- col: 21
          besttrackdata(ifile)%gusts(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 21))))
          !--- col: 22
          besttrackdata(ifile)%eye(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 22))))
          !--- col: 23
          besttrackdata(ifile)%subregion(icnt) = trim(adjustl(sval2d(icnt, 23)))
          !--- col: 24
          besttrackdata(ifile)%maxseas(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 24))))
          !--- col: 25
           besttrackdata(ifile)%initials(icnt) = trim(adjustl(sval2d(icnt, 25)))
          !--- col: 26
```

```
    besttrackdata(ifile)%dir(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 26))))
    !--- col: 27
    besttrackdata(ifile)%intSpeed(icnt)  = intvalstr(trim(adjustl(sval2d(icnt, 27))))
    !--- col: 28
    besttrackdata(ifile)%stormName(icnt) = trim(adjustl(sval2d(icnt, 28)))

    ! This is for the cycleNum, the last column we consider
    IF (icnt == 1) THEN
      kcnt = icnt
      besttrackdata(ifile)%cycleNum(icnt) = icnt
    ELSE
      kcnt = kcnt + 1
      IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
        besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
        kcnt = kcnt - 1
      ELSE
        besttrackdata(ifile)%cycleNum(icnt) = kcnt
      END IF
    END IF

    !---------- Convert lat/lon values to S/N and W/E notations
    IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
      besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
    ELSE
      besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
    END IF

    IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
      besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
    ELSE
      besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
    END IF
    !----------

    !---------- Get the year, month, day, hour from the DGT string
    READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios) besttrackdata(ifile)%year(icnt)
      IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
    READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%month(icnt)
      IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
    READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%day(icnt)
      IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
    READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios) besttrackdata(ifile)%hour(icnt)
      IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
    !----------

  END IF
END DO

besttrackdata(ifile)%thisStorm = ''
besttrackdata(ifile)%loaded    = .true.
besttrackdata(ifile)%numRec    = nlines

!------------------------------------------------------------
! Get the unique storm name and store it in the thisStorm string
ALLOCATE(chkarrstr(nlines))
ALLOCATE(idxarrstr(nlines))

nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)

maxcnt = -1
DO kcnt = 1, nunique
  kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
  IF (kmax > maxcnt) THEN
    maxcnt = kmax
    besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
  END IF
END DO

DEALLOCATE(chkarrstr)
DEALLOCATE(idxarrstr)
!------------------------------------------------------------

!------------------------------------------------------------
! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
! stored in ascending order
ALLOCATE(idx0(besttrackdata(ifile)%numRec))
ALLOCATE(idx1(besttrackdata(ifile)%numRec))
```

```
      CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)

      IF (status /= 0) THEN
        CALL unsetmessagesource()

        CALL terminate()
      END IF

      ! Create the index array to be used in the comparison below
      idx0 = arth(1, 1, besttrackdata(ifile)%numRec)

      IF (.NOT. arrayequal(idx0, idx1)) THEN
        besttrackdata(ifile)%basin     =  besttrackdata(ifile)%basin(idx1)
        besttrackdata(ifile)%cyNum     =  besttrackdata(ifile)%cyNum(idx1)
        besttrackdata(ifile)%dtg       =  besttrackdata(ifile)%dtg(idx1)
        besttrackdata(ifile)%techNum   =  besttrackdata(ifile)%techNum(idx1)
        besttrackdata(ifile)%tech      =  besttrackdata(ifile)%tech(idx1)
        besttrackdata(ifile)%tau       =  besttrackdata(ifile)%tau(idx1)
        besttrackdata(ifile)%intLat    =  besttrackdata(ifile)%intLat(idx1)
        besttrackdata(ifile)%ns        =  besttrackdata(ifile)%ns(idx1)
        besttrackdata(ifile)%intLon    =  besttrackdata(ifile)%intLon(idx1)
        besttrackdata(ifile)%ew        =  besttrackdata(ifile)%ew(idx1)
        besttrackdata(ifile)%intVMax   =  besttrackdata(ifile)%intVMax(idx1)
        besttrackdata(ifile)%intMslp   =  besttrackdata(ifile)%intMslp(idx1)
        besttrackdata(ifile)%ty        =  besttrackdata(ifile)%ty(idx1)
        besttrackdata(ifile)%rad       =  besttrackdata(ifile)%rad(idx1)
        besttrackdata(ifile)%windCode  =  besttrackdata(ifile)%windCode(idx1)
        besttrackdata(ifile)%intRad1   =  besttrackdata(ifile)%intRad1(idx1)
        besttrackdata(ifile)%intRad2   =  besttrackdata(ifile)%intRad2(idx1)
        besttrackdata(ifile)%intRad3   =  besttrackdata(ifile)%intRad3(idx1)
        besttrackdata(ifile)%intRad4   =  besttrackdata(ifile)%intRad4(idx1)
        besttrackdata(ifile)%intPOuter =  besttrackdata(ifile)%intPOuter(idx1)
        besttrackdata(ifile)%intROuter =  besttrackdata(ifile)%intROuter(idx1)
        besttrackdata(ifile)%intRmw    =  besttrackdata(ifile)%intRmw(idx1)
        besttrackdata(ifile)%gusts     =  besttrackdata(ifile)%gusts(idx1)
        besttrackdata(ifile)%eye       =  besttrackdata(ifile)%eye(idx1)
        besttrackdata(ifile)%subregion =  besttrackdata(ifile)%subregion(idx1)
        besttrackdata(ifile)%maxseas   =  besttrackdata(ifile)%maxseas(idx1)
        besttrackdata(ifile)%initials  =  besttrackdata(ifile)%initials(idx1)
        besttrackdata(ifile)%dir       =  besttrackdata(ifile)%dir(idx1)
        besttrackdata(ifile)%intSpeed  =  besttrackdata(ifile)%intSpeed(idx1)
        besttrackdata(ifile)%stormName =  besttrackdata(ifile)%stormName(idx1)
        besttrackdata(ifile)%cycleNum  =  besttrackdata(ifile)%cycleNum(idx1)
      END IF

      DEALLOCATE(idx0)
      DEALLOCATE(idx1)
      !------------------------------------------------------------

      CALL f%Destroy()

      CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')

    END DO ! End of "iFile" loop

    CALL unsetmessagesource()

  END SUBROUTINE readcsvbesttrackfile

!================================================================================

  !----------------------------------------------------------------
  !  S U B R O U T I N E   P R O C E S S   H O L L A N D   D A T A
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE processhollanddata(idTrFile, strOut, status)

    USE pahm_global, ONLY : nm2m, kt2ms, nbtrfiles
    USE utilities, ONLY : touppercase, charunique
    USE timedateutils, ONLY : timeconv
    USE pahm_vortex, ONLY : calcintensitychange, uvtrans

    IMPLICIT NONE

    INTEGER, INTENT(IN)              :: idTrFile
    TYPE(HollandData_T), INTENT(OUT) :: strOut
    INTEGER, INTENT(OUT)             :: status ! error status

    ! numUniqRec, outDTG, idxDTG are used to identify the unique DTG elements in the input structure
    INTEGER                          :: numUniqRec
```

```
    CHARACTER(LEN=10), ALLOCATABLE   :: outDTG(:)
    INTEGER, ALLOCATABLE             :: idxDTG(:)

    INTEGER                          :: plIdx            ! populated index for Holland Data array
    INTEGER                          :: iCnt             ! loop counters

    CHARACTER(LEN=4)                 :: castType         !hindcast,forecast
    REAL(SZ), ALLOCATABLE            :: castTime(:)      ! seconds since start of year

    REAL(SZ)                         :: spdVal, pressVal, rrpVal, rmwVal

    status = 0  ! no error


    CALL setmessagesource("ProcessHollandData")

    IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
      IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
        status = 2

        WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ', idtrfile
        CALL allmessage(error, scratchmessage)

        CALL unsetmessagesource()

        RETURN
      END IF
    ELSE
      status = 1

      WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
                                              ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ', nbtrfiles
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      RETURN
    END IF

    WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
    CALL logmessage(info, scratchmessage)

    ! Most likely the array size will be larger if repeated times are found
    ! in the best track structure.
    ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
    ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))

    ! Get unique lines that represent new points in time.
    ! Repeated time points occur in hindcasts for the purpose of
    ! describing winds in the quadrants of the storm. We don't use the
    ! quadrant-by-quadrant wind data. Repeated time data occur in the
    ! forecast because the time data is just the time that the forecast
    ! was made. The important parameter in the forecast file is the
    ! forecast increment.
    numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)

    !--------------------
    ! Populate the Holland structure
    !--------------------
    CALL allochollstruct(strout, numuniqrec)

    ALLOCATE(casttime(numuniqrec))

    strout%fileName  = besttrackdata(idtrfile)%fileName
    strout%thisStorm = besttrackdata(idtrfile)%thisStorm
    strout%loaded    = .true.
    strout%numRec    = numuniqrec

    WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
    CALL logmessage(info, scratchmessage)

    DO icnt = 1, numuniqrec
      plidx = idxdtg(icnt)

      casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(plidx))))

      ! Convert speeds from knots to m/s
      spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)

```

```
      ! Convert pressure(s) from mbar to Pa
      pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)

      ! Convert all distances from nm to km/m
      rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
      rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx)    ! in m

      strout%basin(icnt)       = besttrackdata(idtrfile)%basin(plidx)
      strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)
      strout%dtg(icnt)         = besttrackdata(idtrfile)%dtg(plidx)
      strout%year(icnt)        = besttrackdata(idtrfile)%year(plidx)
      strout%month(icnt)       = besttrackdata(idtrfile)%month(plidx)
      strout%day(icnt)         = besttrackdata(idtrfile)%day(plidx)
      strout%hour(icnt)        = besttrackdata(idtrfile)%hour(plidx)
      strout%castType(icnt)    = besttrackdata(idtrfile)%tech(plidx)
      strout%fcstInc(icnt)     = besttrackdata(idtrfile)%tau(plidx)
      strout%iLat(icnt)        = besttrackdata(idtrfile)%intLat(plidx)
      strout%lat(icnt)         = besttrackdata(idtrfile)%lat(plidx)
      strout%iLon(icnt)        = besttrackdata(idtrfile)%intLon(plidx)
      strout%lon(icnt)         = besttrackdata(idtrfile)%lon(plidx)

      strout%iSpeed(icnt)      = besttrackdata(idtrfile)%intVMax(plidx)
      strout%speed(icnt)       = spdval
      strout%iCPress(icnt)     = besttrackdata(idtrfile)%intMslp(plidx)
      strout%cPress(icnt)      = pressval
      strout%iRrp(icnt)        = besttrackdata(idtrfile)%intROuter(plidx)
      strout%rrp(icnt)         = rrpval
      strout%iRmw(icnt)        = besttrackdata(idtrfile)%intRmw(plidx)
      strout%rmw(icnt)         = rmwval

      ! PV check if this SELECT code is actually needed. Need to check the different format
      ! of input files.
      SELECT CASE(casttype)
        CASE("BEST")     ! nowcast/hindcast
          ! PV check if this is needed
          CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0, 0.0_sz, casttime(icnt))

        CASE("OFCL")     ! forecast
          ! PV check if this is needed
          IF (icnt > 1) THEN
            IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1))) cycle
          END IF

          IF (strout%fcstInc(icnt) == 0) THEN
            CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
                          strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
          ELSE
           casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) * 3600.0_sz)
          END IF

          IF ((strout%iCPress(icnt) == 0) .OR. (strout%iRmw(icnt) == 0)) THEN
            CALL allmessage(error,                                                             &
                            'The storm hindcast/forecast input file ' // trim(strout%fileName) //  &
                            ' contains invalid data for central pressure or rMax.')
            CALL terminate()
          END IF

        ! Adding a new type to allow the analyst to add lines
        ! that do nothing but produce zero winds and background barometric
        ! pressure. These lines can have a date/time like a BEST line or
        ! a date/time and forecast period like an OFCL line.
        CASE("CALM")
          ! PV check if this is needed
          WRITE(scratchmessage, '(a)') 'The file: ' // trim(strout%fileName) // ' contains at least one "CALM" line.'
          CALL logmessage(echo, scratchmessage)

          IF (icnt > 1) THEN
            IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1))) cycle
          END IF

          IF (strout%fcstInc(icnt) == 0) THEN
            CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
                          strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
          ELSE
            casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) * 3600.0_sz)
```

```
          END IF

        CASE DEFAULT        ! unrecognized
          WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of ' // &
                                       trim(adjustl(strout%fileName))
          CALL allmessage(error, scratchmessage)

          CALL terminate()
      END SELECT

      strout%castTime(icnt) = casttime(icnt)
    END DO   ! numUniqRec

    ! Calculate the cPress intensity change (dP/dt)
    CALL calcintensitychange(strout%cPress, casttime, strout%cPrDt, status, 2)

    ! Calculate storm translation velocities based on change in position,
    ! approximate u and v translation velocities
    CALL uvtrans(strout%lat, strout%lon, casttime, strout%trVx, strout%trVy, status, 2)

    DEALLOCATE(casttime)
!--------------------

    DEALLOCATE(outdtg)
    DEALLOCATE(idxdtg)

    CALL unsetmessagesource()

  END SUBROUTINE processhollanddata

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   G E T   H O L L A N D   F I E L D S
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE gethollandfields(timeIDX)

    USE pahm_mesh, ONLY : slam, sfea, xcslam, ycsfea, np, ismeshok
    USE pahm_global, ONLY : gravity, rhowater, rhoair,                        &
                            backgroundatmpress, bladjustfac, one2ten,      &
                            deg2rad, rad2deg, basee, omega, mb2pa, mb2kpa, &
                            nbtrfiles, besttrackfilename,                  &
                            noutdt, mdbegsimtime, mdendsimtime, mdoutdt,   &
                            wvelx, wvely, wpress, times
    USE utilities, ONLY : sphericaldistance, sphericalfracpoint, getlocandratio
    USE timedateutils, ONLY : juldaytogreg, gregtojulday
    !USE PaHM_NetCDFIO

    IMPLICIT NONE

    INTEGER, INTENT(IN)                    :: timeIDX

    TYPE(hollanddata_t), ALLOCATABLE     :: holStru(:)          ! array of Holland data structures
    INTEGER                              :: stormNumber         ! storm identification number
    REAL(SZ)                             :: hlB                 ! Holland B parameter
    REAL(SZ)                             :: rrp                 ! radius of the last closed isobar (m)
    REAL(SZ)                             :: rmw                 ! radius of max winds (m)
    REAL(SZ)                             :: speed               ! maximum sustained wind speed (m/s)
    REAL(SZ)                             :: cPress              ! central pressure (Pa)
    REAL(SZ)                             :: cPressDef           ! pressure deficit: Ambient Press - cPress (Pa)
    REAL(SZ)                             :: trVX, trVY, trSPD   ! storm translation velocities (m/s)
    REAL(SZ)                             :: trSpdX, trSpdY      ! adjusted translation velocities (m/s)
    REAL(SZ)                             :: lon, lat            ! current eye location

    REAL(SZ), ALLOCATABLE                :: rad(:)              ! distance of nodal points from the eye location
    INTEGER, ALLOCATABLE                 :: radIDX(:)           ! indices of nodal points duch that rad <= rrp
    INTEGER                              :: maxRadIDX           ! total number of radIDX elements
    REAL(SZ)                             :: windMultiplier      ! for storm 2 in lpfs ensemble DO WE NEED THIS?
    REAL(SZ)                             :: dx, dy, theta
    REAL(SZ)                             :: wtRatio
    REAL(SZ)                             :: coriolis

    REAL(SZ)                             :: sfPress             ! calculated surface MSL pressure (Pa)
    REAL(SZ)                             :: grVel               ! wind speed (m/s) at gradient level (top of ABL)
```

```
    REAL(SZ)                               :: sfVelX, sfVelY      ! calculated surface (10-m above ground) wind velocities (m/s)

    INTEGER                                :: iCnt, stCnt, npCnt
    INTEGER                                :: i, jl1, jl2
    INTEGER                                :: status

    CHARACTER(LEN=64)                      :: tmpTimeStr, tmpStr1, tmpStr2

    LOGICAL, SAVE                          :: firstCall = .true.

    CALL setmessagesource("GetHollandFields")

    ! Check if timeIDX is within bounds (1 <= timeIDX <= nOutDT). If it is not then exit the program.
    IF ((timeidx < 1) .OR. (timeidx > noutdt)) THEN
        WRITE(tmpstr1, '(a, i0)') 'timeIDX = ', timeidx
        WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
        WRITE(scratchmessage, '(a)') 'timeIDX should be: 1 <= timeIDX <= nOutDT :' // &
                                     trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
        CALL allmessage(error, scratchmessage)

        CALL unsetmessagesource()

        CALL terminate()
    END IF

    ! This part of the code should only be executed just once
    IF (firstcall) THEN
      firstcall = .false.

      ! Check if the mash variables are set and that nOutDT is greater than zero.
      IF (.NOT. ismeshok) THEN
        WRITE(scratchmessage, '(a)') 'The mesh variables are not established properly. ' // &
                                     'Call subroutine ReadMesh to read/create the mesh topology first.'
        CALL allmessage(error, scratchmessage)

        CALL unsetmessagesource()

        CALL terminate()
      ELSE
        IF ((np <= 0) .OR. (noutdt <= 0)) THEN
          WRITE(tmpstr1, '(a, i0)') 'np = ', np
          WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
          WRITE(scratchmessage, '(a)') 'Variables "np" or "nOutDT" are not defined properly: ' // &
                                       trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
          CALL allmessage(error, scratchmessage)

          CALL unsetmessagesource()

          CALL terminate()
        END IF
      END IF

      ! Allocate storage for the Times array that contains the output times.
      ALLOCATE(times(noutdt))
      DO icnt = 1, noutdt
        times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
      END DO
    END IF


    !-----------------------------
    ! Allocate storage for required arrays.
    IF (.NOT. ALLOCATED(wvelx))  ALLOCATE(wvelx(np))
    IF (.NOT. ALLOCATED(wvely))  ALLOCATE(wvely(np))
    IF (.NOT. ALLOCATED(wpress)) ALLOCATE(wpress(np))

    ! Initialize the arrays. Here we are resetting the fields to their defaults.
    ! This subroutine is called repeatdly and its time the following fields
    ! are recalculated.
    wvelx  = 0.0_sz
    wvely  = wvelx
    wpress = backgroundatmpress * mb2pa
    !-----------------------------

    !-----------------------------
    ! ALLOCATE THE HOLLAND DATA STRUCTURES AND STORE THE HOLLAND
    ! DATA INTO THE DATA STRUCTURE ARRAY FOR SUBSEQUENT USE
    !-----------------------------
    !
    ! Allocate the array of Holland data structures. The Holland
```

```
    ! structures are allocated by calling the ProcessHollandData
    ! subroutine.
    ALLOCATE(holstru(nbtrfiles))

    ! Process and store the "best track" data into the array of Holland structures
    ! for subsequent use. All required data to generate the P-W model wind fields
    ! are contained in these structures. We take into consideration that might be
    ! more than one "best track" file for the simulation period.
    DO stcnt = 1, nbtrfiles
      CALL processhollanddata(stcnt, holstru(stcnt), status)

      IF (.NOT. holstru(stcnt)%loaded) THEN
        WRITE(scratchmessage, '(a)') 'There was an error loading the Holland data structure for the best track file: ' // &
                                     trim(adjustl(besttrackfilename(stcnt)))
        CALL allmessage(error, scratchmessage)

        CALL deallochollstruct(holstru(stcnt))
        DEALLOCATE(holstru)

        CALL unsetmessagesource()

        CALL terminate()
      ELSE IF (status /= 0) THEN
        WRITE(scratchmessage, '(a)') 'There was an error processing the Holland data structure for the best track file: ' // &
                                     trim(adjustl(besttrackfilename(stcnt)))
        CALL allmessage(error, scratchmessage)

        CALL deallochollstruct(holstru(stcnt))
        DEALLOCATE(holstru)

        CALL unsetmessagesource()

        CALL terminate()
      ELSE
        WRITE(scratchmessage, '(a)') 'Processing the Holland data structure for the best track file: ' // &
                                     trim(adjustl(besttrackfilename(stcnt)))
        CALL logmessage(info, scratchmessage)
      END IF
    END DO
    !------------------------------

    !------------------------------
    ! THIS IS THE MAIN TIME LOOP   timeIDX
    !------------------------------
!    WRITE(scratchMessage, '(a)') 'Start of the main time loop'
!    CALL AllMessage(INFO, scratchMessage)
!    DO iCnt = 1, nOutDT
        icnt = timeidx
        WRITE(tmpstr1, '(i5)') icnt
        WRITE(tmpstr2, '(i5)') noutdt
      tmpstr1 = '(' // trim(tmpstr1) // '/' // trim(adjustl(tmpstr2)) // ')'
        WRITE(tmptimestr, '(f20.3)') times(icnt)
      WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(tmpstr1)) // " " // trim(adjustl(tmptimestr))
      CALL allmessage(scratchmessage)

      DO stcnt = 1, nbtrfiles
        ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
        ! factor for the subsequent linear interpolation in time. In order for this to
        ! work, the array holStru%castTime should be ordered in ascending order.
        CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, jl1, jl2, wtratio)

        ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
        ! by exiting this loop
        IF ((jl1 <= 0) .OR. (jl2 <= 0)) THEN
          WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(tmptimestr)) // &
                                       ', skipping generating data for this time'
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        ! Perform linear interpolation in time
        stormnumber = holstru(stcnt)%stormNumber(jl1)

        CALL sphericalfracpoint(holstru(stcnt)%lat(jl1), holstru(stcnt)%lon(jl1), &
                                holstru(stcnt)%lat(jl2), holstru(stcnt)%lon(jl2), &
                                wtratio, lat, lon)
```

```
        !lat    = holStru(stCnt)%lat(jl1) + &
        !         wtRatio * (holStru(stCnt)%lat(jl2) - holStru(stCnt)%lat(jl1))
        !lon    = holStru(stCnt)%lon(jl1) + &
        !         wtRatio * (holStru(stCnt)%lon(jl2) - holStru(stCnt)%lon(jl1))

        ! Radius of the last closed isobar
        rrp = holstru(stcnt)%rrp(jl1) + &
                wtratio * (holstru(stcnt)%rrp(jl2) - holstru(stcnt)%rrp(jl1))

        ! Radius of maximum winds
        rmw = holstru(stcnt)%rmw(jl1) + &
                wtratio * (holstru(stcnt)%rmw(jl2) - holstru(stcnt)%rmw(jl1))

        ! Get all the distances of the mesh nodes from (lat, lon)
        rad    = sphericaldistance(sfea, slam, lat, lon)
        ! ... and the indices of the nodal points where rad <= rrp
        radidx = pack([(i, i = 1, np)], rad <= rrp)
        maxradidx = SIZE(radidx)

        ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
        IF (maxradidx == 0) THEN
          WRITE(tmpstr1, '(f20.3)') rrp
            tmpstr1 = '(rrp = ' // trim(adjustl(tmpstr1)) // ' m)'
          WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar
' // &
                                        trim(adjustl(tmpstr1)) // ' for storm: ' // &
                                        trim(adjustl(holstru(stcnt)%thisStorm))
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        speed  = holstru(stcnt)%speed(jl1) + &
                  wtratio * (holstru(stcnt)%speed(jl2) - holstru(stcnt)%speed(jl1))

        cpress = holstru(stcnt)%cPress(jl1) + &
                  wtratio * (holstru(stcnt)%cPress(jl2) - holstru(stcnt)%cPress(jl1))

        trvx   = holstru(stcnt)%trVx(jl1) + &
                wtratio * (holstru(stcnt)%trVx(jl2) - holstru(stcnt)%trVx(jl1))
        trvy   = holstru(stcnt)%trVy(jl1) + &
                wtratio * (holstru(stcnt)%trVy(jl2) - holstru(stcnt)%trVy(jl1))

        ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
        ! background pressure and return. PV: check if this is actually needed
        IF (cpress < 0.0_sz) THEN
          wvelx  = 0.0_sz
          wvely  = wvelx
          wpress = backgroundatmpress * mb2pa

          WRITE(scratchmessage, '(a)') 'Calm period found, generating zero atmospheric fields for this
time'
          CALL logmessage(info, scratchmessage)

          EXIT
        END IF

        ! Calculate and limit central pressure deficit; some track files (e.g., Charley 2004)
        ! may have a central pressure greater than the ambient pressure that this subroutine assumes
        cpressdef = backgroundatmpress * mb2pa - cpress
        IF (cpressdef < 100.0_sz) cpressdef = 100.0_sz

        ! Subtract the translational speed of the storm from the observed max wind speed to avoid
        ! distortion in the Holland curve fit. The translational speed will be added back later.
        trspd = sqrt(trvx * trvx + trvy * trvy)
        speed = speed - trspd

        ! Convert wind speed from 10 meter altitude (which is what the
        ! NHC forecast contains) to wind speed at the top of the atmospheric
        ! boundary layer (which is what the Holland curve fit requires).
        speed = speed / bladjustfac

        ! Calculate Holland parameters and limit the result to its appropriate range.
        hlb = rhoair * basee * (speed**2) / cpressdef
        IF (hlb < 1.0_sz) hlb = 1.0_sz
        IF (hlb > 2.5_sz) hlb = 2.5_sz

        ! If we are running storm 2 in the Lake Pontchartrain !PV Do we need this?
        ! Forecast System ensemble, the final wind speeds should be multiplied by 1.2.
        windmultiplier = 1.0_sz
        IF (stormnumber == 2) windmultiplier = 1.2_sz
```

```
      DO npcnt = 1, maxradidx
        i = radidx(npcnt)

        dx    = sphericaldistance(lat, lon, lat, slam(i))
        dy    = sphericaldistance(lat, lon, sfea(i), lon)
        theta = atan2(dy, dx)

        ! Compute coriolis
        coriolis = 2.0_sz * omega * sin(sfea(i) * deg2rad)

        ! Compute the pressure (Pa) at a distance rad(i); all distances are in meters
        sfpress = cpress + cpressdef * exp(-(rmw / rad(i))**hlb)

        ! Compute wind speed (speed - trSPD) at gradient level (m/s) and at a distance rad(i);
        ! all distances are in meters. Using absolute value for coriolis for Southern Hempisphere
        grvel = sqrt(speed**2 * (rmw / rad(i))**hlb * exp(1.0_sz - (rmw / rad(i))**hlb) +   &
                     (rad(i) * abs(coriolis) / 2.0_sz)**2) -                                  &
                rad(i) * abs(coriolis) / 2.0_sz

        ! Determine translation speed that should be added to final !PV CHECK ON THIS
        ! storm wind speed. This is tapered to zero as the storm wind tapers
        ! to zero toward the eye of the storm and at long distances from the storm.
        trspdx = (abs(grvel) / speed ) * trvx
        trspdy = (abs(grvel) / speed ) * trvy

        ! Apply mutliplier for Storm #2 in LPFS ensemble.
        grvel = grvel * windmultiplier

        ! Find the wind velocity components.
        sfvelx = -grvel * sin(theta)
        sfvely =  grvel * cos(theta)
        !print *, sfVelX, sfVelY
        ! Convert wind velocity from the gradient level (top of atmospheric boundary layer)
        ! which, is what the Holland curve fit produces, to 10-m wind velocity.
        sfvelx = sfvelx * bladjustfac
        sfvely = sfvely * bladjustfac
        !print *, sfVelX, sfVelY
        ! Convert from 1-minute averaged winds to 10-minute averaged winds.
        sfvelx = sfvelx * one2ten
        sfvely = sfvely * one2ten
        !print *, sfVelX, sfVelY
        ! Add back the storm translation speed.
        sfvelx = sfvelx + trspdx
        sfvely = sfvely + trspdy

        !print *, sfVelX, sfVelY, wVelX(i), wVelY(i)
        !PV Need to interpolate between storms if this nodal point
        !   is affected by more than on storm
        wpress(i) = sfpress
        wvelx(i)  = sfvelx
        wvely(i)  = sfvely

        !print *, sfVelX, sfVelY, wVelX(i), wVelY(i)
        !print *, '-----------------------------------------'
      END DO ! npCnt = 1, maxRadIDX

    END DO ! stCnt = 1, nBTrFiles
!    END DO ! iCnt = 1, nOutDT
!    WRITE(scratchMessage, '(a)') 'End of the main time loop'
!    CALL AllMessage(INFO, scratchMessage)

    !---------- Deallocate the arrays
    IF (ALLOCATED(rad)) DEALLOCATE(rad)
    IF (ALLOCATED(radidx)) DEALLOCATE(radidx)
    DO icnt = 1, nbtrfiles
      CALL deallocholstruct(holstru(icnt))
    END DO
    DEALLOCATE(holstru)
    !----------

    CALL unsetmessagesource()

  END SUBROUTINE gethollandfields

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   W R I T E   B E S T   T R A C K   D A T A
  !----------------------------------------------------------------
  !----------------------------------------------------------------
```

```
 SUBROUTINE writebesttrackdata(inpFile, btrStruc, suffix)

   USE pahm_global, ONLY : lun_btrk, lun_btrk1

   IMPLICIT NONE

   ! Global variables
   CHARACTER(LEN=*)                      :: inpFile
   TYPE(besttrackdata_t), INTENT(IN)      :: btrStruc
   CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: suffix

   ! Local variables
   CHARACTER(LEN=FNAMELEN)                :: outFile
   CHARACTER(LEN=64)                      :: fSuf
   INTEGER                                :: iCnt
   INTEGER                                :: iUnit, errIO
   CHARACTER(LEN=512)                     :: fmtStr


   !---------- Initialize variables
   iunit  = lun_btrk1
   errio  = 0

   fmtstr = '(a2, ",", 1x, i2.2, ",", 1x, a10, ",", 1x, i2, ",", 1x, a4, ",", 1x, i3, ",", 1x, i3, a1, ",", 1x, i4, a1, ",", '
     fmtstr = trim(fmtstr) // ' 1x, i3, ",", 1x, i4, ",", 1x, a2, ",", 1x, i3, ",", 1x, a3, ",", '
     fmtstr = trim(fmtstr) // ' 4(1x, i4, ","), 1x, i4, ",", 1x, i4, ",", 1x, i3, ",", 1x, i4, ",", 1x, i3, ",", '
     fmtstr = trim(fmtstr) // ' 1x, a3,",", 1x, i3,",", 1x, a3, ",", i3,",", 1x, i3,",", 1x, a11,",", 1x, i3, ",")'
   !----------

   fsuf = '_adj'
   IF (PRESENT(suffix)) fsuf = adjustl(suffix)

   CALL setmessagesource("WriteBestTrackData")

   IF (.NOT. btrstruc%loaded) THEN
     WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be written."
     CALL allmessage(info, scratchmessage)

     RETURN
   END IF

   outfile = trim(adjustl(inpfile)) // trim(fsuf)

   WRITE(scratchmessage, '(a)') 'Writing the "adjusted" best track data to: ' // trim(adjustl(outfile))
   CALL logmessage(info, scratchmessage)

   OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)

   IF (errio /= 0) THEN
     WRITE(scratchmessage, '(a)') 'Error opening the outFile: '  // trim(outfile) // &
                                  ', skip writting the "adjusted" best track fields'
     CALL allmessage(error, scratchmessage)

     RETURN
   END IF

   DO icnt = 1, btrstruc%numRec
     WRITE(iunit, fmtstr)                                   &
         btrstruc%basin(icnt),     btrstruc%cyNum(icnt),      &
         btrstruc%dtg(icnt),       btrstruc%techNum(icnt),    &
         btrstruc%tech(icnt),      btrstruc%tau(icnt),        &
         btrstruc%intLat(icnt),    btrstruc%ns(icnt),         &
         btrstruc%intLon(icnt),    btrstruc%ew(icnt),         &
         btrstruc%intVMax(icnt),   btrstruc%intMslp(icnt),    &
         btrstruc%ty(icnt),        btrstruc%rad(icnt),        &
         btrstruc%windCode(icnt),  btrstruc%intRad1(icnt),    &
         btrstruc%intRad2(icnt),   btrstruc%intRad3(icnt),    &
         btrstruc%intRad4(icnt),   btrstruc%intPOuter(icnt),  &
         btrstruc%intROuter(icnt), btrstruc%intRmw(icnt),     &
         btrstruc%gusts(icnt),     btrstruc%eye(icnt),        &
         btrstruc%subregion(icnt), btrstruc%maxseas(icnt),    &
         btrstruc%initials(icnt),  btrstruc%dir(icnt),        &
         btrstruc%intSpeed(icnt),  btrstruc%stormName(icnt),  &
         btrstruc%cycleNum(icnt)
   END DO

   CLOSE(iunit)
```

```

    CALL unsetmessagesource()

  END SUBROUTINE writebesttrackdata

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   A L L O C  B T R  S T R U C T
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE allocbtrstruct(str, nRec)

    IMPLICIT NONE

    TYPE(besttrackdata_t) :: str
    INTEGER, INTENT(IN)   :: nRec

    str%numRec = nrec
    str%loaded = .false.

    !----- Input parameters
    IF (.NOT. ALLOCATED(str%basin))     ALLOCATE(str%basin(nrec))
    IF (.NOT. ALLOCATED(str%cyNum))     ALLOCATE(str%cyNum(nrec))
    IF (.NOT. ALLOCATED(str%dtg))       ALLOCATE(str%dtg(nrec))
    IF (.NOT. ALLOCATED(str%techNum))   ALLOCATE(str%techNum(nrec))
    IF (.NOT. ALLOCATED(str%tech))      ALLOCATE(str%tech(nrec))
    IF (.NOT. ALLOCATED(str%tau))       ALLOCATE(str%tau(nrec))
    IF (.NOT. ALLOCATED(str%intLat))    ALLOCATE(str%intLat(nrec))
    IF (.NOT. ALLOCATED(str%intLon))    ALLOCATE(str%intLon(nrec))
    IF (.NOT. ALLOCATED(str%ew))        ALLOCATE(str%ew(nrec))
    IF (.NOT. ALLOCATED(str%ns))        ALLOCATE(str%ns(nrec))
    IF (.NOT. ALLOCATED(str%intVMax))   ALLOCATE(str%intVMax(nrec))
    IF (.NOT. ALLOCATED(str%intMslp))   ALLOCATE(str%intMslp(nrec))
    IF (.NOT. ALLOCATED(str%ty))        ALLOCATE(str%ty(nrec))
    IF (.NOT. ALLOCATED(str%rad))       ALLOCATE(str%rad(nrec))
    IF (.NOT. ALLOCATED(str%windCode))  ALLOCATE(str%windCode(nrec))
    IF (.NOT. ALLOCATED(str%intRad1))   ALLOCATE(str%intRad1(nrec))
    IF (.NOT. ALLOCATED(str%intRad2))   ALLOCATE(str%intRad2(nrec))
    IF (.NOT. ALLOCATED(str%intRad3))   ALLOCATE(str%intRad3(nrec))
    IF (.NOT. ALLOCATED(str%intRad4))   ALLOCATE(str%intRad4(nrec))
    IF (.NOT. ALLOCATED(str%intPOuter)) ALLOCATE(str%intPOuter(nrec))
    IF (.NOT. ALLOCATED(str%intROuter)) ALLOCATE(str%intROuter(nrec))
    IF (.NOT. ALLOCATED(str%intRmw))    ALLOCATE(str%intRmw(nrec))
    IF (.NOT. ALLOCATED(str%gusts))     ALLOCATE(str%gusts(nrec))
    IF (.NOT. ALLOCATED(str%eye))       ALLOCATE(str%eye(nrec))
    IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
    IF (.NOT. ALLOCATED(str%maxseas))   ALLOCATE(str%maxseas(nrec))
    IF (.NOT. ALLOCATED(str%initials))  ALLOCATE(str%initials(nrec))
    IF (.NOT. ALLOCATED(str%dir))       ALLOCATE(str%dir(nrec))
    IF (.NOT. ALLOCATED(str%intSpeed))  ALLOCATE(str%intSpeed(nrec))
    IF (.NOT. ALLOCATED(str%stormName)) ALLOCATE(str%stormName(nrec))
    IF (.NOT. ALLOCATED(str%cycleNum))  ALLOCATE(str%cycleNum(nrec))

    !----- Converted parameters
    IF (.NOT. ALLOCATED(str%year))      ALLOCATE(str%year(nrec))
    IF (.NOT. ALLOCATED(str%month))     ALLOCATE(str%month(nrec))
    IF (.NOT. ALLOCATED(str%day))       ALLOCATE(str%day(nrec))
    IF (.NOT. ALLOCATED(str%hour))      ALLOCATE(str%hour(nrec))
    IF (.NOT. ALLOCATED(str%lat))       ALLOCATE(str%lat(nrec))
    IF (.NOT. ALLOCATED(str%lon))       ALLOCATE(str%lon(nrec))

  END SUBROUTINE allocbtrstruct

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   D E A L L O C  B T R  S T R U C T
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE deallocbtrstruct(str)

    IMPLICIT NONE

    TYPE(besttrackdata_t) :: str

    str%numRec = -1
    str%loaded = .false.

    !----- Input parameters
    IF (ALLOCATED(str%basin))     DEALLOCATE(str%basin)
```

```
    IF (ALLOCATED(str%cyNum))      DEALLOCATE(str%cyNum)
    IF (ALLOCATED(str%dtg))        DEALLOCATE(str%dtg)
    IF (ALLOCATED(str%techNum))    DEALLOCATE(str%techNum)
    IF (ALLOCATED(str%tech))       DEALLOCATE(str%tech)
    IF (ALLOCATED(str%tau))        DEALLOCATE(str%tau)
    IF (ALLOCATED(str%intLat))     DEALLOCATE(str%intLat)
    IF (ALLOCATED(str%intLon))     DEALLOCATE(str%intLon)
    IF (ALLOCATED(str%ew))         DEALLOCATE(str%ew)
    IF (ALLOCATED(str%ns))         DEALLOCATE(str%ns)
    IF (ALLOCATED(str%intVMax))    DEALLOCATE(str%intVMax)
    IF (ALLOCATED(str%intMslp))    DEALLOCATE(str%intMslp)
    IF (ALLOCATED(str%ty))         DEALLOCATE(str%ty)
    IF (ALLOCATED(str%rad))        DEALLOCATE(str%rad)
    IF (ALLOCATED(str%windCode))   DEALLOCATE(str%windCode)
    IF (ALLOCATED(str%intRad1))    DEALLOCATE(str%intRad1)
    IF (ALLOCATED(str%intRad2))    DEALLOCATE(str%intRad2)
    IF (ALLOCATED(str%intRad3))    DEALLOCATE(str%intRad3)
    IF (ALLOCATED(str%intRad4))    DEALLOCATE(str%intRad4)
    IF (ALLOCATED(str%intPOuter))  DEALLOCATE(str%intPOuter)
    IF (ALLOCATED(str%intROuter))  DEALLOCATE(str%intROuter)
    IF (ALLOCATED(str%intRmw))     DEALLOCATE(str%intRmw)
    IF (ALLOCATED(str%gusts))      DEALLOCATE(str%gusts)
    IF (ALLOCATED(str%eye))        DEALLOCATE(str%eye)
    IF (ALLOCATED(str%subregion))  DEALLOCATE(str%subregion)
    IF (ALLOCATED(str%maxseas))    DEALLOCATE(str%maxseas)
    IF (ALLOCATED(str%initials))   DEALLOCATE(str%initials)
    IF (ALLOCATED(str%dir))        DEALLOCATE(str%dir)
    IF (ALLOCATED(str%intSpeed))   DEALLOCATE(str%intSpeed)
    IF (ALLOCATED(str%stormName))  DEALLOCATE(str%stormName)
    IF (ALLOCATED(str%cycleNum))   DEALLOCATE(str%cycleNum)

     !----- Converted parameters
    IF (ALLOCATED(str%year))       DEALLOCATE(str%year)
    IF (ALLOCATED(str%month))      DEALLOCATE(str%month)
    IF (ALLOCATED(str%day))        DEALLOCATE(str%day)
    IF (ALLOCATED(str%hour))       DEALLOCATE(str%hour)
    IF (ALLOCATED(str%lat))        DEALLOCATE(str%lat)
    IF (ALLOCATED(str%lon))        DEALLOCATE(str%lon)

  END SUBROUTINE deallocbtrstruct

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   A L L O C   H O L L   S T R U C T
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE allochollstruct(str, nRec)

    IMPLICIT NONE

    TYPE(hollanddata_t) :: str
    INTEGER, INTENT(IN) :: nRec

    str%numRec = nrec
    str%loaded = .false.

    !----- Input parameters
    IF (.NOT. ALLOCATED(str%basin))       ALLOCATE(str%basin(nrec))

    IF (.NOT. ALLOCATED(str%dtg))         ALLOCATE(str%dtg(nrec))
    IF (.NOT. ALLOCATED(str%stormNumber)) ALLOCATE(str%stormNumber(nrec))
    IF (.NOT. ALLOCATED(str%year))        ALLOCATE(str%year(nrec))
    IF (.NOT. ALLOCATED(str%month))       ALLOCATE(str%month(nrec))
    IF (.NOT. ALLOCATED(str%day))         ALLOCATE(str%day(nrec))
    IF (.NOT. ALLOCATED(str%hour))        ALLOCATE(str%hour(nrec))

    IF (.NOT. ALLOCATED(str%castTime))    ALLOCATE(str%castTime(nrec))
    IF (.NOT. ALLOCATED(str%castType))    ALLOCATE(str%castType(nrec))
    IF (.NOT. ALLOCATED(str%fcstInc))     ALLOCATE(str%fcstInc(nrec))

    IF (.NOT. ALLOCATED(str%iLat))        ALLOCATE(str%iLat(nrec))
    IF (.NOT. ALLOCATED(str%lat))         ALLOCATE(str%lat(nrec))
    IF (.NOT. ALLOCATED(str%iLon))        ALLOCATE(str%iLon(nrec))
    IF (.NOT. ALLOCATED(str%lon))         ALLOCATE(str%lon(nrec))

    IF (.NOT. ALLOCATED(str%iSpeed))      ALLOCATE(str%iSpeed(nrec))
    IF (.NOT. ALLOCATED(str%speed))       ALLOCATE(str%speed(nrec))

    IF (.NOT. ALLOCATED(str%iCPress))     ALLOCATE(str%iCPress(nrec))
    IF (.NOT. ALLOCATED(str%cPress))      ALLOCATE(str%cPress(nrec))
```

```
   IF (.NOT. ALLOCATED(str%iRrp))         ALLOCATE(str%iRrp(nrec))
   IF (.NOT. ALLOCATED(str%rrp))          ALLOCATE(str%rrp(nrec))

   IF (.NOT. ALLOCATED(str%iRmw))         ALLOCATE(str%iRmw(nrec))
   IF (.NOT. ALLOCATED(str%rmw))          ALLOCATE(str%rmw(nrec))

   IF (.NOT. ALLOCATED(str%cPrDt))        ALLOCATE(str%cPrDt(nrec))

   IF (.NOT. ALLOCATED(str%trVx))         ALLOCATE(str%trVx(nrec))
   IF (.NOT. ALLOCATED(str%trVy))         ALLOCATE(str%trVy(nrec))

 END SUBROUTINE allochollstruct

!================================================================================

 !----------------------------------------------------------------
 ! S U B R O U T I N E   D E A L L O C  H O L L  S T R U C T
 !----------------------------------------------------------------
 !----------------------------------------------------------------
 SUBROUTINE deallochollstruct(str)

   IMPLICIT NONE

   TYPE(hollanddata_t), INTENT(OUT) :: str

   str%numRec = -1
   str%loaded = .false.

   !----- Input parameters
   IF (ALLOCATED(str%basin))        DEALLOCATE(str%basin)

   IF (ALLOCATED(str%dtg))          DEALLOCATE(str%dtg)
   IF (ALLOCATED(str%stormNumber))  DEALLOCATE(str%stormNumber)
   IF (ALLOCATED(str%year))         DEALLOCATE(str%year)
   IF (ALLOCATED(str%month))        DEALLOCATE(str%month)
   IF (ALLOCATED(str%day))          DEALLOCATE(str%day)
   IF (ALLOCATED(str%hour))         DEALLOCATE(str%hour)

   IF (ALLOCATED(str%castTime))     DEALLOCATE(str%castTime)
   IF (ALLOCATED(str%castType))     DEALLOCATE(str%castType)
   IF (ALLOCATED(str%fcstInc))      DEALLOCATE(str%fcstInc)

   IF (ALLOCATED(str%iLat))         DEALLOCATE(str%iLat)
   IF (ALLOCATED(str%lat))          DEALLOCATE(str%lat)
   IF (ALLOCATED(str%iLon))         DEALLOCATE(str%iLon)
   IF (ALLOCATED(str%lon))          DEALLOCATE(str%lon)

   IF (ALLOCATED(str%iSpeed))       DEALLOCATE(str%iSpeed)
   IF (ALLOCATED(str%speed))        DEALLOCATE(str%speed)

   IF (ALLOCATED(str%iCPress))      DEALLOCATE(str%iCPress)
   IF (ALLOCATED(str%cPress))       DEALLOCATE(str%cPress)

   IF (ALLOCATED(str%iRrp))         DEALLOCATE(str%iRrp)
   IF (ALLOCATED(str%rrp))          DEALLOCATE(str%rrp)

   IF (ALLOCATED(str%iRmw))         DEALLOCATE(str%iRmw)
   IF (ALLOCATED(str%rmw))          DEALLOCATE(str%rmw)

   IF (ALLOCATED(str%cPrDt))        DEALLOCATE(str%cPrDt)

   IF (ALLOCATED(str%trVx))         DEALLOCATE(str%trVx)
   IF (ALLOCATED(str%trVy))         DEALLOCATE(str%trVy)

 END SUBROUTINE deallochollstruct

!================================================================================

END MODULE parwind
```

9.32 sizes.F90 File Reference

Contains the definitions of various number types and utilities used in PaHM.

Data Types

- interface pahm_sizes::comparereals
- interface pahm_sizes::fixnearwholereal

Modules

- module pahm_sizes

Functions/Subroutines

- integer function pahm_sizes::comparedoublereals (rVal1, rVal2, eps)

  Compares two double precision numbers.
- integer function pahm_sizes::comparesinglereals (rVal1, rVal2, eps)

  Compares two single precision numbers.
- real(hp) function pahm_sizes::fixnearwholedoublereal (rVal, eps)

  Rounds a double precision real number to its nearest whole number.
- real(sp) function pahm_sizes::fixnearwholesinglereal (rVal, eps)

  Rounds a single precision real number to its nearest whole number.

Variables

- integer, parameter pahm_sizes::sp = SELECTED_REAL_KIND(6, 37)
- integer, parameter pahm_sizes::hp = SELECTED_REAL_KIND(15, 307)
- integer, parameter pahm_sizes::int16 = SELECTED_INT_KIND(38)
- integer, parameter pahm_sizes::int8 = SELECTED_INT_KIND(18)
- integer, parameter pahm_sizes::int4 = SELECTED_INT_KIND(9)
- integer, parameter pahm_sizes::int2 = SELECTED_INT_KIND(4)
- integer, parameter pahm_sizes::int1 = SELECTED_INT_KIND(2)
- integer, parameter pahm_sizes::long = INT8
- integer, parameter pahm_sizes::llong = INT16
- integer, parameter pahm_sizes::wp = HP
- integer, parameter pahm_sizes::ip = INT8
- integer, parameter pahm_sizes::sz = HP
- integer, parameter pahm_sizes::nbyte = 8
- real(sz), parameter pahm_sizes::rmissv = -999999.0_SZ
- integer, parameter pahm_sizes::imissv = -999999
- character(len=1), parameter pahm_sizes::blank = ' '
- integer, parameter pahm_sizes::fnamelen = 1024

9.32.1 Detailed Description

Contains the definitions of various number types and utilities used in PaHM.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file sizes.F90.

9.33 sizes.F90

Go to the documentation of this file.

```
!-----------------------------------------------------------------
!                M O D U L E   S I Z E S
!-----------------------------------------------------------------
!-----------------------------------------------------------------

MODULE pahm_sizes

  IMPLICIT NONE

  !----------------------------------------------------------------------
  ! I N T E R F A C E S
  !----------------------------------------------------------------------
  INTERFACE comparereals
    MODULE PROCEDURE comparesinglereals
    MODULE PROCEDURE comparedoublereals
  END INTERFACE comparereals

  INTERFACE fixnearwholereal
    MODULE PROCEDURE fixnearwholesinglereal
    MODULE PROCEDURE fixnearwholedoublereal
  END INTERFACE fixnearwholereal
  !----------------------------------------------------------------------

  ! SP = single precision, HP = high (double) precision
  INTEGER, PARAMETER :: sp = selected_real_kind(6,   37)   !  6  digits, range \([10^{-37}  , 10^{+37}   - 1]\), 32 bits
  INTEGER, PARAMETER :: hp = selected_real_kind(15, 307)   ! 15  digits, range \([10^{-307} , 10^{+307}  - 1]\), 64 bits

  ! Precision of integers:
  INTEGER, PARAMETER :: int16 = selected_int_kind(38)      ! Range \([-2^{127},+2^{127} - 1]\), 39 digits plus sign; 128 bits
  INTEGER, PARAMETER ::  int8 = selected_int_kind(18)      ! Range \([-2^{63},+2^{63} - 1]\),   19 digits plus sign;  64 bits
  INTEGER, PARAMETER ::  int4 = selected_int_kind( 9)      ! Range \([-2^{31},+2^{31} - 1]\),   10 digits plus sign;  32 bits
  INTEGER, PARAMETER ::  int2 = selected_int_kind( 4)      ! Range \([-2^{15},+2^{15} - 1]\),    5 digits plus sign;  16 bits
  INTEGER, PARAMETER ::  int1 = selected_int_kind( 2)      ! Range \([-2^{7} ,+2^{7}  - 1]\),    3 digits plus sign;   8 bits
  INTEGER, PARAMETER :: long  = int8
  INTEGER, PARAMETER :: llong = int16

  INTEGER,PARAMETER :: wp = hp   ! default real kind (for csv_module)
  INTEGER,PARAMETER :: ip = int8 ! default integer kind (for csv_module)

  ! By default we perform all calculations in double precision
  ! SET NUMBER OF BYTES "SZ" IN REAL(SZ) DECLARATIONS
  ! SET "NBYTE" FOR PROCESSING INPUT DATA RECORD LENGTH
#ifdef REAL4
  INTEGER, PARAMETER :: sz = sp
  INTEGER, PARAMETER :: nbyte = 4
#else
  INTEGER, PARAMETER :: sz = hp
  INTEGER, PARAMETER :: nbyte = 8
#endif

  ! Used to initialize the mesh arrays and in NetCDF output files for missing values.
  ! Also used to initialize some input variables to check if these variables
  ! were supplied user defined values.
  REAL(sz), PARAMETER :: rmissv = -999999.0_sz
  INTEGER, PARAMETER  :: imissv = -999999

  CHARACTER(LEN=1), PARAMETER :: blank = ' '

  ! Filename length (considers the presence of the full path in the filename)
  INTEGER, PARAMETER :: fnamelen = 1024


  CONTAINS


  !-------------------------------------------------------------
  ! F U N C T I O N   CompareDoubleReals
  !-------------------------------------------------------------
  !-------------------------------------------------------------
```

```
  INTEGER FUNCTION comparedoublereals(rVal1, rVal2, eps) RESULT(myValOut)

    IMPLICIT NONE

    ! Global variables
    REAL(hp), INTENT(IN)           :: rval1, rval2
    REAL(hp), OPTIONAL, INTENT(IN) :: eps

    ! Local variables
    REAL(hp)                       :: epssys, epsusr, value


    epssys = 2.0_hp * epsilon(rval1)

    IF (PRESENT(eps)) THEN
      epsusr = abs(eps)
    ELSE
      epsusr = epssys
    ENDIF

    IF ((abs(rval1) < 1.0_hp) .OR. (abs(rval2) < 1.0_hp)) THEN
      value = rval1 - rval2
    ELSE
      value = (rval1 - rval2) / max(rval1, rval2)
      IF (abs(value) < 1.0_hp) value = rval1 - rval2
    END IF

    IF (abs(value) < epsusr) THEN
      myvalout = 0
    ELSE IF (rval1 < rval2) THEN
      myvalout = -1
    ELSE
      myvalout = 1
    END IF

    RETURN

  END FUNCTION comparedoublereals

!================================================================================

  !-----------------------------------------------------------------
  ! F U N C T I O N   C O M P A R E   S I N G L E   R E A L S
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  INTEGER FUNCTION comparesinglereals(rVal1, rVal2, eps) RESULT(myValOut)

    IMPLICIT NONE

    ! Global variables
    REAL(sp), INTENT(IN)           :: rval1, rval2
    REAL(sp), OPTIONAL, INTENT(IN) :: eps

    ! Local variables
    REAL(sp)                       :: epssys, epsusr, value


    epssys = 2.0_sp * epsilon(rval1)

    IF (PRESENT(eps)) THEN
      epsusr = abs(eps)
    ELSE
      epsusr = epssys
    ENDIF

    IF ((abs(rval1) < 1.0_sp) .OR. (abs(rval2) < 1.0_sp)) THEN
      value = rval1 - rval2
    ELSE
      value = (rval1 - rval2) / max(rval1, rval2)
      IF (abs(value) < 1.0_sp) value = rval1 - rval2
    END IF

    IF (abs(value) < epsusr) THEN
      myvalout = 0
    ELSE IF (rval1 < rval2) THEN
      myvalout = -1
    ELSE
      myvalout = 1
    END IF

    RETURN
```

```fortran

  END FUNCTION comparesinglereals

!==============================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   F I X  N E A R  W H O L E  D O U B L E  R E A L
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(hp) function fixnearwholedoublereal(rval, eps) result(myvalout)

    IMPLICIT NONE

    ! Global Variables
    REAL(hp), INTENT(IN)           :: rval
    REAL(hp), OPTIONAL, INTENT(IN) :: eps

    ! Local Variables
    REAL(hp)                       :: epssys, epsusr, value


    epssys = 2.0_hp * epsilon(rval)

    IF (PRESENT(eps)) THEN
      epsusr = abs(eps)
    ELSE
      epsusr = epssys
    ENDIF

    myvalout = rval
    value    = anint(myvalout)
    IF (comparereals(myvalout, value, epsusr) == 0) myvalout = value

    RETURN

  END FUNCTION fixnearwholedoublereal

!==============================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   F I X  N E A R  W H O L E  S I N G L E  R E A L
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sp) function fixnearwholesinglereal(rval, eps) result(myvalout)

    IMPLICIT NONE

    ! Global Variables
    REAL(sp), INTENT(IN)           :: rval
    REAL(sp), OPTIONAL, INTENT(IN) :: eps

    ! Local Variables
    REAL(sp)                       :: epssys, epsusr, value


    epssys = 2.0_sp * epsilon(rval)

    IF (PRESENT(eps)) THEN
      epsusr = abs(eps)
    ELSE
      epsusr = epssys
    ENDIF

    myvalout = rval
    value    = anint(myvalout)
    IF (comparereals(myvalout, value, epsusr) == 0) myvalout = value

    RETURN

  END FUNCTION fixnearwholesinglereal

!==============================================================================

END MODULE pahm_sizes
```

9.34 sortutils.F90 File Reference

Data Types

- interface sortutils::indexx
- interface sortutils::arth
- interface sortutils::arraycopy
- interface sortutils::arrayequal
- interface sortutils::swap

Modules

- module sortutils

Functions/Subroutines

- subroutine sortutils::indexxint (arr1D, idx1D, status)
  Indexes a 1D integer array in ascending order.
- subroutine icompxchg (i, j)
- subroutine sortutils::indexxint8 (arr1D, idx1D, status)
  Indexes a 1D 32-bit integer array in ascending order.
- subroutine sortutils::indexxstring (arr1D, idx1D, status, caseSens)
  Indexes a 1D string array in ascending order.
- subroutine sortutils::indexxsingle (arr1D, idx1D, status)
  Indexes a 1D single precision array in ascending order.
- subroutine sortutils::indexxdouble (arr1D, idx1D, status)
  Indexes a 1D double precision array in ascending order.
- subroutine sortutils::quicksort (arr1D, status)
  Sorts the array arr1D into ascending numerical order using Quicksort.
- subroutine sortutils::sort2 (arr1D, slv1D, status)
  Sorts two 1D arrays into ascending numerical order using Quicksort.
- subroutine sortutils::arraycopyint (src, dest, nCP, nNCP)
  Copies the 1D source integer array "src" into the 1D destination array "dest".
- subroutine sortutils::arraycopysingle (src, dest, nCP, nNCP)
  Copies the 1D source single precision array "src" into the 1D destination array "dest".
- subroutine sortutils::arraycopydouble (src, dest, nCP, nNCP)
  Copies the 1D source double precision array "src" into the 1D destination array "dest".
- logical function sortutils::arrayequalint (arr1, arr2)
  Compares two one-dimensional integer arrays for equality.
- logical function sortutils::arrayequalsingle (arr1, arr2)
  Compares two one-dimensional single precision arrays for equality.
- logical function sortutils::arrayequaldouble (arr1, arr2)
  Compares two one-dimensional double precision arrays for equality.
- integer function sortutils::stringlexcomp (str1, str2, mSensitive)
  Performs a lexical comparison between two strings.
- subroutine sortutils::swapint (a, b, mask)

Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine sortutils::swapsingle (a, b, mask)

  Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine sortutils::swapdouble (a, b, mask)

  Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").
- subroutine sortutils::swapintvec (a, b, mask)

  Swaps the contents of a and b (integer). increment and a number of terms "n" (including "first").
- subroutine sortutils::swapsinglevec (a, b, mask)

  Swaps the contents of a and b (single precision). increment and a number of terms "n" (including "first").
- subroutine sortutils::swapdoublevec (a, b, mask)

  Swaps the contents of a and b (double precision). increment and a number of terms "n" (including "first").
- pure integer function, dimension(n) sortutils::arthint (first, increment, n)

  Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(sp) function, dimension(n) sortutils::arthsingle (first, increment, n)

  Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").
- pure real(hp) function, dimension(n) sortutils::arthdouble (first, increment, n)

  Returns an arithmetic progression, given a first term "first", an increment and a number of terms "n" (including "first").

9.34.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file sortutils.F90.

9.34.2 Function/Subroutine Documentation

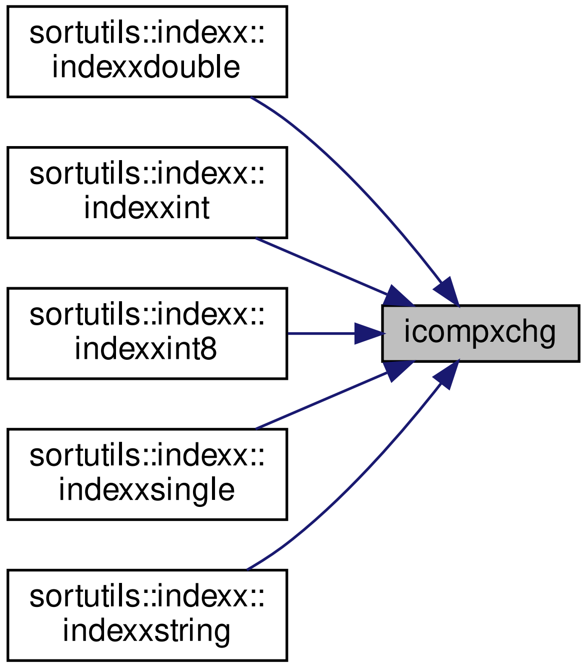
9.34.2.1 icompxchg()

```
subroutine icompxchg (
        integer, intent(inout) i,
        integer, intent(inout) j )
```

Definition at line 214 of file sortutils.F90.

Referenced by sortutils::indexx::indexxdouble(), sortutils::indexx::indexxint(), sortutils::indexx::indexxint8(), sortutils::indexx::indexxsingle() and sortutils::indexx::indexxstring().

Here is the caller graph for this function:



9.35 sortutils.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                M O D U L E   U T I L I T I E S
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE sortutils

  USE pahm_sizes
  USE pahm_messages

  !----------------------------------------------------------------
  ! I N T E R F A C E S
  !----------------------------------------------------------------
  INTERFACE indexx
    MODULE PROCEDURE indexxint
    MODULE PROCEDURE indexxint8
    MODULE PROCEDURE indexxstring
    MODULE PROCEDURE indexxsingle
    MODULE PROCEDURE indexxdouble
  END INTERFACE indexx

  INTERFACE arth
    MODULE PROCEDURE arthint
    MODULE PROCEDURE arthsingle
    MODULE PROCEDURE arthdouble
  END INTERFACE arth

```

```
INTERFACE arraycopy
  MODULE PROCEDURE arraycopyint
  MODULE PROCEDURE arraycopysingle
  MODULE PROCEDURE arraycopydouble
END INTERFACE arraycopy

INTERFACE arrayequal
  MODULE PROCEDURE arrayequalint
  MODULE PROCEDURE arrayequalsingle
  MODULE PROCEDURE arrayequaldouble
END INTERFACE arrayequal

INTERFACE swap
  MODULE PROCEDURE swapint
  MODULE PROCEDURE swapsingle
  MODULE PROCEDURE swapdouble
  MODULE PROCEDURE swapintvec
  MODULE PROCEDURE swapsinglevec
  MODULE PROCEDURE swapdoublevec
END INTERFACE swap
!-----------------------------------------------------------------------


CONTAINS


!----------------------------------------------------------------
! S U B R O U T I N E   I N D E X X  I N T
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE indexxint(arr1D, idx1D, status)

  IMPLICIT NONE

  ! Global variables
  INTEGER, DIMENSION(:), INTENT(IN)  :: arr1D
  INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
  INTEGER, OPTIONAL, INTENT(OUT)     :: status

  ! Local variables
  INTEGER, PARAMETER                 :: NN = 15, nstack = 50
  INTEGER                            :: a
  INTEGER                            :: nARR, nIDX, tmpIDX
  INTEGER                            :: k, i, j, l, r
  INTEGER                            :: ist, stack(NSTACK)
  CHARACTER(LEN=64)                  :: tmpStr1, tmpStr2


  CALL setmessagesource("IndexxInt")

  IF (PRESENT(status)) status = 0

  narr = SIZE(arr1d, 1)
  nidx = SIZE(idx1d, 1)

  IF (narr /= nidx) THEN
    WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
    WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
    WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
                                 trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

    CALL allmessage(error, scratchmessage)
    CALL unsetmessagesource()

    IF (PRESENT(status)) status = 1

    RETURN
  END IF

  idx1d = arth(1, 1, narr)

  ist = 0
  l   = 1
  r   = narr

  DO
    IF (r - l < nn) THEN
      DO j = l + 1, r
        tmpidx = idx1d(j)
        a = arr1d(tmpidx)
        DO i = j - 1, l, -1
```

```
          IF (arr1d(idx1d(i)) <= a) EXIT
          idx1d(i + 1) = idx1d(i)
        END DO
        idx1d(i + 1) = tmpidx
      END DO

      IF (ist == 0) THEN
        CALL unsetmessagesource()

        RETURN
      END IF

      r   = stack(ist)
      l   = stack(ist - 1)
      ist = ist - 2
    ELSE
      k = (l + r) / 2

      CALL swap(idx1d(k), idx1d(l + 1))
      CALL icompxchg(idx1d(l), idx1d(r))
      CALL icompxchg(idx1d(l + 1), idx1d(r))
      CALL icompxchg(idx1d(l), idx1d(l + 1))

      i = l + 1
      j = r
      tmpidx = idx1d(l + 1)
      a = arr1d(tmpidx)

      DO
        DO
          i = i + 1
          IF (arr1d(idx1d(i)) > a) EXIT
        END DO

        DO
          j = j - 1
          IF (arr1d(idx1d(j)) < a) EXIT
        END DO

        IF (j < i) EXIT
        CALL swap(idx1d(i), idx1d(j))
      END DO

      idx1d(l + 1) = idx1d(j)
      idx1d(j) = tmpidx
      ist = ist + 2

      IF (ist > nstack) THEN
        WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
        WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                     trim(adjustl(tmpstr1))

        CALL logmessage(error, scratchmessage)
        CALL unsetmessagesource()

        IF (PRESENT(status)) status = 2

        RETURN

      END IF

      IF (r - i + 1 >= j - l) THEN
        stack(ist) = r
        stack(ist - 1) = i
        r = j - 1
      ELSE
        stack(ist) = j - 1
        stack(ist - 1) = l
        l = i
      END IF
    END IF
  END DO

  CALL unsetmessagesource()


  CONTAINS

  SUBROUTINE icompxchg(i, j)

    IMPLICIT NONE
```

```

    ! Global variables
    INTEGER, INTENT(INOUT) :: i, j

    ! Local variables
    INTEGER :: swp

    IF (arr1d(j) < arr1d(i)) THEN
      swp = i
      i   = j
      j   = swp
    END IF

  END SUBROUTINE icompxchg

END SUBROUTINE indexxint

!===============================================================================

!-----------------------------------------------------------------
! S U B R O U T I N E   I N D E X X  I N T  8
!-----------------------------------------------------------------
!-----------------------------------------------------------------
SUBROUTINE indexxint8(arr1D, idx1D, status)

  IMPLICIT NONE

  ! Global variables
  INTEGER(INT8), DIMENSION(:), INTENT(IN)  :: arr1D
  INTEGER, DIMENSION(:), INTENT(OUT)       :: idx1D
  INTEGER, OPTIONAL, INTENT(OUT)           :: status

  ! Local variables
  INTEGER, PARAMETER                       :: NN = 15, nstack = 50
  INTEGER(INT8)                            :: a
  INTEGER                                  :: nARR, nIDX, tmpIDX
  INTEGER                                  :: k, i, j, l, r
  INTEGER                                  :: ist, stack(NSTACK)
  CHARACTER(LEN=64)                        :: tmpStr1, tmpStr2


  CALL setmessagesource("IndexxInt8")

  IF (PRESENT(status)) status = 0

  narr = SIZE(arr1d, 1)
  nidx = SIZE(idx1d, 1)

  IF (narr /= nidx) THEN
    WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
    WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
    WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
                                 trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

    CALL allmessage(error, scratchmessage)
    CALL unsetmessagesource()

    IF (PRESENT(status)) status = 1

    RETURN
  END IF

  idx1d = arth(1, 1, narr)

  ist = 0
  l   = 1
  r   = narr

  DO
    IF (r - l < nn) THEN
      DO j = l + 1, r
        tmpidx = idx1d(j)
        a = arr1d(tmpidx)
        DO i = j - 1, l, -1
          IF (arr1d(idx1d(i)) <= a) EXIT
          idx1d(i + 1) = idx1d(i)
        END DO
        idx1d(i + 1) = tmpidx
      END DO

      IF (ist == 0) THEN
```

```
          CALL unsetmessagesource()

          RETURN
        END IF

        r   = stack(ist)
        l   = stack(ist - 1)
        ist = ist - 2
      ELSE
        k = (l + r) / 2

        CALL swap(idx1d(k), idx1d(l + 1))
        CALL icompxchg(idx1d(l), idx1d(r))
        CALL icompxchg(idx1d(l + 1), idx1d(r))
        CALL icompxchg(idx1d(l), idx1d(l + 1))

        i = l + 1
        j = r
        tmpidx = idx1d(l + 1)
        a = arr1d(tmpidx)

        DO
          DO
            i = i + 1
            IF (arr1d(idx1d(i)) > a) EXIT
          END DO

          DO
            j = j - 1
            IF (arr1d(idx1d(j)) < a) EXIT
          END DO

          IF (j < i) EXIT
          CALL swap(idx1d(i), idx1d(j))
        END DO

        idx1d(l + 1) = idx1d(j)
        idx1d(j) = tmpidx
        ist = ist + 2

        IF (ist > nstack) THEN
          WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
          WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                       trim(adjustl(tmpstr1))

          CALL logmessage(error, scratchmessage)
          CALL unsetmessagesource()

          IF (PRESENT(status)) status = 2

          RETURN

        END IF

        IF (r - i + 1 >= j - l) THEN
          stack(ist) = r
          stack(ist - 1) = i
          r = j - 1
        ELSE
          stack(ist) = j - 1
          stack(ist - 1) = l
          l = i
        END IF
      END IF
    END DO

    CALL unsetmessagesource()


    CONTAINS

    SUBROUTINE icompxchg(i, j)

      IMPLICIT NONE

      ! Global variables
      INTEGER, INTENT(INOUT) :: i, j

      ! Local variables
      INTEGER :: swp

```

```fortran
      IF (arr1d(j) < arr1d(i)) THEN
        swp = i
        i   = j
        j   = swp
      END IF

    END SUBROUTINE icompxchg

  END SUBROUTINE indexxint8

  !==============================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   I N D E X X  S T R I N G
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE indexxstring(arr1D, idx1D, status, caseSens)

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), DIMENSION(:), INTENT(IN) :: arr1D
    LOGICAL, OPTIONAL, INTENT(IN)              :: caseSens
    INTEGER, DIMENSION(:), INTENT(OUT)         :: idx1D
    INTEGER, OPTIONAL, INTENT(OUT)             :: status

    ! Local variables
    INTEGER, PARAMETER                         :: NN = 15, nstack = 50
    CHARACTER(LEN=LEN(arr1D(1)))               :: a
    INTEGER                                    :: nARR, nIDX, tmpIDX
    INTEGER                                    :: k, i, j, l, r
    INTEGER                                    :: ist, stack(NSTACK)
    CHARACTER(LEN=64)                          :: tmpStr1, tmpStr2
    LOGICAL                                    :: sFlag


    CALL setmessagesource("IndexxString")

    sflag = .true.
    IF (PRESENT(casesens)) sflag = casesens

    IF (PRESENT(status)) status   = 0

    narr = SIZE(arr1d, 1)
    nidx = SIZE(idx1d, 1)

    IF (narr /= nidx) THEN
      WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
      WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
      WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
                                   trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

      CALL allmessage(error, scratchmessage)
      CALL unsetmessagesource()

      IF (PRESENT(status)) status = 1

      RETURN
    END IF

    idx1d = arth(1, 1, narr)

    ist = 0
    l   = 1
    r   = narr

    DO
      IF (r - l < nn) THEN
        DO j = l + 1, r
          tmpidx = idx1d(j)
          a = arr1d(tmpidx)
          DO i = j - 1, l, -1
            IF (stringlexcomp(arr1d(idx1d(i)), a, sflag) <= 0) EXIT
            idx1d(i + 1) = idx1d(i)
          END DO
          idx1d(i + 1) = tmpidx
        END DO

        IF (ist == 0) THEN
          CALL unsetmessagesource()

```

```
          RETURN
        END IF

        r   = stack(ist)
        l   = stack(ist - 1)
        ist = ist - 2
      ELSE
        k = (l + r) / 2

        CALL swap(idx1d(k), idx1d(l + 1))
        CALL icompxchg(idx1d(l), idx1d(r))
        CALL icompxchg(idx1d(l + 1), idx1d(r))
        CALL icompxchg(idx1d(l), idx1d(l + 1))

        i = l + 1
        j = r
        tmpidx = idx1d(l + 1)
        a = arr1d(tmpidx)

        DO
          DO
            i = i + 1
            IF (stringlexcomp(arr1d(idx1d(i)), a, sflag) > 0) EXIT
          END DO

          DO
            j = j - 1
            IF (stringlexcomp(arr1d(idx1d(j)), a, sflag) < 0) EXIT
          END DO

          IF (j < i) EXIT
          CALL swap(idx1d(i), idx1d(j))
        END DO

        idx1d(l + 1) = idx1d(j)
        idx1d(j) = tmpidx
        ist = ist + 2

        IF (ist > nstack) THEN
          WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
          WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                       trim(adjustl(tmpstr1))

          CALL logmessage(error, scratchmessage)
          CALL unsetmessagesource()

          IF (PRESENT(status)) status = 2

          RETURN

        END IF

        IF (r - i + 1 >= j - l) THEN
          stack(ist) = r
          stack(ist - 1) = i
          r = j - 1
        ELSE
          stack(ist) = j - 1
          stack(ist - 1) = l
          l = i
        END IF
      END IF
    END DO

    CALL unsetmessagesource()


    CONTAINS

    SUBROUTINE icompxchg(i, j)

      IMPLICIT NONE

      ! Global variables
      INTEGER, INTENT(INOUT) :: i, j

      ! Local variables
      INTEGER :: swp

      IF (stringlexcomp(arr1d(j), arr1d(i), sflag) < 0) THEN
        swp = i
```

```
      i   = j
      j   = swp
    END IF

  END SUBROUTINE icompxchg

END SUBROUTINE indexxstring

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   I N D E X X  S I N G L E
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE indexxsingle(arr1D, idx1D, status)

  IMPLICIT NONE

  ! Global variables
  REAL(SP), DIMENSION(:), INTENT(IN) :: arr1D
  INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
  INTEGER, OPTIONAL, INTENT(OUT)     :: status

  ! Local variables
  INTEGER, PARAMETER                 :: NN = 15, nstack = 50
  REAL(SP)                           :: a
  INTEGER                            :: nARR, nIDX, tmpIDX
  INTEGER                            :: k, i, j, l, r
  INTEGER                            :: ist, stack(NSTACK)
  CHARACTER(LEN=64)                  :: tmpStr1, tmpStr2


  CALL setmessagesource("IndexxSingle")

  IF (PRESENT(status)) status = 0

  narr = SIZE(arr1d, 1)
  nidx = SIZE(idx1d, 1)

  IF (narr /= nidx) THEN
    WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
    WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
    WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
                                 trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

    CALL logmessage(error, scratchmessage)
    CALL unsetmessagesource()

    IF (PRESENT(status)) status = 1

    RETURN
  END IF

  idx1d = arth(1, 1, narr)

  ist = 0
  l   = 1
  r   = narr

  DO
    IF (r - l < nn) THEN
      DO j = l + 1, r
        tmpidx = idx1d(j)
        a = arr1d(tmpidx)
        DO i = j - 1, l, -1
          IF (arr1d(idx1d(i)) <= a) EXIT
          idx1d(i + 1) = idx1d(i)
        END DO
        idx1d(i + 1) = tmpidx
      END DO

      IF (ist == 0) THEN
        CALL unsetmessagesource()

        RETURN
      END IF

      r   = stack(ist)
      l   = stack(ist - 1)
      ist = ist - 2
    ELSE
```

```
      k = (l + r) / 2

      CALL swap(idx1d(k), idx1d(l + 1))
      CALL icompxchg(idx1d(l), idx1d(r))
      CALL icompxchg(idx1d(l + 1), idx1d(r))
      CALL icompxchg(idx1d(l), idx1d(l + 1))

      i = l + 1
      j = r
      tmpidx = idx1d(l + 1)
      a = arr1d(tmpidx)

      DO
        DO
          i = i + 1
          IF (arr1d(idx1d(i)) > a) EXIT
        END DO

        DO
          j = j - 1
          IF (arr1d(idx1d(j)) < a) EXIT
        END DO

        IF (j < i) EXIT
        CALL swap(idx1d(i), idx1d(j))
      END DO

      idx1d(l + 1) = idx1d(j)
      idx1d(j) = tmpidx
      ist = ist + 2

      IF (ist > nstack) THEN
        WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
        WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                     trim(adjustl(tmpstr1))

        CALL logmessage(error, scratchmessage)
        CALL unsetmessagesource()

        IF (PRESENT(status)) status = 2

        RETURN

      END IF

      IF (r - i + 1 >= j - l) THEN
        stack(ist) = r
        stack(ist - 1) = i
        r = j - 1
      ELSE
        stack(ist) = j - 1
        stack(ist - 1) = l
        l = i
      END IF
    END IF
  END DO

  CALL unsetmessagesource()


  CONTAINS

  SUBROUTINE icompxchg(i, j)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(INOUT) :: i, j

    ! Local variables
    INTEGER :: swp

    IF (arr1d(j) < arr1d(i)) THEN
      swp = i
      i   = j
      j   = swp
    END IF

  END SUBROUTINE icompxchg

END SUBROUTINE indexxsingle
```

```
!==============================================================================

!------------------------------------------------------------------
! S U B R O U T I N E   I N D E X X  D O U B L E
!------------------------------------------------------------------
!------------------------------------------------------------------
SUBROUTINE indexxdouble(arr1D, idx1D, status)

  IMPLICIT NONE

  ! Global variables
  REAL(HP), DIMENSION(:), INTENT(IN) :: arr1D
  INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
  INTEGER, OPTIONAL, INTENT(OUT)     :: status

  ! Local variables
  INTEGER, PARAMETER                 :: NN = 15, nstack = 50
  REAL(HP)                           :: a
  INTEGER                            :: nARR, nIDX, tmpIDX
  INTEGER                            :: k, i, j, l, r
  INTEGER                            :: ist, stack(NSTACK)
  CHARACTER(LEN=64)                  :: tmpStr1, tmpStr2


  CALL setmessagesource("IndexxDouble")

  IF (PRESENT(status)) status = 0

  narr = SIZE(arr1d, 1)
  nidx = SIZE(idx1d, 1)

  IF (narr /= nidx) THEN
    WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
    WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
    WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
                                 trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

    CALL logmessage(error, scratchmessage)
    CALL unsetmessagesource()

    IF (PRESENT(status)) status = 1

    RETURN
  END IF

  idx1d = arth(1, 1, narr)

  ist = 0
  l   = 1
  r   = narr

  DO
    IF (r - l < nn) THEN
      DO j = l + 1, r
        tmpidx = idx1d(j)
        a = arr1d(tmpidx)
        DO i = j - 1, l, -1
          IF (arr1d(idx1d(i)) <= a) EXIT
          idx1d(i + 1) = idx1d(i)
        END DO
        idx1d(i + 1) = tmpidx
      END DO

      IF (ist == 0) THEN
        CALL unsetmessagesource()

        RETURN
      END IF

      r   = stack(ist)
      l   = stack(ist - 1)
      ist = ist - 2
    ELSE
      k = (l + r) / 2

      CALL swap(idx1d(k), idx1d(l + 1))
      CALL icompxchg(idx1d(l), idx1d(r))
      CALL icompxchg(idx1d(l + 1), idx1d(r))
      CALL icompxchg(idx1d(l), idx1d(l + 1))

```

```
      i = l + 1
      j = r
      tmpidx = idx1d(l + 1)
      a = arr1d(tmpidx)

      DO
        DO
          i = i + 1
          IF (arr1d(idx1d(i)) > a) EXIT
        END DO

        DO
          j = j - 1
          IF (arr1d(idx1d(j)) < a) EXIT
        END DO

        IF (j < i) EXIT
        CALL swap(idx1d(i), idx1d(j))
      END DO

      idx1d(l + 1) = idx1d(j)
      idx1d(j) = tmpidx
      ist = ist + 2

      IF (ist > nstack) THEN
        WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
        WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                     trim(adjustl(tmpstr1))

        CALL logmessage(error, scratchmessage)
        CALL unsetmessagesource()

        IF (PRESENT(status)) status = 2

        RETURN

      END IF

      IF (r - i + 1 >= j - l) THEN
        stack(ist) = r
        stack(ist - 1) = i
        r = j - 1
      ELSE
        stack(ist) = j - 1
        stack(ist - 1) = l
        l = i
      END IF
    END IF
  END DO

  CALL unsetmessagesource()


  CONTAINS

  SUBROUTINE icompxchg(i, j)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(INOUT) :: i, j

    ! Local variables
    INTEGER :: swp

    IF (arr1d(j) < arr1d(i)) THEN
      swp = i
      i   = j
      j   = swp
    END IF

  END SUBROUTINE icompxchg

END SUBROUTINE indexxdouble

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   Q U I C K   S O R T
!----------------------------------------------------------------
!----------------------------------------------------------------
```

```fortran
SUBROUTINE quicksort(arr1D, status)

  IMPLICIT NONE

  ! Global variables
  REAL(SZ), DIMENSION(:), INTENT(INOUT) :: arr1D
  INTEGER, OPTIONAL, INTENT(OUT)        :: status

  ! Local variables
  INTEGER, PARAMETER                    :: NN = 15, nstack = 50
  REAL(SZ)                              :: a
  INTEGER                               :: nARR
  INTEGER                               :: k, i, j, l, r
  INTEGER                               :: ist, stack(NSTACK)
  CHARACTER(LEN=64)                     :: tmpStr1


  CALL setmessagesource("QuickSort")

  IF (PRESENT(status)) status = 0

  narr = size(arr1d, 1)

  ist = 0
  l   = 1
  r   = narr

  DO
    ! Insertion sort when subarray small enough
    IF (r - l < nn) THEN
      DO j = l + 1, r
        a = arr1d(j)
        DO i = j - 1, l, -1
          IF (arr1d(i) <= a) EXIT
          arr1d(i + 1) = arr1d(i)
        END DO
        arr1d(i + 1) = a
      END DO

      IF (ist == 0) THEN
        CALL unsetmessagesource()

        RETURN
      END IF

      ! Pop stack and begin a new round of partitioning
      r   = stack(ist)
      l   = stack(ist - 1)
      ist = ist - 2

    ! Choose median of left, center, and right elements as partitioning
    ! element a. Also rearrange so that a(l) <= a(l + 1) <= a(r)
    ELSE
      k = (l + r) / 2

      CALL swap(arr1d(k), arr1d(l + 1))
      CALL swap(arr1d(l), arr1d(r), arr1d(l) > arr1d(r))
      CALL swap(arr1d(l + 1), arr1d(r), arr1d(l + 1) > arr1d(r))
      CALL swap(arr1d(l), arr1d(l + 1), arr1d(l) > arr1d(l + 1))

      ! Initialize pointers for partitioning
      i = l + 1
      j = r
      a = arr1d(l + 1) ! Partitioning element.

      DO ! Here is the meat.
        ! Scan up to find element >= a
        DO
          i = i + 1
          IF (arr1d(i) > a) EXIT
        END DO

        ! Scan down to find element <= a
        DO
          j = j - 1
          IF (arr1d(j) < a) EXIT
        END DO

        ! Pointers crossed. Exit with partitioning complete.
        IF (j < i) EXIT

```

```
        CALL swap(arr1d(i), arr1d(j)) !Exchange elements.
      END DO

      ! Insert partitioning element
      arr1d(l + 1) = arr1d(j)
      arr1d(j) = a
      ist = ist + 2

      ! Push pointers to larger subarray on stack; process smaller subarray immediately.
      IF (ist > nstack) THEN
        WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
        WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
                                     trim(adjustl(tmpstr1))

        CALL logmessage(error, scratchmessage)
        CALL unsetmessagesource()

        IF (PRESENT(status)) status = 2

        RETURN

      END IF

      IF (r - i + 1 >= j - l) THEN
        stack(ist)     = r
        stack(ist - 1) = i
        r = j - 1
      ELSE
        stack(ist)     = j - 1
        stack(ist - 1) = l
        l = i
      END IF
    END IF
  END DO

  CALL unsetmessagesource()

END SUBROUTINE quicksort

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   S O R T  2
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE sort2(arr1D, slv1D, status)

  IMPLICIT NONE

  ! Global variables
  REAL(SZ), DIMENSION(:), INTENT(INOUT) :: arr1D, slv1D
  INTEGER, OPTIONAL, INTENT(OUT)        :: status

  ! Local variables
  INTEGER                          :: nARR, nSLV
  INTEGER, DIMENSION(SIZE(arr1D)) :: idx1D
  CHARACTER(LEN=64)                :: tmpStr1, tmpStr2


  CALL setmessagesource("Sort2")

  narr = SIZE(arr1d, 1)
  nslv = SIZE(slv1d, 1)

  IF (narr /= nslv) THEN
    WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
    WRITE(tmpstr2, '(a, i0)') 'nSLV = ', nslv
    WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and slv1D is not the same: ' // &
                                 trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))

    CALL logmessage(error, scratchmessage)
    CALL unsetmessagesource()

    IF (PRESENT(status)) status = 1

    RETURN
  END IF

  ! Make the index array
  CALL indexx(arr1d, idx1d, status)

```

```
  ! Sort the array
  arr1d = arr1d(idx1d)

  ! Rearrange slave
  slv1d = slv1d(idx1d)

  CALL unsetmessagesource()

END SUBROUTINE sort2

!================================================================================


!----------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  C O P Y  I N T
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE arraycopyint(src, dest, nCP, nNCP)

  IMPLICIT NONE

  ! Global variables
  INTEGER, DIMENSION(:), INTENT(IN)  :: src
  INTEGER, DIMENSION(:), INTENT(OUT) :: dest
  INTEGER, INTENT(OUT)               :: nCP, nNCP

  ncp  = min(SIZE(src), SIZE(dest))
  nncp = SIZE(src) - ncp
  dest(1:ncp) = src(1:ncp)

END SUBROUTINE arraycopyint

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  C O P Y  S I N G L E
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE arraycopysingle(src, dest, nCP, nNCP)

  IMPLICIT NONE

  ! Global variables
  REAL(SP), DIMENSION(:), INTENT(IN)  :: src
  REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
  INTEGER, INTENT(OUT)                :: nCP, nNCP

  ncp  = min(SIZE(src), SIZE(dest))
  nncp = SIZE(src) - ncp
  dest(1:ncp) = src(1:ncp)

END SUBROUTINE arraycopysingle

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  C O P Y  D O U B L E
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE arraycopydouble(src, dest, nCP, nNCP)

  IMPLICIT NONE

  ! Global variables
  REAL(HP), DIMENSION(:), INTENT(IN)  :: src
  REAL(HP), DIMENSION(:), INTENT(OUT) :: dest
  INTEGER, INTENT(OUT)                :: nCP, nNCP

  ncp  = min(SIZE(src), SIZE(dest))
  nncp = SIZE(src) - ncp
  dest(1:ncp) = src(1:ncp)

END SUBROUTINE arraycopydouble

!================================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  E Q U A L  I N T
!----------------------------------------------------------------
!----------------------------------------------------------------
LOGICAL FUNCTION arrayequalint(arr1, arr2) RESULT(myValOut)
```

```

  IMPLICIT NONE

  ! Global variables
  INTEGER, DIMENSION(:), INTENT(IN) :: arr1, arr2


  IF (SIZE(arr1) /= SIZE(arr2)) THEN
    myvalout = .false.

    RETURN
  END IF

  myvalout = .true.
  IF (any(arr1 - arr2 /= 0)) myvalout = .false.

  RETURN

END FUNCTION arrayequalint

!==============================================================================

!-----------------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  E Q U A L  S I N G L E
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
LOGICAL FUNCTION arrayequalsingle(arr1, arr2) RESULT(myValOut)

  IMPLICIT NONE

  ! Global variables
  REAL(sp), DIMENSION(:), INTENT(IN) :: arr1, arr2

  ! Local variables
  INTEGER :: i


  IF (SIZE(arr1) /= SIZE(arr2)) THEN
    myvalout = .false.

    RETURN
  END IF

  myvalout = .true.

  DO i = 1, SIZE(arr1, 1)
    IF (comparereals(arr1(i), arr2(i), 0.00000001_sp) /= 0) THEN
      myvalout = .false.

      EXIT
    END IF
  END DO

  RETURN

END FUNCTION arrayequalsingle

!==============================================================================

!-----------------------------------------------------------------------
! S U B R O U T I N E   A R R A Y  E Q U A L  S I N G L E
!-----------------------------------------------------------------------
!-----------------------------------------------------------------------
LOGICAL FUNCTION arrayequaldouble(arr1, arr2) RESULT(myValOut)

  IMPLICIT NONE

  ! Global variables
  REAL(hp), DIMENSION(:), INTENT(IN) :: arr1, arr2

  ! Local variables
  INTEGER :: i


  IF (SIZE(arr1) /= SIZE(arr2)) THEN
    myvalout = .false.

    RETURN
  END IF

  myvalout = .true.
```

```
  DO i = 1, SIZE(arr1, 1)
    IF (comparereals(arr1(i), arr2(i), 0.0000000000001_hp) /= 0) THEN
      myvalout = .false.

      EXIT
    END IF
  END DO

  RETURN

END FUNCTION arrayequaldouble

!==============================================================================

!----------------------------------------------------------------
! F U N C T I O N   S T R I N G  L E X  C O M P
!----------------------------------------------------------------
!----------------------------------------------------------------
INTEGER FUNCTION stringlexcomp(str1, str2, mSensitive) RESULT(myValOut)

  USE utilities, ONLY : touppercase

  IMPLICIT NONE

  ! Global variables
  CHARACTER(LEN=*), INTENT(IN)  :: str1, str2
  LOGICAL, OPTIONAL, INTENT(IN) :: msensitive

  ! Local variables
  LOGICAL :: sflag

  sflag = .true.
  IF (PRESENT(msensitive)) sflag = msensitive

  IF (sflag) THEN
    IF (trim(str1) == trim(str2)) THEN
      myvalout = 0
    ELSE IF (trim(str1) < trim(str2)) THEN
      myvalout = -1
    ELSE
      myvalout = 1
    END IF
  ELSE
    IF (touppercase(trim(str1)) == touppercase(trim(str2))) THEN
      myvalout = 0
    ELSE IF (touppercase(trim(str1)) < touppercase(trim(str2))) THEN
      myvalout = -1
    ELSE
      myvalout = 1
    END IF
  END IF

  RETURN

END FUNCTION stringlexcomp

!==============================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   S W A P  I N T
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE swapint(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  INTEGER, INTENT(INOUT)        :: a, b
  LOGICAL, OPTIONAL, INTENT(IN) :: mask

  ! Local variables
  INTEGER :: dum
  LOGICAL :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
```

```
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapint

!==============================================================================

!--------------------------------------------------------------
! S U B R O U T I N E   S W A P  S I N G L E
!--------------------------------------------------------------
!--------------------------------------------------------------
SUBROUTINE swapsingle(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  REAL(SP), INTENT(INOUT)        :: a, b
  LOGICAL, OPTIONAL, INTENT(IN)  :: mask

  ! Local variables
  REAL(SP) :: dum
  LOGICAL  :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapsingle

!==============================================================================

!--------------------------------------------------------------
! S U B R O U T I N E   S W A P  D O U B L E
!--------------------------------------------------------------
!--------------------------------------------------------------
SUBROUTINE swapdouble(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  REAL(HP), INTENT(INOUT)        :: a, b
  LOGICAL, OPTIONAL, INTENT(IN)  :: mask

  ! Local variables
  REAL(HP) :: dum
  LOGICAL  :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapdouble

!==============================================================================

!--------------------------------------------------------------
! S U B R O U T I N E   S W A P  I N T  V E C
!--------------------------------------------------------------
!--------------------------------------------------------------
SUBROUTINE swapintvec(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  INTEGER, DIMENSION(:), INTENT(INOUT) :: a, b
  LOGICAL, OPTIONAL, INTENT(IN)        :: mask

  ! Local variables
```

```
  INTEGER, DIMENSION(SIZE(a)) :: dum
  LOGICAL                     :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapintvec

!==============================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   S W A P  S I N G L E  V E C
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE swapsinglevec(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  REAL(SP), DIMENSION(:), INTENT(INOUT) :: a, b
  LOGICAL, OPTIONAL, INTENT(IN)         :: mask

  ! Local variables
  REAL(SP), DIMENSION(SIZE(a)) :: dum
  LOGICAL                      :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapsinglevec

!==============================================================================

!----------------------------------------------------------------
! S U B R O U T I N E   S W A P  D O U B L E  V E C
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE swapdoublevec(a, b, mask)

  IMPLICIT NONE

  ! Global variables
  REAL(HP), DIMENSION(:), INTENT(INOUT) :: a, b
  LOGICAL, OPTIONAL, INTENT(IN)         :: mask

  ! Local variables
  REAL(HP), DIMENSION(SIZE(a)) :: dum
  LOGICAL                      :: mFlag


  mflag = .true.
  IF (PRESENT(mask)) mflag = mask

  IF (mflag) THEN
    dum = a
    a   = b
    b   = dum
  END IF

END SUBROUTINE swapdoublevec

!==============================================================================

 !----------------------------------------------------------------
! S U B R O U T I N E   A R T H  I N T
!----------------------------------------------------------------
!----------------------------------------------------------------
```

```
pure FUNCTION arthint(first, increment, n) RESULT(arthOut)

  IMPLICIT NONE

  ! Global variables
  INTEGER, INTENT(IN)   :: first, increment
  INTEGER, INTENT(IN)   :: n
  INTEGER, DIMENSION(n) :: arthout

  ! Local variables
  INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
  INTEGER :: k, k2
  INTEGER :: temp


  IF (n > 0) arthout(1) = first

  IF (n <= nparth) THEN
    DO k = 2, n
      arthout(k) = arthout(k - 1) + increment
    END DO
  ELSE
    DO k = 2, nparth2
      arthout(k) = arthout(k - 1) + increment
    END DO

    temp = increment * nparth2
    k = nparth2

    DO
      IF (k >= n) EXIT
      k2 = k + k
      arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
      temp = temp + temp
      k = k2
    END DO
  END IF

  RETURN

END FUNCTION arthint

!==============================================================================

!-----------------------------------------------------------------
! S U B R O U T I N E   A R T H  S I N G L E
!-----------------------------------------------------------------
!-----------------------------------------------------------------
pure FUNCTION arthsingle(first, increment, n) RESULT(arthOut)

  IMPLICIT NONE

  ! Global variables
  REAL(sp), INTENT(IN)   :: first, increment
  INTEGER, INTENT(IN)    :: n
  REAL(sp), DIMENSION(n) :: arthout

  ! Local variables
  INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
  INTEGER  :: k, k2
  REAL(sp) :: temp


  IF (n > 0) arthout(1) = first

  IF (n <= nparth) THEN
    DO k = 2, n
      arthout(k) = arthout(k - 1) + increment
    END DO
  ELSE
    DO k = 2, nparth2
      arthout(k) = arthout(k - 1) + increment
    END DO

    temp = increment * nparth2
    k = nparth2

    DO
      IF (k >= n) EXIT
      k2 = k + k
      arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
```

```
        temp = temp + temp
        k = k2
      END DO
    END IF

    RETURN

  END FUNCTION arthsingle

  !===============================================================================

  !-----------------------------------------------------------------
  ! S U B R O U T I N E   A R T H  D O U B L E
  !-----------------------------------------------------------------
  !-----------------------------------------------------------------
  pure FUNCTION arthdouble(first, increment, n) RESULT(arthOut)

    IMPLICIT NONE

    ! Global variables
    REAL(hp), INTENT(IN)   :: first, increment
    INTEGER, INTENT(IN)    :: n
    REAL(hp), DIMENSION(n) :: arthout

    ! Local variables
    INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
    INTEGER  :: k, k2
    REAL(hp) :: temp


    IF (n > 0) arthout(1) = first

    IF (n <= nparth) THEN
      DO k = 2, n
        arthout(k) = arthout(k - 1) + increment
      END DO
    ELSE
      DO k = 2, nparth2
        arthout(k) = arthout(k - 1) + increment
      END DO

      temp = increment * nparth2
      k = nparth2

      DO
        IF (k >= n) EXIT
        k2 = k + k
        arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
        temp = temp + temp
        k = k2
      END DO
    END IF

    RETURN

  END FUNCTION arthdouble

  !===============================================================================

END MODULE sortutils
```

9.36 timedateutils.F90 File Reference

Data Types

- interface timedateutils::timeconv
- interface timedateutils::gregtojulday
- interface timedateutils::splitdatetimestring

Modules

- module timedateutils

Functions/Subroutines

- subroutine timedateutils::timeconvisec (iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- subroutine timedateutils::timeconvrsec (iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)

  Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- logical function timedateutils::leapyear (iYear)

  Checks for a leap year.
- integer function timedateutils::yeardays (iYear)

  Determines the days of the year.
- integer function timedateutils::monthdays (iYear, iMonth)

  Determines the days in the month of the year.
- integer function timedateutils::dayofyear (iYear, iMonth, iDay)

  Determines the day of the year.
- real(sz) function timedateutils::gregtojuldayisec (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

  Determines the Julian date from a Gregorian date.
- real(sz) function timedateutils::gregtojuldayrsec (iYear, iMonth, iDay, iHour, iMin, rSec, mJD)

  Determines the Julian date from a Gregorian date.
- real(sz) function timedateutils::gregtojulday2 (iDate, iTime, mJD)

  Determines the Julian date from a Gregorian date.
- subroutine timedateutils::juldaytogreg (julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

  Determines the Julian date from a Gregorian date.
- subroutine timedateutils::dayofyeartogreg (inYR, inDY, iYear, iMonth, iDay)

  Determines the Gregorian date (year, month, day) from a day of the year.
- subroutine timedateutils::splitdatetimestring (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)

  Splits a date string into components.
- subroutine timedateutils::splitdatetimestring2 (inDateTime, iDate, iTime)

  Splits a date string into two components.
- character(len=len(indatetime)) function timedateutils::preprocessdatetimestring (inDateTime)

  Pre-processes an arbitrary date string.
- integer function timedateutils::joindate (iYear, iMonth, iDay)

  Pre-processes an arbitrary date string.
- subroutine timedateutils::splitdate (inDate, iYear, iMonth, iDay)

  Pre-processes an arbitrary date string.
- character(len=64) function timedateutils::datetime2string (year, month, day, hour, min, sec, sep, units, zone, err)

  Constructs a NetCDF time string.
- real(sz) function timedateutils::gettimeconvsec (units, invert)

  Calculates the conversion factor between time units and seconds.
- real(sz) function timedateutils::elapsedsecs (inTime1, inTime2, inUnits)

  Calculates the elapsed time in seconds.

Variables

- integer, parameter timedateutils::firstgregdate = 1582 ∗ 10000 + 10 ∗ 100 + 05
- integer, parameter timedateutils::firstgregtime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter timedateutils::offfirstgregday = 2299150.5_HP
- integer, parameter timedateutils::modjuldate = 1858 ∗ 10000 + 11 ∗ 100 + 17
- integer, parameter timedateutils::modjultime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter timedateutils::offmodjulday = 2400000.5_HP
- integer, parameter timedateutils::unixdate = 1970 ∗ 10000 + 1 ∗ 100 + 1
- integer, parameter timedateutils::unixtime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter timedateutils::offunixjulday = 2440587.5_HP
- integer, parameter timedateutils::modeldate = 1990 ∗ 10000 + 1 ∗ 100 + 1
- integer, parameter timedateutils::modeltime = 0 ∗ 10000 + 0 ∗ 100 + 0
- real(hp), parameter timedateutils::offmodeljulday = 2447892.5_HP
- integer, parameter timedateutils::usemodjulday = 0
- integer, parameter timedateutils::mdjdate = UNIXDATE
- integer, parameter timedateutils::mdjtime = UNIXTIME
- real(hp), parameter timedateutils::mdjoffset = OFFUNIXJULDAY

9.36.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file timedateutils.F90.

9.37 timedateutils.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!               M O D U L E   T I M E  D A T E  U T I L S
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE timedateutils

  USE pahm_sizes
  USE pahm_messages

  PRIVATE :: upp

  !----------------------------------------------------------------------
  ! I N T E R F A C E S
  !----------------------------------------------------------------------
  INTERFACE timeconv
    MODULE PROCEDURE timeconvisec
    MODULE PROCEDURE timeconvrsec
  END INTERFACE timeconv

  INTERFACE gregtojulday
    MODULE PROCEDURE gregtojuldayisec
    MODULE PROCEDURE gregtojuldayrsec
    MODULE PROCEDURE gregtojulday2
  END INTERFACE gregtojulday

  INTERFACE splitdatetimestring
    MODULE PROCEDURE splitdatetimestring
    MODULE PROCEDURE splitdatetimestring2
```

```
END INTERFACE splitdatetimestring
!-----------------------------------------------------------------------

! Julian day number for the first date of the Gregorian calendar (10/05/1582).
INTEGER, PARAMETER  :: firstgregdate   = 1582 * 10000 + 10 * 100 + 05
INTEGER, PARAMETER  :: firstgregtime   = 0 * 10000 + 0 * 100 + 0
REAL(hp), PARAMETER :: offfirstgregday = 2299150.5_hp

! A modified version of the Julian date denoted MJD obtained by subtracting
! 2,400,000.5 days from the Julian date JD, The MJD therefore gives the number
! of days since midnight of November 17, 1858. This date corresponds to
! 2400000.5 days after day 0 of the Julian calendar
! (https://scienceworld.wolfram.com/astronomy/ModifiedJulianDate.html).
INTEGER, PARAMETER  :: modjuldate    = 1858 * 10000 + 11 * 100 + 17
INTEGER, PARAMETER  :: modjultime    = 0 * 10000 + 0 * 100 + 0
REAL(hp), PARAMETER :: offmodjulday = 2400000.5_hp

! Julian day number for the first date of Unix time. This MJD gives the number
! of days since midnight of January 1, 1970.
INTEGER, PARAMETER  :: unixdate      = 1970 * 10000 + 1 * 100 + 1
INTEGER, PARAMETER  :: unixtime      = 0 * 10000 + 0 * 100 + 0
REAL(hp), PARAMETER :: offunixjulday = 2440587.5_hp

! Julian day number for the first date of Model time. This MJD gives the number
! of days since midnight of January 1, 1990.
INTEGER, PARAMETER  :: modeldate      = 1990 * 10000 + 1 * 100 + 1
INTEGER, PARAMETER  :: modeltime      = 0 * 10000 + 0 * 100 + 0
REAL(hp), PARAMETER :: offmodeljulday = 2447892.5_hp

!-------------------- MOD JUL DAY
! Definitions to use or not modified julian day calculations
! If USEMODJULDAY >= 1 use MJD calculation
INTEGER, PARAMETER  :: usemodjulday = 0
!--- First option for a modified julian day
!INTEGER, PARAMETER  :: MDJDATE   = MODJULDATE
!INTEGER, PARAMETER  :: MDJTIME   = MODJULDATE
!REAL(HP), PARAMETER :: MDJOFFSET = OFFMODJULDAY
!---

!--- Second option for a modified julian day
INTEGER, PARAMETER  :: mdjdate   = unixdate
INTEGER, PARAMETER  :: mdjtime   = unixtime
REAL(hp), PARAMETER :: mdjoffset = offunixjulday

!--- Third option for a modified julian day
!INTEGER, PARAMETER  :: MDJDATE   = MODELDATE
!INTEGER, PARAMETER  :: MDJTIME   = MODELTIME
!REAL(HP), PARAMETER :: MDJOFFSET = OFFMODELJULDAY
!---
!--------------------


CONTAINS


!----------------------------------------------------------------
! S U B R O U T I N E   T I M E  C O N V  I S E C
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE timeconvisec(iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)

  USE pahm_global, ONLY : refyear, refmonth, refday, refhour, refmin, refsec

  IMPLICIT NONE

  ! Global variables
  INTEGER, INTENT(IN)   :: iYear, iMonth, iDay, iHour, iMin, iSec
  REAL(SZ), INTENT(OUT) :: timeSec

  ! Local variables
  REAL(SZ)          :: jd0, jd1
  CHARACTER(LEN=64) :: tmpStr1, tmpStr2

  !----- START CALCULATIONS -----

  CALL setmessagesource("TimeConv")

  jd0 = gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
  jd1 = gregtojulday(iyear, imonth, iday, ihour, imin, isec)

  IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
```

```
      timesec = rmissv

      WRITE(tmpstr1, '(f20.3)') jd0
      WRITE(tmpstr2, '(f20.3)') jd1
      WRITE(scratchmessage, '(a)') 'Invalid julian dates calculated: refJD = ' // &
                           trim(adjustl(tmpstr1)) // ', inpJD = ' // trim(adjustl(tmpstr2))

      CALL allmessage(error, scratchmessage)
      CALL unsetmessagesource()

      CALL terminate()
    END IF

    timesec = elapsedsecs(jd0, jd1, 'days')

    CALL unsetmessagesource()

    RETURN

  END SUBROUTINE timeconvisec

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   T I M E   C O N V   R S E C
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE timeconvrsec(iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)

    USE pahm_global, ONLY : refyear, refmonth, refday, refhour, refmin, refsec

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN)    :: iYear, iMonth, iDay, iHour, iMin
    REAL(SZ), INTENT(IN)   :: rSec
    REAL(SZ), INTENT(OUT) :: timeSec

    ! Local variables
    REAL(SZ)           :: jd0, jd1
    CHARACTER(LEN=64) :: tmpStr1, tmpStr2

    !----- START CALCULATIONS -----

    CALL setmessagesource("TimeConv")

    jd0 = gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
    jd1 = gregtojulday(iyear, imonth, iday, ihour, imin, rsec)

    IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
      timesec = rmissv

      WRITE(tmpstr1, '(f20.3)') jd0
      WRITE(tmpstr2, '(f20.3)') jd1
      WRITE(scratchmessage, '(a)') 'Invalid julian dates calculated: refJD = ' // &
                           trim(adjustl(tmpstr1)) // ', inpJD = ' // trim(adjustl(tmpstr2))

      CALL allmessage(error, scratchmessage)
      CALL unsetmessagesource()

      CALL terminate()
    END IF

    timesec = elapsedsecs(jd0, jd1, 'days')

    CALL unsetmessagesource()

    RETURN

  END SUBROUTINE timeconvrsec

!================================================================================

!DEL  !----------------------------------------------------------------
!DEL ! S U B R O U T I N E   T I M E   C O N V   A D C I R C <- TO BE DELETED
!DEL !----------------------------------------------------------------
!DEL !----------------------------------------------------------------
!DEL SUBROUTINE TimeConvADCIRC(year, month, day, hour, minute, sec, timeSec)

!DEL   IMPLICIT NONE

```

```
!DEL   INTEGER  :: year, month, day, hour, minute, leap
!DEL   REAL(SZ) :: timeSec, sec, secPerDay, secPerHour, secPerMin

!DEL   !----- START CALCULATIONS -----

!DEL   secPerDay  = 86400_SZ
!DEL   secPerHour =  3600.0_SZ
!DEL   secPerMin  =    60.0_SZ

!DEL   CALL SetMessageSource("TimeConv")

!DEL   timeSec = (day - 1) * secPerDay + hour * secPerHour + minute * secPerMin + sec
!DEL   IF (month >= 2) timeSec = timeSec + 31 * secPerDay

!DEL   leap = (year / 4) * 4
!DEL   IF ((leap == year) .AND. (month >= 3)) timeSec = timeSec + 29 * secPerDay
!DEL   IF ((leap /= year) .AND. (month >= 3)) timeSec = timeSec + 28 * secPerDay

!DEL   IF (month >= 4)  timeSec = timeSec + 31 * secPerDay
!DEL   IF (month >= 5)  timeSec = timeSec + 30 * secPerDay
!DEL   IF (month >= 6)  timeSec = timeSec + 31 * secPerDay
!DEL   IF (month >= 7)  timeSec = timeSec + 30 * secPerDay
!DEL   IF (month >= 8)  timeSec = timeSec + 31 * secPerDay
!DEL   IF (month >= 9)  timeSec = timeSec + 31 * secPerDay
!DEL   IF (month >= 10) timeSec = timeSec + 30 * secPerDay
!DEL   IF (month >= 11) timeSec = timeSec + 31 * secPerDay
!DEL   IF (month == 12) timeSec = timeSec + 30 * secPerDay

!DEL   IF (month > 12) THEN
!DEL     CALL AllMessage(ERROR, 'Fatal error in subroutine TimeConv: month > 12.')
!DEL     CALL Terminate()
!DEL   END IF

!DEL   CALL UnsetMessageSource()

!DEL   RETURN

!DEL END SUBROUTINE TimeConvADCIRC

!DEL================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   L E A P  Y E A R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  LOGICAL FUNCTION leapyear(iYear) RESULT(myVal)

    IMPLICIT NONE

    INTEGER, INTENT(IN) :: iyear

    !----- START CALCULATIONS -----

    IF (iyear < 1582) Then
      myval = .false.

      RETURN
    END IF

    ! ADCIRC uses the construct leap = (iYear / 4) * 4 == iYear
    ! to determine if a year is a leap year. This produces wrong
    ! results, example while 1700, 1900, 2100 are not leap years,
    ! the above construct determines that these years are leap years.
    ! Needs to be fixed.

    IF ((mod(iyear, 100) /= 0) .AND. (mod(iyear, 4) == 0)) THEN
      myval = .true.
    ELSE IF (mod(iyear, 400) == 0) THEN
      myval = .true.
    ELSE
      myval = .false.
    END IF

    RETURN
  END FUNCTION leapyear

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   Y E A R  D A Y S
  !----------------------------------------------------------------
```

```
  !-------------------------------------------------------------------
  INTEGER FUNCTION yeardays(iYear) RESULT(myVal)

    IMPLICIT NONE

    INTEGER, INTENT(IN) :: iyear

    !----- START CALCULATIONS -----

    myval = 365
    IF (leapyear(iyear)) myval = 366

    RETURN
  END FUNCTION yeardays

!================================================================================

  !-------------------------------------------------------------------
  ! F U N C T I O N   M O N T H  D A Y S
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  INTEGER FUNCTION monthdays(iYear, iMonth) RESULT(myVal)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN) :: iyear, imonth

    ! Local variables
    INTEGER :: leap, monlen(12, 2)

    !----- START CALCULATIONS -----

    IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12)) THEN
      myval = imissv

      RETURN
    END IF

    ! Initialize lenghts of months:
    monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,     &
                        31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /),  &
                     (/ 12, 2 /))

    leap = 1
    IF (leapyear(iyear)) leap = 2

    myval = monlen(imonth, leap)

    RETURN
  END FUNCTION monthdays

!================================================================================

  !-------------------------------------------------------------------
  ! F U N C T I O N   D A Y  O F  Y E A R
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  INTEGER FUNCTION dayofyear(iYear, iMonth, iDay) RESULT(myVal)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN) :: iyear, imonth, iday

    ! Local variables
    REAL(sz) :: jd0, jd1

    !----- START CALCULATIONS -----

    jd0 = gregtojulday(iyear, 1, 1, 0, 0, 0)
    jd1 = gregtojulday(iyear, imonth, iday, 0, 0, 0)

    IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
      myval = imissv

      RETURN
    END IF

    myval = int(jd1 - jd0 + 1.0_sz)

```

```
    RETURN
  END FUNCTION dayofyear

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G R E G  T O  J U L  D A Y  I S E C
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sz) function gregtojuldayisec(iyear, imonth, iday, ihour, imin, isec, mjd) result(myval)

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN)           :: iyear, imonth, iday, ihour, imin, isec
    INTEGER, OPTIONAL, INTENT(IN) :: mjd

    ! Local variables
    INTEGER  :: leap, monlen(12, 2)
    LOGICAL  :: modjul
    REAL(hp) :: temp1, temp2

    !----- START CALCULATIONS -----

    modjul = .false.
    IF (PRESENT(mjd)) THEN
      modjul = (mjd > 0)
    ELSE
      modjul = (usemodjulday > 0)
    END IF

    ! Initialize lenghts of months:
    monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,     &
                        31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /),  &
                     (/ 12, 2 /))

    ! This function intentionally works on Gregorian dates only. For modeling
    ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
    ! it is not necessary to go beyond that date.

    ! Is this a LEAP year?
    leap = 1
    IF (leapyear(iyear)) leap = 2

    IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
      myval = rmissv

      RETURN
    ELSE IF ((imonth < 1) .OR. (imonth > 12)                      .OR.   &
             (iday   < 1) .OR. (iday   > monlen(imonth, leap)) .OR.   &
             (ihour  < 0) .OR. (ihour  > 23)                      .OR.   &
             (imin   < 0) .OR. (imin   > 59)                      .OR.   &
             (isec   < 0) .OR. (isec   > 60)) THEN
      myval = rmissv

      RETURN
    ELSE
      temp1 = int((imonth - 14.0_hp) / 12.0_hp)
      temp2 =   iday - 32075.0_hp                                        &
              + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp)        &
              + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp) &
              - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
      temp1 =   real(ihour, hp) * 3600.0_hp &
              + real(imin, hp) * 60.0_hp    &
              + real(isec, hp) - 43200.0_hp

      IF (modjul) THEN
      print *, 'we are using mod jul with MDJOFFSET = ', mdjoffset
        myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
      ELSE
        myval = temp2 + (temp1 / 86400.0_hp)
      END IF
    END IF

    RETURN
  END FUNCTION gregtojuldayisec

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G R E G  T O  J U L  D A Y  R S E C
```

```
!---------------------------------------------------------------
!---------------------------------------------------------------
REAL(sz) function gregtojuldayrsec(iyear, imonth, iday, ihour, imin, rsec, mjd) result(myval)

  IMPLICIT NONE

  ! Global variables
  INTEGER, INTENT(IN)           :: iyear, imonth, iday, ihour, imin
  REAL(sz), INTENT(IN)          :: rsec
  INTEGER, OPTIONAL, INTENT(IN) :: mjd

  ! Local variables
  INTEGER  :: leap, monlen(12, 2)
  LOGICAL  :: modjul
  REAL(hp) :: temp1, temp2

  !----- START CALCULATIONS -----

  modjul = .false.
  IF (PRESENT(mjd)) THEN
    modjul = (mjd > 0)
  ELSE
    modjul = (usemodjulday > 0)
  END IF

  ! Initialize lenghts of months:
  monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,     &
                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /),  &
                   (/ 12, 2 /))

  ! This function intentionally works on Gregorian dates only. For modeling
  ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
  ! it is not necessary to go beyond that date.

  ! Is this a LEAP year?
  leap = 1
  IF (leapyear(iyear)) leap = 2

  IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
    myval = rmissv

    RETURN
  ELSE IF ((imonth < 1) .OR. (imonth > 12)                    .OR.   &
           (iday   < 1) .OR. (iday   > monlen(imonth, leap)) .OR.   &
           (ihour  < 0) .OR. (ihour  > 23)                    .OR.   &
           (imin   < 0) .OR. (imin   > 59)                    .OR.   &
           (rsec   < 0) .OR. (rsec   > 60)) THEN
    myval = rmissv

    RETURN
  ELSE
    temp1 = int((imonth - 14.0_hp) / 12.0_hp)
    temp2 =   iday - 32075.0_hp                                       &
            + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp)         &
            + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp) &
            - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
    temp1 =   real(ihour, hp) * 3600.0_hp &
            + real(imin, hp) * 60.0_hp    &
            + real(rsec, hp) - 43200.0_hp

    IF (modjul) THEN
      myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
    ELSE
      myval = temp2 + (temp1 / 86400.0_hp)
    END IF
  END IF

  RETURN
END FUNCTION gregtojuldayrsec

!================================================================================

!---------------------------------------------------------------
! F U N C T I O N   G R E G  T O  J U L  D A Y  I S E C  2
!---------------------------------------------------------------
!---------------------------------------------------------------
REAL(sz) function gregtojulday2(idate, itime, mjd) result(myval)

  IMPLICIT NONE

  ! Global variables
```

```
    INTEGER, INTENT(IN)           :: idate, itime
    INTEGER, OPTIONAL, INTENT(IN) :: mjd

    ! Local variables
    INTEGER  :: iyear, imonth, iday, ihour, imin, isec
    INTEGER  :: leap, monlen(12, 2)
    LOGICAL  :: modjul
    REAL(hp) :: temp1, temp2

    !----- START CALCULATIONS -----

    modjul = .false.
    IF (PRESENT(mjd)) THEN
      modjul = (mjd > 0)
    ELSE
      modjul = (usemodjulday > 0)
    END IF

    ! Initialize lenghts of months:
    monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,     &
                        31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /),  &
                     (/ 12, 2 /))

    ! This function intentionally works on Gregorian dates only. For modeling
    ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
    ! it is not necessary to go beyond that date.

    ! Is this a LEAP year?
    leap = 1
    IF (leapyear(iyear)) leap = 2

    CALL splitdate(idate, iyear, imonth, iday)
    CALL splitdate(itime, ihour, imin, isec)

    IF ((iyear  < 1582) .OR. (imonth < 1) .OR. (imonth > 12)                    &
                        .OR. (iday   < 1) .OR. (iday   > monlen(imonth, leap))   &
                        .OR. (ihour  < 0) .OR. (ihour  > 23)                    &
                        .OR. (imin   < 0) .OR. (imin   > 59)                    &
                        .OR. (isec   < 0) .OR. (isec   > 60)) THEN
      myval = rmissv

      RETURN
    ELSE
      IF (idate < firstgregdate) THEN
        myval = rmissv

        RETURN
      ELSE
        temp1 = int((imonth - 14.0_hp) / 12.0_hp)
        temp2 =   iday - 32075.0_hp                                                 &
                + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp)             &
                + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp)     &
                - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
        temp1 =   real(ihour, hp) * 3600.0_hp   &
                + real(imin, hp) * 60.0_hp      &
                + real(isec, hp) - 43200.0_hp

        IF (modjul) THEN
          myval   = temp2 + (temp1 / 86400.0_hp) - mdjoffset
        ELSE
          myval   = temp2 + (temp1 / 86400.0_hp)
        END IF
      END IF
    END IF

    RETURN
  END FUNCTION gregtojulday2

!=================================================================================

  !-------------------------------------------------------------------
  ! S U B R O U T I N E   J U L D A Y  T O  G R E G
  !-------------------------------------------------------------------
  !-------------------------------------------------------------------
  SUBROUTINE juldaytogreg(julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)

    IMPLICIT NONE

    ! Global Variables
    REAL(SZ), INTENT(IN)          :: julDay
    INTEGER, OPTIONAL, INTENT(IN) :: mJD
```

```
    INTEGER, INTENT(OUT)          :: iYear, iMonth, iDay, iHour, iMin, iSec

    ! Local Variables
    REAL(HP) :: temp1 , temp2 , temp3 , temp4 , temp5
    REAL(HP) :: thisJulDay, myJulDay, delta
    INTEGER  :: nTry
    LOGICAL  :: modJul

    !----- START CALCULATIONS -----

    modjul = .false.
    IF (PRESENT(mjd)) THEN
      modjul = (mjd > 0)
    ELSE
      modjul = (usemodjulday > 0)
    END IF

    IF (modjul) THEN
      thisjulday = julday + mdjoffset
    ELSE
      thisjulday = julday
    END IF

    ! Check for valid Julian day (Gregorian calendar only)
    IF (thisjulday < offfirstgregday) THEN
      iyear  = imissv
      imonth = imissv
      iday   = imissv
      ihour  = imissv
      imin   = imissv
      isec   = imissv

      RETURN
    END IF

    delta = 0.0_hp
    ntry = 1
    DO WHILE (ntry <= 2)
      myjulday= thisjulday + delta
      temp4 = myjulday
      temp5 = dmod(myjulday, 1.0_hp)

      IF (temp5 < 0.5) THEN
        temp3  = 0.5_hp + temp5
        temp4  = aint(temp4)
      ELSE
        temp3  = temp5 - 0.5_hp
        temp4  = aint(temp4) + 1.0_hp
      END IF

      temp1  = temp4 + 68569.0
      temp2  = aint(4.0_hp * temp1 / 146097.0_hp)
      temp1  = temp1 - aint((146097.0_hp * temp2 + 3.0_hp) / 4.0_hp)
      iyear  = int(4000.0_hp * (temp1 + 1.0_hp) / 1461001.0_hp)
      temp1  = temp1 - aint((1461.0_hp * iyear) / 4.0_hp) + 31.0_hp
      imonth = int(80.0_hp * temp1 / 2447.0_hp)
      iday   = int(temp1 - aint(2447.0_hp * imonth / 80.0_hp))
      temp1  = aint(imonth / 11.0_hp)
      imonth = int(imonth + 2.0 - 12.0_hp * temp1)
      iyear  = int(100.0_hp * (temp2 - 49.0_hp) + iyear + temp1)
      ihour  = int(temp3 * 24.0_hp)
      imin   = int(temp3 * 1440.0_hp - 60.0_hp * ihour)
      isec   = nint(temp3 * 86400.0_hp - 3600.0_hp * ihour - 60.0_hp * imin)

      IF (isec >= 60) THEN
        IF (ntry < 2) THEN
          delta = 0.49999_hp / 86400.0_hp
          ntry = ntry + 1
        ELSE
          iyear = imissv
          EXIT
        END IF
      ELSE
        EXIT
      END IF
    END DO

  END SUBROUTINE juldaytogreg

!================================================================================
```

```
!-----------------------------------------------------------------
! S U B R O U T I N E   D A Y  O F  Y E A R  T O  G R E G
!-----------------------------------------------------------------
!-----------------------------------------------------------------
SUBROUTINE dayofyeartogreg(inYR, inDY, iYear, iMonth, iDay)

  IMPLICIT NONE

  ! Global Variables
  INTEGER, INTENT(IN)  :: inYR, inDY
  INTEGER, INTENT(OUT) :: iYear, iMonth, iDay

  ! Local Variables
  REAL(SZ) :: julDay
  INTEGER  :: yr, mo, da, hh, mm, ss

  !----- START CALCULATIONS -----

  ! Check for valid day of year (Gregorian calendar only)
  IF ((inyr < 1582) .OR. (indy < 1) .OR. (indy > 366) ) THEN
    iyear  = imissv
    imonth = imissv
    iday   = imissv

    RETURN
  END IF

  julday = gregtojulday(inyr, 1, 1, 0, 0, 0) + (indy - 1) * 1.0_hp

  CALL juldaytogreg(julday, yr, mo, da, hh, mm, ss)

  iyear  = yr
  imonth = mo
  iday   = da

END SUBROUTINE dayofyeartogreg

!================================================================================

!-----------------------------------------------------------------
! S U B R O U T I N E   S P L I T  D A T E  T I M E  S T R I N G
!-----------------------------------------------------------------
!-----------------------------------------------------------------
SUBROUTINE splitdatetimestring(inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)

  IMPLICIT NONE

  ! Global Variables
  CHARACTER(LEN=*), INTENT(IN)    :: inDateTime
  INTEGER, INTENT(OUT)            :: iYear, iMonth, iDay, iHour, iMin, iSec

  ! Local Variables
  CHARACTER(LEN=LEN(inDateTime)) :: tmpDateStr
  INTEGER                        :: errIO

  !----- START CALCULATIONS -----

  tmpdatestr = preprocessdatetimestring(indatetime)

  IF (trim(tmpdatestr) == ”) THEN
    iyear  = imissv
    imonth = 0
    iday   = 0
    ihour  = 0
    imin   = 0
    isec   = 0

    RETURN
  END IF

  READ(tmpdatestr(1:4), '(I4.4)', iostat=errio) iyear
    IF ((errio /= 0) .OR. (iyear < 1582)) iyear = imissv

  READ(tmpdatestr(5:6), '(I2.2)', iostat=errio) imonth
    IF ((errio /= 0) .OR. (imonth < 1) .OR. (imonth > 12)) imonth = 0

  READ(tmpdatestr(7:8), '(I2.2)', iostat=errio) iday
    IF ((errio /= 0) .OR. (iday < 0) .OR. (iday > monthdays(iyear, imonth))) iday = 0

  READ(tmpdatestr(9:10), '(I2.2)', iostat=errio) ihour
    IF ((errio /= 0) .OR. (ihour < 0) .OR. (ihour >= 23)) ihour = 0
```

```
    READ(tmpdatestr(11:12), '(I2.2)', iostat=errio) imin
      IF ((errio /= 0) .OR. (imin < 0) .OR. (imin >= 60)) imin = 0

    READ(tmpdatestr(13:14), '(I2.2)', iostat=errio) isec
      IF ((errio /= 0) .OR. (isec < 0) .OR. (isec >= 60)) isec = 0

  END SUBROUTINE splitdatetimestring

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S P L I T   D A T E   T I M E   S T R I N G   2
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE splitdatetimestring2(inDateTime, iDate, iTime)

    IMPLICIT NONE

    ! Global Variables
    CHARACTER(LEN=*), INTENT(IN)    :: inDateTime
    INTEGER, INTENT(OUT)            :: iDate, iTime

    ! Local Variables
    INTEGER                         :: iYear, iMonth, iDay, iHour, iMin, iSec

    !----- START CALCULATIONS -----

    CALL splitdatetimestring(indatetime, iyear, imonth, iday, ihour, imin, isec)

    IF ((iyear == imissv) .OR. (imonth <= 0) .OR. (iday <= 0)) THEN
      idate = imissv
    ELSE
      idate = joindate(iyear, imonth, iday)
    END IF

    itime = joindate(ihour, imin, isec)

  END SUBROUTINE splitdatetimestring2

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   P R E   P R O C E S S   D A T E   T I M E   S T R I N G
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  FUNCTION preprocessdatetimestring(inDateTime) Result(myValOut)

    IMPLICIT NONE

    ! Global Variables
    CHARACTER(LEN=*), INTENT(IN)    :: indatetime
    CHARACTER(LEN=LEN(inDateTime)) :: myvalout

    ! Local Variables
    CHARACTER(LEN=1)                :: c
    INTEGER                         :: i, ipos

    !----- START CALCULATIONS -----

    myvalout = blank
    ipos = 1

    DO i = 1, len(indatetime)
      c = indatetime(i:i)
      IF ((48 <= ichar(c)) .AND. (ichar(c) <= 57)) THEN
        myvalout(ipos:ipos) = c
        ipos = ipos + 1
      ENDIF
    END DO

    RETURN

  END FUNCTION preprocessdatetimestring

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   J O I N   D A T E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
```

```fortran
  INTEGER FUNCTION joindate(iYear, iMonth, iDay) RESULT(myVal)

    IMPLICIT NONE

    ! Global Variables
    INTEGER, INTENT(IN) :: iyear, imonth, iday

    !----- START CALCULATIONS -----

    myval = iyear * 10000 + imonth * 100 + iday

  END FUNCTION joindate

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S P L I T   D A T E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE splitdate(inDate, iYear, iMonth, iDay)

    IMPLICIT NONE

    ! Global Variables
    INTEGER, INTENT(IN)  :: inDate
    INTEGER, INTENT(OUT) :: iYear, iMonth, iDay

    !----- START CALCULATIONS -----

    iyear  = indate / 10000
    imonth = indate / 100 - iyear * 100
    iday   = indate - imonth * 100 - iyear * 10000

  END SUBROUTINE splitdate

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   D A T E   T I M E 2   S T R I N G
  !----------------------------------------------------------------
  !              (optional - for sep <= 0 use ' ', for sep > 0 use 'T')
  !              (optional - units = [S(seconds), M(minutes), H(hours), D(days), W(weeks)])
  !----------------------------------------------------------------
  FUNCTION datetime2string(year, month, day, hour, min, sec, sep, units, zone, err) result(myValOut)

    IMPLICIT NONE

    INTEGER,            INTENT(IN)           :: year, month, day
    INTEGER, OPTIONAL,  INTENT(IN)           :: sep, hour, min, sec
    CHARACTER(LEN=*), OPTIONAL, INTENT(IN)   :: units, zone
    INTEGER, OPTIONAL, INTENT(OUT)           :: err ! Error status, 0 if success, nonzero in case of format
 error.

    ! The resulting date time string. Considering using trim() on it.
    CHARACTER(LEN=64) :: myvalout
    CHARACTER(LEN=20) :: myunits, myzone
    CHARACTER(LEN=1)  :: mytimesep
    INTEGER           :: myhour, mymin, mysec, myerr

      myhour = 0
    IF (PRESENT(hour)) myhour = hour
      mymin = 0
    IF (PRESENT(min))  mymin = min
      mysec = 0
    IF (PRESENT(sec))  mysec = sec

    mytimesep = ' '
    IF (PRESENT(sep)) THEN
      IF (sep  > 0) mytimesep = 'T'
      IF (sep <= 0) mytimesep = ' '
    END IF

    IF (PRESENT(units)) THEN
      SELECT CASE(trim(adjustl(upp(units))))
        CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
          myunits = 'seconds since'
        CASE('MINUTES', 'MINUTE', 'MIN', 'M')
          myunits = 'minutes since'
        CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
          myunits = 'hours since'
        CASE('DAYS', 'DAY', 'DA', 'D')
```

```
          myunits = 'days since'
        CASE('WEEKS', 'WEEK', 'WE', 'W')
          myunits = 'weeks since'
        CASE DEFAULT
          myvalout = ' '
      END SELECT
    ELSE
      myunits = ' '
    END IF

    IF (PRESENT(zone)) THEN
      myzone = adjustl(zone)
    ELSE
      myzone = ' '
    END IF

    !WRITE(myValOut, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":", i2.2, ":", i2.2, "Z")', IOSTAT=myErr) &
    !                  year, month, day, myTimeSep, myHour, myMin, mySec
    WRITE(myvalout, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":", i2.2, ":", i2.2)', iostat=myerr) &
                      year, month, day, mytimesep, myhour, mymin, mysec

     IF (len_trim(myunits) /= 0) THEN
       myvalout = trim(myunits) // " " // trim(myvalout)
     END IF

     IF (len_trim(myzone) /= 0) THEN
       myvalout = trim(myvalout) // " " // trim(myzone)
     END IF

    IF (PRESENT(err)) err = myerr

    RETURN

  END FUNCTION datetime2string

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G E T   T I M E   C O N V   S E C
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sz) function gettimeconvsec(units, invert) result(myvalout)

    IMPLICIT NONE

    CHARACTER(LEN=*), INTENT(IN)  :: units
    INTEGER, OPTIONAL, INTENT(IN) :: invert

    INTEGER                       :: myinvert
    CHARACTER(LEN=LEN(units))     :: myunits
    REAL(sz), PARAMETER           :: minsecs  = 60.0_sz
    REAL(sz), PARAMETER           :: hoursecs = 3600.0_sz
    REAL(sz), PARAMETER           :: daysecs  = 86400.0_sz
    REAL(sz), PARAMETER           :: weeksecs = 604800.0_sz


    myinvert = 0
    IF (PRESENT(invert)) THEN
      IF (invert  > 0) myinvert = 1
      IF (invert <= 0) myinvert = 0
    END IF

    myunits = adjustl(units)
    IF (myinvert == 0) THEN
      SELECT CASE(trim(upp(myunits)))
        CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
          myvalout = 1.0_sz
        CASE('MINUTES', 'MINUTE', 'MIN', 'M')
          myvalout = minsecs
        CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
          myvalout = hoursecs
        CASE('DAYS', 'DAY', 'DA', 'D')
          myvalout = daysecs
        CASE('WEEKS', 'WEEK', 'WE', 'W')
          myvalout = weeksecs
        CASE DEFAULT
          myvalout = 1.0_sz
      END SELECT
    ELSE
      SELECT CASE(trim(upp(myunits)))
        CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
```

```
        myvalout = 1.0_sz
      CASE('MINUTES', 'MINUTE', 'MIN', 'M')
        myvalout = 1.0_sz / minsecs
      CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
        myvalout = 1.0_sz / hoursecs
      CASE('DAYS', 'DAY', 'DA', 'D')
        myvalout = 1.0_sz / daysecs
      CASE('WEEKS', 'WEEK', 'WE', 'W')
        myvalout = 1.0_sz / weeksecs
      CASE DEFAULT
        myvalout = 1.0_sz
    END SELECT
  END IF

  RETURN

END FUNCTION gettimeconvsec

!================================================================================

!----------------------------------------------------------------
! F U N C T I O N   E L A P S E D   S E C S
!----------------------------------------------------------------
!----------------------------------------------------------------
REAL(sz) function elapsedsecs(intime1, intime2, inunits) result(myval)

  IMPLICIT NONE

  ! Global Variables
  REAL(sz), INTENT(IN)                       :: intime1, intime2
  CHARACTER(LEN=*), OPTIONAL, INTENT(IN)     :: inunits

  ! Local Variables
  REAL(sz)                       :: uconfac
  CHARACTER(LEN=:), ALLOCATABLE  :: unitsval

  !----- START CALCULATIONS -----

  IF (PRESENT(inunits)) THEN
    ALLOCATE(CHARACTER(LEN=LEN(inUnits)) :: unitsval)
    unitsval = inunits
  ELSE
    ALLOCATE(CHARACTER(LEN=1) :: unitsval)
    unitsval = 'S'
  END IF

  uconfac = gettimeconvsec(unitsval)

  myval = (intime2 - intime1) * uconfac
  myval = fixnearwholereal(myval, 0.001_sz)

  DEALLOCATE(unitsval)

  RETURN

END FUNCTION elapsedsecs

!================================================================================

!----------------------------------------------------------------
! F U N C T I O N   U P P
!----------------------------------------------------------------
!----------------------------------------------------------------
FUNCTION upp(inpString) RESULT(outString)

  CHARACTER(*), INTENT(IN)   :: inpstring

  INTEGER, PARAMETER         :: duc = ichar('A') - ichar('a')
  CHARACTER(LEN(inpString))  :: outstring
  CHARACTER                  :: ch
  INTEGER                    :: i

  DO i = 1, len(inpstring)
    ch = inpstring(i:i)
    IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
    outstring(i:i) = ch
  END DO

  RETURN

END FUNCTION upp
```

```

!==============================================================================

END MODULE timedateutils
```

9.38 utilities.F90 File Reference

Data Types

- interface utilities::geotocpp
- interface utilities::cpptogeo
- interface utilities::sphericaldistance

Modules

- module utilities

Functions/Subroutines

- subroutine utilities::openfileforread (lun, fileName, errorIO)
  This subroutine opens an existing file for reading.
- subroutine utilities::readcontrolfile (inpFile)
  This subroutine reads the program's main control file.
- subroutine utilities::printmodelparams ()
  This subroutine prints on the screen the values of the program's parameters.
- integer function utilities::getlinerecord (inpLine, outLine, lastCommFlag)
  Gets a line from a file.
- integer function utilities::parseline (inpLine, outLine, keyWord, nVal, cVal, rVal)
  This function parses lines of text from input script/control files.
- integer function utilities::checkcontrolfileinputs ()
  Checks the user defined control file inputs.
- integer function utilities::loadintvar (nInp, vInp, nOut, vOut)
  This function loads input values into a requested model integer variable.
- integer function utilities::loadlogvar (nInp, vInp, nOut, vOut)
  This function loads input values into a requested model logical variable.
- integer function utilities::loadrealvar (nInp, vInp, nOut, vOut)
  This function loads input values into a requested model real variable.
- pure character(len(inpstring)) function utilities::tolowercase (inpString)
  Convert a string to lower-case.
- pure character(len(inpstring)) function utilities::touppercase (inpString)
  Convert a string to upper-case.
- real(sz) function utilities::convlon (inpLon)
  Convert longitude values from the (0, 360) to the (-180, 180) notation.
- subroutine utilities::geotocpp_scalar (lat, lon, lat0, lon0, x, y)
  Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.
- subroutine utilities::geotocpp_1d (lat, lon, lat0, lon0, x, y)
  Transform from geographical (lon, lat) coordinates into CPP (x, y) coordinates.

- subroutine utilities::cpptogeo_scalar (x, y, lat0, lon0, lat, lon)

    Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- subroutine utilities::cpptogeo_1d (x, y, lat0, lon0, lat, lon)

    Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- real(sz) function utilities::sphericaldistance_scalar (lat1, lon1, lat2, lon2)

    Calculates the distance of two points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:), allocatable utilities::sphericaldistance_1d (lats, lons, lat0, lon0)

    Calculates the distance of points along the great circle using the Vincenty formula.
- real(sz) function, dimension(:, :), allocatable utilities::sphericaldistance_2d (lats, lons, lat0, lon0)

    Calculates the distance of points along the great circle using the Vincenty formula.
- real(sz) function utilities::sphericaldistanceharv (lat1, lon1, lat2, lon2)

    Calculates the distance of two points along the great circle using the Haversine formula.
- subroutine utilities::sphericalfracpoint (lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)

    Calculates the coordinates of an intermediate point between two points along the great circle.
- subroutine utilities::getlocandratio (val, arrVal, idx1, idx2, wtRatio)

    Calculates the location of a value in an 1D array of values.
- integer function utilities::charunique (inpVec, outVec, idxVec)

    Find the unique non-blank elements in 1D character array.
- real(sp) function utilities::valstr (String)

    Returns the value of the leading double precision real numeric string.
- real(hp) function utilities::dvalstr (String)

    Returns the value of the leading double precision real numeric string.
- integer function utilities::intvalstr (String)

    Returns the value of the leading integer numeric string.
- integer function utilities::realscan (String, Pos, Value)

    Scans string looking for the leading single precision real numeric string.
- integer function utilities::drealscan (String, Pos, Value)

    Scans string looking for the leading double precision real numeric string.
- integer function utilities::intscan (String, Pos, Signed, Value)

    Scans string looking for the leading integer numeric string.

Variables

- real(sz), parameter utilities::closetol = 0.001_SZ

9.38.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file utilities.F90.

9.39 utilities.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!                M O D U L E   U T I L I T I E S
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE utilities

  USE pahm_sizes
  USE pahm_messages

  IMPLICIT NONE

  INTEGER, PRIVATE    :: numBTFiles = 0
  REAL(sz), PARAMETER :: closetol = 0.001_sz

  !-----------------------------------------------------------------------
  ! I N T E R F A C E S
  !-----------------------------------------------------------------------
  INTERFACE geotocpp
    MODULE PROCEDURE geotocpp_scalar
    MODULE PROCEDURE geotocpp_1d
  END INTERFACE geotocpp

  INTERFACE cpptogeo
    MODULE PROCEDURE cpptogeo_scalar
    MODULE PROCEDURE cpptogeo_1d
  END INTERFACE cpptogeo

  INTERFACE sphericaldistance
    MODULE PROCEDURE sphericaldistance_scalar
    MODULE PROCEDURE sphericaldistance_1d
    MODULE PROCEDURE sphericaldistance_2d
  END INTERFACE sphericaldistance
  !-----------------------------------------------------------------------


  CONTAINS


  !-----------------------------------------------------------------------
  !     S U B R O U T I N E   O P E N   F I L E   F O R   R E A D
  !-----------------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE openfileforread(lun, fileName, errorIO)

    USE pahm_global

    IMPLICIT NONE

    ! Global variables
    INTEGER, INTENT(IN)           :: lun         ! fortran logical unit number
    CHARACTER(LEN=*), INTENT(IN)  :: fileName    ! full pathname of file
    INTEGER, INTENT(OUT)          :: errorIO     ! zero if the file opened successfully

    ! Local variables
    LOGICAL                       :: fileFound   ! .TRUE. if the file is present
    CHARACTER(LEN=LEN(fileName)) :: tmpFileName ! full pathname of file

    CALL setmessagesource("OpenFileForRead")

    errorio = 0

    tmpfilename = adjustl(filename)

    ! Check to see if file exists
    WRITE(scratchmessage, '("Searching for file to open on unit ", i0, "...")') lun
    CALL logmessage(info, trim(scratchmessage))

    INQUIRE(file=trim(filename), exist=filefound)
    IF (.NOT. filefound) THEN
      WRITE(scratchmessage, '("The file : ", a, " was not found.")') trim(tmpfilename)
      CALL allmessage(info, scratchmessage)

      errorio = 1

      CALL unsetmessagesource()
```

```fortran

      RETURN  ! file not found
    ELSE
      WRITE(scratchmessage, '("The file : ", a, " was found. The file will be opened.")') trim(tmpfilename)

      CALL logmessage(info, trim(scratchmessage))
    END IF

    ! Open existing file
    OPEN(unit=lun, file=trim(tmpfilename), status='OLD', action='READ', iostat=errorio)
    IF (errorio /= 0) THEN
      WRITE(scratchmessage, '("Could not open the file: ", a, ".")') trim(tmpfilename)

      CALL allmessage(error, trim(scratchmessage))

      CALL unsetmessagesource()

      RETURN  ! file found but could not be opened
    ELSE
      WRITE(scratchmessage, '("The file ", a, " was opened successfully.")') trim(tmpfilename)

      CALL logmessage(info, trim(scratchmessage))
    END IF

    CALL unsetmessagesource()

    RETURN

  END SUBROUTINE openfileforread

!================================================================================

  !-----------------------------------------------------------------------
  !     S U B R O U T I N E   R E A D   C O N T R O L   F I L E
  !-----------------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE readcontrolfile(inpFile)

    USE pahm_global
    USE pahm_messages
    USE timedateutils, ONLY : timeconv, splitdatetimestring, joindate,  gregtojulday, juldaytogreg, &
                              gettimeconvsec, datetime2string

    IMPLICIT NONE

    ! Global variables
    CHARACTER(LEN=*), INTENT(IN)        :: inpFile

    ! Local variables
    LOGICAL                             :: fileFound   ! .TRUE. if the file is present
    CHARACTER(LEN=LEN(inpFile))         :: tmpFileName
    CHARACTER(LEN=512)                  :: inpLine, outLine
    CHARACTER(LEN=40)                   :: keyWord

    INTEGER                             :: iUnit, errIO, status

    INTEGER                             :: nPnts
    INTEGER                             :: nVal, i
    REAL(SZ), DIMENSION(200)            :: realVal
    CHARACTER(LEN=512), DIMENSION(200)  :: charVal
    CHARACTER(LEN=512)                  :: tmpCharVal

    INTEGER                             :: iValOut(1)
    REAL(SZ)                            :: rValOut(1)

    LOGICAL                             :: wrtPARS, gotNBTRFILES = .false.

    CHARACTER(LEN=512)                  :: cntlFmtStr, fmtDimParInvalid, fmtParNotFound
    CHARACTER(LEN=FNAMELEN)             :: tmpStr
    REAL(SZ)                            :: jday


    !---------- Initialize variables
    ! Global variables
    numbtfiles                = 0

    ! Local variables
    inpline                   = blank
    outline                   = blank
    keyword                   = blank
```

```
    charval                     = blank
    cntlfmtstr                  = blank
    fmtdimparinvalid            = blank
    fmtparnotfound              = blank
    tmpstr                      = blank

    iunit = lun_ctrl
    errio = 0
    !----------


    CALL setmessagesource("ReadControlFile")

    !---------- Establish the format variables
    cntlfmtstr = ' "in control file ' // "<" // trim(adjustl(inpfile)) // ">" // '"'

    fmtdimparinvalid  = '(" Invalid dimension parameter: ", a, 1x, i0, 2x, '
      fmtdimparinvalid  = trim(fmtdimparinvalid) // trim(cntlfmtstr) // ', 1x, a)'

    fmtparnotfound = '(" Could not find input parameter: ", a, 1x, '
      fmtparnotfound = trim(fmtparnotfound) // trim(cntlfmtstr) // ', 1x, a)'
    !----------

    tmpfilename = adjustl(inpfile)

    INQUIRE(file=trim(tmpfilename), exist=filefound)
    IF (.NOT. filefound) THEN
      WRITE(lun_screen, '("The control file : ", a, " was not found, cannot continue.")')
 trim(tmpfilename)

      stop  ! file not found
    ELSE
      WRITE(lun_screen, '("The contol file : ", a, " was found and will be opened for reading.")')
 trim(tmpfilename)
    END IF

    ! Open existing file
    OPEN(unit=iunit, file=trim(tmpfilename), status='OLD', action='READ', iostat=errio)
    IF (errio /= 0) THEN
      WRITE(lun_screen, '("Could not open the contol file: ", a, ".")') trim(tmpfilename)

      stop  ! file found but could not be opened
    END IF

    DO WHILE (.true.)
      READ(unit=iunit, fmt='(a)', err=10, END=20) inpline
      status = parseline(inpline, outline, keyword, nval, charval, realval)

      IF (status > 0) THEN
        SELECT CASE (touppercase(trim(keyword)))
          !----- CASE
          CASE ('TITLE')
            IF (nval == 1) THEN
              title = trim(adjustl(charval(nval)))
            ELSE
              WRITE(title, '(a, 1x, a)') trim(adjustl(title)), trim(adjustl(charval(nval)))
            END IF

          !----- CASE
          CASE ('LOGFILENAME')
            IF (nval == 1) THEN
              logfilename = trim(adjustl(charval(nval)))
            ELSE
              IF (trim(adjustl(logfilename)) == '') THEN
                logfilename = trim(adjustl(charval(nval)))
              END IF
            END IF

          !----- CASE
          CASE ('WRITEPARAMS')
            npnts = loadintvar(nval, realval, 1, ivalout)
            IF (ivalout(1) > 0) THEN
              writeparams = .true.
            ELSE
              writeparams = .false.
            END IF

          !----- CASE
          CASE ('NBTRFILES')
            npnts = loadintvar(nval, realval, 1, ivalout)
            nbtrfiles = ivalout(1)
```

```
            IF (nbtrfiles > 0) THEN
              ALLOCATE(besttrackfilename(nbtrfiles))
              besttrackfilename = blank
            END IF
            gotnbtrfiles = .true.

          !----- CASE
          CASE ('BESTTRACKFILENAME')
            IF (.NOT. gotnbtrfiles) THEN
              WRITE(scratchmessage, fmtparnotfound) 'nBTrFiles', '(add the "nBTrFiles" keyword before
"bestTrackFileName").'
              CALL allmessage(error, scratchmessage)
            ELSE
              IF (ALLOCATED(besttrackfilename)) THEN
                tmpstr = adjustl(charval(nval))
                IF (trim(tmpstr) == '') THEN
                  nval = nval - 1
                ELSE
                  IF (nval <= nbtrfiles) THEN ! because bestTrackFileName has been allocated this way
above
                    numbtfiles = numbtfiles + 1
                    besttrackfilename(numbtfiles) = trim(tmpstr)
                    besttrackfilenamespecified = .true.
                  END IF
                END IF
              END IF
            END IF

          !----- CASE
          CASE ('MESHFILETYPE')
            IF (nval == 1) THEN
              meshfiletype = trim(adjustl(charval(nval)))
            ELSE
              IF (trim(adjustl(meshfiletype)) == '') THEN
                meshfiletype = trim(adjustl(charval(nval)))
              END IF
            END IF

          !----- CASE
          CASE ('MESHFILENAME')
            IF (nval == 1) THEN
              meshfilename = trim(adjustl(charval(nval)))
            ELSE
              IF (trim(adjustl(meshfilename)) == '') THEN
                meshfilename = trim(adjustl(charval(nval)))
              END IF
            END IF
            IF (trim(adjustl(meshfilename)) /= '') meshfilenamespecified = .true.

          !----- CASE
          CASE ('MESHFILEFORM')
            IF (nval == 1) THEN
              meshfileform = trim(adjustl(charval(nval)))
            ELSE
              IF (trim(adjustl(meshfileform)) == '') THEN
                meshfileform = trim(adjustl(charval(nval)))
              END IF
            END IF

          !----- CASE
          CASE ('GRAVITY')
            npnts = loadrealvar(nval, realval, 1, rvalout)
            gravity = rvalout(1)

          !----- CASE
          CASE ('RHOWATER')
            npnts = loadrealvar(nval, realval, 1, rvalout)
            rhowater = rvalout(1)

          !----- CASE
          CASE ('RHOAIR')
            npnts = loadrealvar(nval, realval, 1, rvalout)
            rhoair = rvalout(1)

          !----- CASE
          CASE ('BACKGROUNDATMPRESS')
            npnts = loadrealvar(nval, realval, 1, rvalout)
            backgroundatmpress = rvalout(1)

          !----- CASE
          CASE ('BLADJUSTFAC')
```

```
          npnts = loadrealvar(nval, realval, 1, rvalout)
          bladjustfac = rvalout(1)

        !----- CASE
        CASE ('REFDATETIME')
          IF (nval == 1) THEN
            refdatetime = trim(adjustl(charval(nval)))
          ELSE
            IF (trim(adjustl(refdatetime)) == '') THEN
              refdatetime = trim(adjustl(charval(nval)))
            END IF
          END IF

          CALL splitdatetimestring(refdatetime, refyear, refmonth, refday, refhour, refmin, refsec)

          IF ((refyear == imissv) .OR. (refmonth <= 0) .OR. (refday <= 0)) THEN
            refdate = imissv
          ELSE
            refdate = joindate(refyear, refmonth, refday)
          END IF
          reftime = joindate(refhour, refmin, refsec)
          refdatespecified = .true.

        !----- CASE
        CASE ('UNITTIME')
          IF (begdatespecified .OR. begsimspecified .OR. &
              enddatespecified .OR. endsimspecified) THEN
            scratchmessage = 'add the "unitTime" keyword before the ' // &
                             '"begDateTime"/"begTime" and "endDateTime"/"endTime" keywords'
            CALL allmessage(error, scratchmessage)
            CALL terminate()
          ELSE
            IF (nval == 1) THEN
              tmpcharval = touppercase(adjustl(charval(nval)))
              unittime = tmpcharval(1:1)
            ELSE
              IF (trim(adjustl(unittime)) == '') THEN
                tmpcharval = touppercase(adjustl(charval(nval)))
                unittime = tmpcharval(1:1)
              END IF
            END IF
          END IF

        !----- CASE
        CASE ('BEGDATETIME')
          IF (begdatespecified .OR. begsimspecified) THEN
            scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
            CALL allmessage(error, scratchmessage)

            begdatespecified = .false.
          ELSE
            IF (.NOT. refdatespecified) THEN
              scratchmessage = 'Add the "refDateTime" keyword before "begDateTime").'
              CALL allmessage(error, scratchmessage)
            ELSE
              IF (nval == 1) THEN
                begdatetime = trim(adjustl(charval(nval)))
              ELSE
                IF (trim(adjustl(begdatetime)) == '') THEN
                  begdatetime = trim(adjustl(charval(nval)))
                END IF
              END IF

              CALL splitdatetimestring(begdatetime, begyear, begmonth, begday, beghour, begmin, begsec)

              IF ((begyear == imissv) .OR. (begmonth <= 0) .OR. (begday <= 0)) THEN
                begdate = imissv
              ELSE
                begdate = joindate(begyear, begmonth, begday)
              END IF
              begtime = joindate(beghour, begmin, begsec)

              CALL timeconv(begyear, begmonth, begday, beghour, begmin, begsec, mdbegsimtime)
              begsimtime = mdbegsimtime * gettimeconvsec(unittime, 1)

              begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)

              begdatespecified = .true.
              begsimspecified  = .true.
            END IF
          END IF
```

```

        !----- CASE
        CASE ('ENDDATETIME')
          IF (enddatespecified .OR. endsimspecified) THEN
            scratchmessage = 'Only one of "endDateTime" or "endSimTime" can be specified'
            CALL allmessage(error, scratchmessage)

            enddatespecified = .false.
          ELSE
            IF (.NOT. refdatespecified) THEN
              scratchmessage = 'Add the "refDateTime" keyword before "endDateTime").'
              CALL allmessage(error, scratchmessage)
            ELSE
              IF (nval == 1) THEN
                enddatetime = trim(adjustl(charval(nval)))
              ELSE
                IF (trim(adjustl(enddatetime)) == '') THEN
                  enddatetime = trim(adjustl(charval(nval)))
                END IF
              END IF

              CALL splitdatetimestring(enddatetime, endyear, endmonth, endday, endhour, endmin, endsec)

              IF ((endyear == imissv) .OR. (endmonth <= 0) .OR. (endday <= 0)) THEN
                enddate = imissv
              ELSE
                enddate = joindate(endyear, endmonth, endday)
              END IF
              endtime = joindate(endhour, endmin, endsec)

              CALL timeconv(endyear, endmonth, endday, endhour, endmin, endsec, mdendsimtime)
              endsimtime = mdendsimtime * gettimeconvsec(unittime, 1)

              enddatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)

              enddatespecified = .true.
              endsimspecified  = .true.
            END IF
          END IF

        !----- CASE
        CASE ('OUTDT')
          npnts = loadrealvar(nval, realval, 1, rvalout)
          outdt = rvalout(1)
          mdoutdt = fixnearwholereal(outdt * gettimeconvsec(unittime), closetol)

        !----- CASE
        CASE ('BEGSIMTIME')
          IF (begdatespecified .OR. begsimspecified) THEN
            scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
            CALL allmessage(error, scratchmessage)

            begsimspecified = .false.
          ELSE
            IF (.NOT. refdatespecified) THEN
              scratchmessage = 'Add the "refDateTime" keyword before "begSimTime").'
              CALL allmessage(error, scratchmessage)
            ELSE
              npnts = loadrealvar(nval, realval, 1, rvalout)
              begsimtime = rvalout(1)

              mdbegsimtime = begsimtime * gettimeconvsec(unittime)

              jday = (mdbegsimtime *  gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
       refhour, refmin, refsec)
              CALL juldaytogreg(jday, begyear, begmonth, begday, beghour, begmin, begsec)
              begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)

              begdatespecified = .true.
              begsimspecified  = .true.
            END IF
          END IF

        !----- CASE
        CASE ('ENDSIMTIME')
          IF (enddatespecified .OR. endsimspecified) THEN
            scratchmessage = 'Only one of "endDateTime" and "endSimTime" can be specified'
            CALL allmessage(error, scratchmessage)

            endsimspecified = .false.
          ELSE
```

```
             IF (.NOT. refdatespecified) THEN
              scratchmessage = 'Add the "refDateTime" keyword before "endSimTime").'
              CALL allmessage(error, scratchmessage)
            ELSE
              npnts = loadrealvar(nval, realval, 1, rvalout)
              endsimtime = rvalout(1)

              mdendsimtime = endsimtime * gettimeconvsec(unittime)

              jday = (mdendsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
              CALL juldaytogreg(jday, endyear, endmonth, endday, endhour, endmin, endsec)
              begdatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)

              enddatespecified = .true.
              endsimspecified  = .true.
            END IF
          END IF

        !----- CASE
        CASE ('OUTFILENAME')
          IF (nval == 1) THEN
            outfilename = trim(adjustl(charval(nval)))
          ELSE
            IF (trim(adjustl(outfilename)) == '') THEN
              outfilename = trim(adjustl(charval(nval)))
            END IF
          END IF
          IF (trim(adjustl(outfilename)) /= '') outfilenamespecified = .true.

        !----- CASE
        CASE ('NCVARNAM_PRES')
          IF (nval == 1) THEN
            ncvarnam_pres = trim(adjustl(charval(nval)))
          ELSE
            IF (trim(adjustl(ncvarnam_pres)) == '') THEN
              ncvarnam_pres = trim(adjustl(charval(nval)))
            END IF
          END IF

        !----- CASE
        CASE ('NCVARNAM_WNDX')
          IF (nval == 1) THEN
            ncvarnam_wndx = trim(adjustl(charval(nval)))
          ELSE
            IF (trim(adjustl(ncvarnam_wndx)) == '') THEN
              ncvarnam_wndx = trim(adjustl(charval(nval)))
            END IF
          END IF

        !----- CASE
        CASE ('NCVARNAM_WNDY')
          IF (nval == 1) THEN
            ncvarnam_wndy = trim(adjustl(charval(nval)))
          ELSE
            IF (trim(adjustl(ncvarnam_wndy)) == '') THEN
              ncvarnam_wndy = trim(adjustl(charval(nval)))
            END IF
          END IF

        !----- CASE
        CASE ('NCSHUFFLE')
          npnts = loadintvar(nval, realval, 1, ivalout)
          ncshuffle = ivalout(1)

        !----- CASE
        CASE ('NCDEFLATE')
          npnts = loadintvar(nval, realval, 1, ivalout)
          ncdeflate = ivalout(1)

        !----- CASE
        CASE ('NCDLEVEL')
          npnts = loadintvar(nval, realval, 1, ivalout)
          ncdlevel = ivalout(1)

        !----- CASE
        CASE ('MODELTYPE')
          npnts = loadintvar(nval, realval, 1, ivalout)
          modeltype = ivalout(1)

        !----- CASE
```

```
          CASE DEFAULT
            ! Do nothing
        END SELECT
      END IF
    END DO

    10 WRITE(lun_screen, '("Error while processing line: ", a, " in file: ", a)') &
          trim(adjustl(inpline)), trim(tmpfilename)

    CLOSE(iunit)
    stop

    20 CLOSE(iunit)

    WRITE(lun_screen, '(a)') 'Finished processing the input fields from the control file ...'

    !--------------------
    !--- CHECK INPUT VARIABLES AND SET DEFAULTS
    !--------------------
    CALL initlogging()

    IF (checkcontrolfileinputs() /= 0) THEN
      WRITE(scratchmessage, '(a)') &
              'Errors found while processing the input variables. Check the log file for details.'
      CALL screenmessage(error, scratchmessage)
      CALL unsetmessagesource()
      CALL terminate()
    END IF

    CALL printmodelparams()

    CALL unsetmessagesource()

  END SUBROUTINE readcontrolfile

!================================================================================

  !----------------------------------------------------------------------
  !     S U B R O U T I N E   P R I N T   M O D E L   P A R A M S
  !----------------------------------------------------------------------
  !---------------------------------------------------------------
  SUBROUTINE printmodelparams()

    USE pahm_global

    IMPLICIT NONE

    CHARACTER(LEN=128) :: tmpStr
    INTEGER            :: i

    IF (writeparams) THEN
        print *, ''
      WRITE(*, '(a)')    '---------- MODEL PARAMETERS ----------'

      WRITE(*, '(a, a)')    '   title                = ', trim(adjustl(title))

      DO i = 1, nbtrfiles
        WRITE(*, '(a, "(", i1, ")", a)')  '   bestTrackFileName', i, " = " //
 trim(adjustl(besttrackfilename(i)))
      END DO
      WRITE(*, '(a, a)')    '   meshFileType         = ', trim(adjustl(meshfiletype))
      WRITE(*, '(a, a)')    '   meshFileName         = ', trim(adjustl(meshfilename))
      WRITE(*, '(a, a)')    '   meshFileForm         = ', trim(adjustl(meshfileform))

        print *, ''
         WRITE(tmpstr, '(f20.5)') gravity
      WRITE(*, '(a, a)')    '   gravity              = ', trim(adjustl(tmpstr)) // " m/s^2"
         WRITE(tmpstr, '(f20.5)') rhowater
      WRITE(*, '(a, a)')    '   rhoWater             = ', trim(adjustl(tmpstr)) // " kg/m^3"
         WRITE(tmpstr, '(f20.5)') rhoair
      WRITE(*, '(a, a)')    '   rhoAir               = ', trim(adjustl(tmpstr)) // " kg/m^3"
         WRITE(tmpstr, '(f20.5)') backgroundatmpress
      WRITE(*, '(a, a)')    '   backgroundAtmPress   = ', trim(adjustl(tmpstr)) // " mbar"
         WRITE(tmpstr, '(f20.2)') bladjustfac
      WRITE(*, '(a, a)')    '   blAdjustFac          = ', trim(adjustl(tmpstr))

        print *, ''
      WRITE(*, '(a, a)')    '   refDateTime          = ', trim(adjustl(refdatetime))
      WRITE(*, '(a, i4.4)') '   refYear              = ', refyear
      WRITE(*, '(a, i2.2)') '   refMonth             = ', refmonth
      WRITE(*, '(a, i2.2)') '   refDay               = ', refday
```

```
      WRITE(*, '(a, i2.2)') '   refHour             = ', refhour
      WRITE(*, '(a, i2.2)') '   refMin              = ', refmin
      WRITE(*, '(a, i2.2)') '   refSec              = ', refsec
      WRITE(*, '(a, l1)')   '   refDateSpecified    = ', refdatespecified

        print *, ''
      WRITE(*, '(a, a)')    '   begDateTime         = ', trim(adjustl(begdatetime))
      WRITE(*, '(a, i4.4)') '   begYear             = ', begyear
      WRITE(*, '(a, i2.2)') '   begMonth            = ', begmonth
      WRITE(*, '(a, i2.2)') '   begDay              = ', begday
      WRITE(*, '(a, i2.2)') '   begHour             = ', beghour
      WRITE(*, '(a, i2.2)') '   begMin              = ', begmin
      WRITE(*, '(a, i2.2)') '   begSec              = ', begsec
      WRITE(*, '(a, l1)')   '   begDateSpecified    = ', begdatespecified

        print *, ''
      WRITE(*, '(a, a)')    '   endDateTime         = ', trim(adjustl(enddatetime))
      WRITE(*, '(a, i4.4)') '   endYear             = ', endyear
      WRITE(*, '(a, i2.2)') '   endMonth            = ', endmonth
      WRITE(*, '(a, i2.2)') '   endDay              = ', endday
      WRITE(*, '(a, i2.2)') '   endHour             = ', endhour
      WRITE(*, '(a, i2.2)') '   endMin              = ', endmin
      WRITE(*, '(a, i2.2)') '   endSec              = ', endsec
      WRITE(*, '(a, l1)')   '   endDateSpecified    = ', enddatespecified

        print *, ''
      WRITE(*, '(a, a1)')   '   unitTime            = ', unittime
         WRITE(tmpstr, '(f20.5)') outdt
      WRITE(*, '(a, a)')    '   outDT               = ', trim(adjustl(tmpstr)) // " " //
 tolowercase(trim(unittime))
         WRITE(tmpstr, '(f20.5)') mdoutdt
      WRITE(*, '(a, a)')    '   mdOutDT             = ', trim(adjustl(tmpstr)) // " s"
         WRITE(tmpstr, '(f20.5)') begsimtime
      WRITE(*, '(a, a)')    '   begSimTime          = ', trim(adjustl(tmpstr)) // " " //
 tolowercase(trim(unittime))
        WRITE(tmpstr, '(f20.5)') mdbegsimtime
      WRITE(*, '(a, a)')    '   mdBegSimTime        = ', trim(adjustl(tmpstr)) // " s"
      WRITE(*, '(a, l1)')   '   begSimSpecified     = ', begsimspecified
         WRITE(tmpstr, '(f20.5)') endsimtime
      WRITE(*, '(a, a)')    '   endSimTime          = ',  trim(adjustl(tmpstr)) // " " //
 tolowercase(trim(unittime))
         WRITE(tmpstr, '(f20.5)') mdendsimtime
      WRITE(*, '(a, a)')    '   mdEndSimTime        = ', trim(adjustl(tmpstr)) // " s"
      WRITE(*, '(a, l1)')   '   endSimSpecified     = ', endsimspecified
         WRITE(tmpstr, '(i10)') noutdt
      WRITE(*, '(a, a)')    '   nOutDT              = ', trim(adjustl(tmpstr))

        print *, ''
      WRITE(*, '(a, a)')    '   outFileName         = ', trim(adjustl(outfilename))
      WRITE(*, '(a, i1)')   '   ncShuffle           = ', ncshuffle
      WRITE(*, '(a, i1)')   '   ncDeflate           = ', ncdeflate
      WRITE(*, '(a, i1)')   '   ncDLevel            = ', ncdlevel
      WRITE(*, '(a, a)')    '   ncVarNam_Pres       = ', trim(ncvarnam_pres)
      WRITE(*, '(a, a)')    '   ncVarNam_WndX       = ', trim(ncvarnam_wndx)
      WRITE(*, '(a, a)')    '   ncVarNam_WndY       = ', trim(ncvarnam_wndy)

        print *, ''
      WRITE(*, '(a, i1)')   '   modelType           = ', modeltype

      WRITE(*, '(a)')    '---------- MODEL PARAMETERS ----------'
        print *, ''
    END IF

  END SUBROUTINE printmodelparams

!================================================================================

  !------------------------------------------------------------------
  ! F U N C T I O N   G E T   L I N E   R E C O R D
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  INTEGER FUNCTION getlinerecord(inpLine, outLine, lastCommFlag) RESULT(myLen)

    IMPLICIT NONE

    ! Imported variable declarations.
    CHARACTER(LEN=*), INTENT(IN)              :: inpline
    CHARACTER(LEN=LEN(inpLine)), INTENT(OUT)  :: outline
    INTEGER, OPTIONAL                         :: lastcommflag

    ! Local variable declarations.
```

```
    CHARACTER(LEN=LEN(inpLine))               :: line
    CHARACTER                                 :: tmpinpline(len(inpline))
    INTEGER                                   :: lenline, commflag, icomm

    ! Table of some ASCII character symbols
    !   CHAR     ASCII
    !    TAB        9
    !    SPC       32
    !    !         33
    !    #         35
    !    *         42
    !    +         43
    !    -         45
    !    /         47
    !    :         58
    !    =         61
    !    \         92
    !    |        124

    mylen      = 0
    outline    = blank
    tmpinpline = blank

    commflag = 0
    IF (PRESENT(lastcommflag)) THEN
      IF (lastcommflag <= 0) commflag = 0
      IF (lastcommflag > 0)  commflag = 1
    END IF

    tmpinpline = transfer(inpline, tmpinpline)
    tmpinpline = pack(tmpinpline, tmpinpline /= achar(9), spread(' ', 1, len(inpline)))

    line       =  trim(adjustl(transfer(tmpinpline, line)))
    lenline    = len_trim(line)

    IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
      IF (commflag > 0) THEN
        icomm = index(line, char(33), back = .false.)
        IF (icomm == 0) icomm = index(line, char(35), back = .false.)
        IF (icomm > 0) lenline = icomm - 1

        line = trim(adjustl(line(1:lenline)))
        lenline = len_trim(line)
      END IF

      outline = line
      mylen   = lenline
    END IF

    RETURN

  END FUNCTION getlinerecord

!==============================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   P A R S E  L I N E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION parseline(inpLine, outLine, keyWord, nVal, cVal, rVal) RESULT(myStatus)

    IMPLICIT NONE

    ! Imported variable declarations.
    CHARACTER(LEN=*), INTENT(IN)                          :: inpline
    CHARACTER(LEN=LEN(inpLine)), INTENT(OUT)              :: outline
    CHARACTER(LEN=40), INTENT(INOUT)                      :: keyword
    INTEGER, INTENT(INOUT)                                :: nval
    CHARACTER(LEN=512), DIMENSION(200), INTENT(INOUT)     :: cval
    REAL(sz), DIMENSION(200), INTENT(INOUT)               :: rval

    ! Local variable declarations
    LOGICAL                                               :: isstring, kextract, decflag, nested
    INTEGER                                               :: iblank, icont, ipipe, kstr, kend
    INTEGER                                               :: lend, lens, lstr, lval, nmul, schar
    INTEGER                                               :: copies, i, ic, ie, ierr, is, j, status
    INTEGER, DIMENSION(20)                                :: imul
    CHARACTER(LEN=256)                                    :: vstring, string
    CHARACTER(LEN=LEN(inpLine))                           :: line
    INTEGER                                               :: lenline

```

```
! Table of some ASCII character symbols
!   CHAR     ASCII
!    TAB        9
!    SPC       32
!    !         33
!    #         35
!    *         42
!    +         43
!    -         45
!    /         47
!    :         58
!    =         61
!    \         92
!    |        124

! Initialize.
line        = blank
vstring     = blank
string      = blank

lenline = getlinerecord(inpline, line, 1)
outline = line

! If not a blank or comment line [CHAR(33)=!], decode and extract input
! values.  Find equal sign [CHAR(61)].
status = -1
nested = .false.
IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
  status = 1
  kstr = 1
  kend = index(line, char(61), back = .false.) - 1
  lstr = index(line, char(61), back = .true.) + 1

  ! Determine if KEYWORD is followed by double equal sign (==) indicating
  ! nested parameter.
  IF ((lstr - kend) == 3) nested = .true.

  ! Extract KEYWORD, trim leading and trailing blanks.
  kextract = .false.
  IF (kend > 0) THEN
    lend = lenline
    keyword = line(kstr:kend)
    nval = 0
    kextract = .true.
  ELSE
    lstr = 1
    lend = lenline
    kextract = .true.
  END IF

  ! Extract parameter values string.  Remove continuation symbol
  ! [CHAR(92)=\] or multi-line value [CHAR(124)=|], if any.  Trim
  ! leading trailing blanks.
  IF (kextract) THEN
    icont = index(line, char(92 ), back = .false.)
    ipipe = index(line, char(124), back = .false.)
    IF (icont > 0) lend = icont - 1
    IF (ipipe > 0) lend = ipipe - 1
    vstring = adjustl(line(lstr:lend))
    lval = len_trim(vstring)

    ! The PROGRAM, VERSION and TITLE KEYWORDS are special ones because
    ! they can include strings, numbers, spaces, and continuation symbol.
    isstring = .false.
    SELECT CASE (touppercase(trim(keyword)))
      CASE ('TITLE')
        nval = nval + 1
        cval(nval) = vstring(1:lval)
        isstring = .true.

      CASE ('LOGFILENAME')
        nval = nval + 1
        cval(nval) = vstring(1:lval)
        isstring = .true.

      CASE ('BESTTRACKFILENAME')
        nval = nval + 1
        cval(nval) = vstring(1:lval)
        isstring = .true.

      CASE ('MESHFILENAME')
```

```
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('MESHFILETYPE')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('MESHFILEFORM')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('REFDATETIME')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

           CASE ('UNITTIME')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('BEGDATETIME')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('ENDDATETIME')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('OUTFILENAME')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('NCVARNAM_PRES')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('NCVARNAM_WNDX')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE ('NCVARNAM_WNDY')
            nval = nval + 1
            cval(nval) = vstring(1:lval)
            isstring = .true.

          CASE DEFAULT
            ! For every other KEYWORD except the above.
            ! Check if there is a multiplication symbol [CHAR(42)=*] in the variable
            ! string indicating repetition of input values.
            nmul = 0
            DO i = 1, lval
              IF (vstring(i:i) == char(42)) THEN
                nmul = nmul + 1
                imul(nmul) = i
              END IF
            END DO
            ic = 1

            ! Check for blank spaces [CHAR(32)=' '] between entries and decode.
            is = 1
            ie = lval
            iblank = 0
            decflag = .false.
            DO i = 1, lval
              IF (vstring(i:i) == char(32)) THEN
                IF (vstring(i + 1:i + 1) /= char(32)) decflag = .true.
                iblank = i
              ELSE
                ie = i
              END IF
              IF (decflag .OR. (i == lval)) THEN
                nval = nval + 1
```

```

        ! Processing numeric values.  Check starting character to determine
        ! if numeric or character values. It is possible to have both when
        ! processing repetitions via the multiplication symbol.
        schar = ichar(vstring(is:is))
        IF (((48 <= schar) .AND. (schar <= 57)) .OR. (schar == 43) .OR. (schar == 45)) THEN
         IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
          READ(vstring(is:imul(ic) - 1), *) copies
          schar = ichar(vstring(imul(ic) + 1:imul(ic) + 1))
          IF ((43 <= schar) .AND. (schar <= 57)) THEN
           READ(vstring(imul(ic) + 1:ie), *) rval(nval)
           DO j = 1, copies - 1
            rval(nval + j) = rval(nval)
           END DO
          ELSE
           string = vstring(imul(ic) + 1:ie)
           lens = len_trim(string)
           cval(nval) = string(1:lens)
           DO j = 1, copies - 1
            cval(nval + j) = cval(nval)
           END DO
          END IF
          nval = nval + copies - 1
          ic = ic + 1
         ELSE
          string = vstring(is:ie)
          lens = len_trim(string)
          !READ(string(1:lEns), *) rVal(nVal)
          READ(string(1:lens), *, iostat=ierr) rval(nval)
          IF (ierr /= 0) THEN
           WRITE(*, *) '#### ERROR :: Cannot interpret string ', string(1:lens),   &
                 ' as a REAL number.'
          END IF
         END IF
        ELSE

         ! Processing character values (logicals and strings).
         IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
          READ(vstring(is:imul(ic) - 1), *) copies
          cval(nval) = vstring(imul(ic) + 1:ie)
          DO j = 1, copies - 1
           cval(nval + j) = cval(nval)
          END DO
          nval = nval + copies - 1
          ic = ic + 1
         ELSE
          string = vstring(is:ie)
          cval(nval) = trim(adjustl(string))
         END IF
         isstring = .true.
        END IF
        is = iblank + 1
        ie = lval
        decflag = .false.
       END IF
      END DO
    END SELECT ! keyWord
   END IF ! kExtract
   status = nval
  END IF

  mystatus = status

  RETURN

 END FUNCTION parseline

!========================================================================

 !----------------------------------------------------------------------
 !    S U B R O U T I N E   C H E C K  C O N T R O L  F I L E  I N P U T S
 !----------------------------------------------------------------------
 !-----------------------------------------------------------------
 INTEGER FUNCTION checkcontrolfileinputs() RESULT(errStatus)

  USE pahm_global
  USE timedateutils, ONLY : firstgregdate, firstgregtime, &
                            gregtojulday, joindate, gettimeconvsec

  IMPLICIT NONE

```

```
  ! Local variables
  INTEGER                        :: errio, errnum
  INTEGER                        :: icnt
  LOGICAL                        :: filefound
  REAL(sz)                       :: gregjd, refjd, jd0, jd1
  REAL(sz)                       :: timesec, timeconvfac
  CHARACTER(LEN=64)              :: tmpstr, tmpstr1, tmpstr2


  !---------- Initialize variables
  errio          = 0
  errnum         = 0
  errstatus      = 0
  scratchmessage = blank


  CALL setmessagesource("CheckControlFileInputs")

  !----- 1) Best track files (mandatory variables) -----
  IF (nbtrfiles <= 0) THEN
    errnum = 1

    WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
                        '. Invalid value supplied for dimension parameter: nBTrFiles = ', &
                        nbtrfiles, ' (should be greater than zero).'
    CALL logmessage(error, scratchmessage)
  ELSE IF (besttrackfilenamespecified) THEN
    IF (numbtfiles /= nbtrfiles) THEN
      errnum = 2

      WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
                          '. The number of files for <bestTrackFileName> should be equal to nBTrFiles: ', &
                          nbtrfiles, '.'
      CALL logmessage(error, scratchmessage)
    END IF

    DO icnt = 1, numbtfiles
      INQUIRE(file=trim(adjustl(besttrackfilename(icnt))), iostat=errio, exist=filefound)
      IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
        errnum = 3

        WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                            '. Could not access the best track file for read: ' // &
                            trim(adjustl(besttrackfilename(icnt)))
        CALL logmessage(error, scratchmessage)
      END IF
    END DO
  ELSE
    errnum = 4

    WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                        '. bestTrackFileName(s) not specified. This is a mandatory variable. '
    CALL logmessage(error, scratchmessage)
  END IF

  !----- 2) Mesh file (mandatory variables) -----
  IF (.NOT. meshfilenamespecified) THEN
    errnum = 5

    WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                     '. Parameter <meshFileName> is not specified (mandatory variable) '
    CALL logmessage(error, scratchmessage)
  ELSE
    meshfilename = adjustl(meshfilename)
    INQUIRE(file=trim(meshfilename), iostat=errio, exist=filefound)
    IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
      errnum = 6

      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                       '. Could not access the mesh file for read: ' // trim(meshfilename)
      CALL logmessage(error, scratchmessage)
    END IF
  END IF

  ! Check for meshFileType
  meshfiletype = touppercase(trim(adjustl(meshfiletype)))
  SELECT CASE (trim(meshfiletype))
    !CASE ('ADCIRC', 'SCHISM', 'FVCOM', 'ROMS', 'GENERIC')
    CASE ('ADCIRC', 'SCHISM')
      ! These are the valid values
```

```
      CASE DEFAULT
        WRITE(scratchmessage, '(a)') 'This file type is not supported: meshFileType = ' // trim(meshfiletype)
        CALL logmessage(info, scratchmessage)

        meshfiletype = 'ADCIRC'

        WRITE(scratchmessage, '(a)') 'This value of meshFileType is adjusted to: meshFileType = ' // trim(meshfiletype)
        CALL logmessage(info, scratchmessage)
    END SELECT

    ! Check for meshFileForm
    meshfileform = touppercase(trim(adjustl(meshfileform)))
    SELECT CASE (trim(meshfileform))
      !CASE ('ASCII', 'NETCDF')
      CASE ('ASCII')
        ! These are valid values

      CASE DEFAULT
        WRITE(scratchmessage, '(a)') 'This file format is not supported: meshFileForm = ' // trim(meshfileform)
        CALL logmessage(info, scratchmessage)

        meshfileform = 'ASCII'

        WRITE(scratchmessage, '(a)') 'This value of meshFileForm is adjusted to: meshFileForm = ' // trim(meshfileform)
        CALL logmessage(info, scratchmessage)
    END SELECT

    !----- 3) Reference date and time (mandatory variables) -----
    gregjd = gregtojulday(firstgregdate, firstgregtime)
    refjd  = gregtojulday(refdate, reftime)
    IF (refjd < gregjd) THEN
      errnum = 7
      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                            '. Invalid DateTime string was supplied: refDateTime = ' // trim(refdatetime)
      CALL logmessage(error, scratchmessage)
    END IF

    !----- 4) Stepping parameters (mandatory variables) -----
    ! check for valid start time
    IF (begsimspecified) THEN
      IF (refjd + (mdbegsimtime * gettimeconvsec('D', 1)) < gregjd) THEN
        errnum = 8
        WRITE(tmpstr, '(f20.5)') begsimtime
        WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                              '. Invalid start time in reference to refDateTime was supplied: begSimTime = ' // &
                              trim(adjustl(tmpstr))
        CALL logmessage(error, scratchmessage)
      END IF
    ELSE
      errnum = 81
      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                            '. Neither "begDateTime" or "begSimTime" are defined properly'
      CALL logmessage(error, scratchmessage)
    END IF

    ! check for valid stop time
    IF (endsimspecified) THEN
      IF (comparereals(endsimtime, begsimtime, closetol) <= 0) THEN
        errnum = 9
        WRITE(tmpstr1, '(f20.5)') begsimtime
        WRITE(tmpstr2, '(f20.5)') endsimtime
        WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                              '. Stop time should be greater than start time: begSimTime = ' // &
                              trim(adjustl(tmpstr1)) // ', endSimTime = ' // trim(adjustl(tmpstr2))
        CALL logmessage(error, scratchmessage)
      END IF
    ELSE
      errnum = 91
      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                            '. Neither "endDateTime" or "endSimTime" are defined properly'
      CALL logmessage(error, scratchmessage)
    END IF

    ! check for valid outDT; (endSimTime - begSimTime) should be an integral integral multiple of outDT
    IF (outdt <= 0) THEN
```

```
      WRITE(tmpstr, '(f20.5)') outdt
      WRITE(scratchmessage, '(a)') 'Frequency of output data should be greater than zero: outDT = ' // &
                               trim(adjustl(tmpstr))
      CALL logmessage(info, scratchmessage)

      mdoutdt = 3600.0
      outdt = fixnearwholereal(mdoutdt * gettimeconvsec(unittime, 1), closetol)

      WRITE(tmpstr, '(f20.5)') outdt
      WRITE(scratchmessage, '(a)') 'The outDT value is adjusted to: outDT = ' // trim(adjustl(tmpstr))
      CALL logmessage(info, scratchmessage)
    END IF

    jd0 = refjd + (mdbegsimtime * gettimeconvsec('D', 1))
    jd1 = refjd + (mdendsimtime * gettimeconvsec('D', 1))
    timesec = fixnearwholereal((jd1 - jd0) * gettimeconvsec('D'), closetol)
    IF ((timesec < mdoutdt) .OR. comparereals(modulo(timesec, mdoutdt), 0.0_sz) /= 0) THEN
      errnum = 10

      WRITE(tmpstr1, '(f20.5)') timesec
      WRITE(tmpstr2, '(f20.5)') outdt
      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                          '. The value of (endSimTime - begSimTime) = ' // trim(adjustl(tmpstr1)) // &
                          ' should be an integral multiple of outDT = ' // trim(adjustl(tmpstr2))
      CALL logmessage(error, scratchmessage)
    ELSE
      noutdt = int(timesec / mdoutdt) + 1
    END IF

    !----- 4) outFileName (mandatory variable) -----
    outfilename = adjustl(outfilename)
    IF (.NOT. outfilenamespecified) THEN
      errnum = 11

      WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
                          '. Output filename is not specified: outFileName = ' // trim(outfilename)
      CALL logmessage(error, scratchmessage)
    END IF

    !----- 5) NetCDF variables ncShuffle, ncDeflate, ncDLevel and others -----
    IF (ncshuffle <= 0) THEN
      ncshuffle = 0
    ELSE
      ncshuffle = 1
    END IF

    IF (ncdeflate <= 0) THEN
      ncdeflate = 0
    ELSE
      ncdeflate = 1
    END IF

    IF (ncdlevel <= 0) THEN
      ncdlevel = 0
    ELSE
      IF (ncdlevel > 9) ncdlevel = 9
    END IF

    ncvarnam_pres = trim(adjustl(ncvarnam_pres))
      IF (len_trim(ncvarnam_pres) == 0) ncvarnam_pres = trim(adjustl(def_ncnam_pres))
    ncvarnam_wndx = trim(adjustl(ncvarnam_wndx))
      IF (len_trim(ncvarnam_wndx) == 0) ncvarnam_wndx = trim(adjustl(def_ncnam_wndx))
    ncvarnam_wndy = trim(adjustl(ncvarnam_wndy))
      IF (len_trim(ncvarnam_wndy) == 0) ncvarnam_wndy = trim(adjustl(def_ncnam_wndy))

    !----- 5) modelType (mandatory variable) -----
    SELECT CASE (modeltype)
      !CASE (1, 2, 3, 4)
      CASE (1)
        ! These are all valid values

      CASE DEFAULT
        errnum = 12

        WRITE(scratchmessage, '("errNum = ", i0, a, i0)') errnum, &
                            '. This model type is not supported: modelType = ', modeltype
        CALL logmessage(error, scratchmessage)
    END SELECT

    !----- 6) various physical parameters -----
    IF ((gravity < 9.76) .OR. (gravity > 9.83)) THEN
```

```
    WRITE(tmpstr1, '(f20.5, a)') gravity
      tmpstr1 = trim(tmpstr1) // ' m/s^2'
    WRITE(tmpstr2, '(f20.5, a)') defv_gravity
      tmpstr2 = trim(tmpstr2) // ' m/s^2'
    WRITE(scratchmessage, '(a)') 'The value of gravity = ' // trim(adjustl(tmpstr1)) // &
                                 ' is adjusted to: gravity = ' // trim(adjustl(tmpstr2))

    CALL logmessage(info, scratchmessage)

    gravity = defv_gravity
  END IF

  IF ((rhowater < 992.0) .OR. (rhowater > 1029.0)) THEN
    WRITE(tmpstr1, '(f20.5, a)') rhowater
      tmpstr1 = trim(tmpstr1) // ' kg/m^3'
    WRITE(tmpstr2, '(f20.5, a)') defv_rhowater
      tmpstr2 = trim(tmpstr2) // ' kg/m^3'
    WRITE(scratchmessage, '(a)') 'The value of rhoWater = ' // trim(adjustl(tmpstr1)) // &
                                 ' is adjusted to: rhoWater = ' // trim(adjustl(tmpstr2))

    CALL logmessage(info, scratchmessage)

    rhowater = defv_rhowater
  END IF

  IF ((rhoair < 1.0) .OR. (rhoair > 1.3)) THEN
    WRITE(tmpstr1, '(f20.5, a)') rhoair
      tmpstr1 = trim(tmpstr1) // ' kg/m^3'
    WRITE(tmpstr2, '(f20.5, a)') defv_rhoair
      tmpstr2 = trim(tmpstr2) // ' kg/m^3'
    WRITE(scratchmessage, '(a)') 'The value of rhoAir = ' // trim(adjustl(tmpstr1)) // &
                                 ' is adjusted to: rhoAir = ' // trim(adjustl(tmpstr2))

    CALL logmessage(info, scratchmessage)

    rhoair = defv_rhoair
  END IF

  IF ((backgroundatmpress < 900.0) .OR. (backgroundatmpress > 1025.0)) THEN
    WRITE(tmpstr1, '(f20.5, a)') backgroundatmpress
      tmpstr1 = trim(tmpstr1) // ' mb'
    WRITE(tmpstr2, '(f20.5, a)') defv_atmpress
      tmpstr2 = trim(tmpstr2) // ' mb'
    WRITE(scratchmessage, '(a)') 'The value of backgroundAtmPress = ' // trim(adjustl(tmpstr1)) // &
                                 ' is adjusted to: backgroundAtmPress = ' // trim(adjustl(tmpstr2))

    CALL logmessage(info, scratchmessage)

    backgroundatmpress = defv_atmpress
  END IF

  IF ((bladjustfac < 0.65) .OR. (bladjustfac > 1.0)) THEN
    WRITE(tmpstr1, '(f20.5)') bladjustfac
    WRITE(tmpstr2, '(f20.5)') defv_bladjustfac
    WRITE(scratchmessage, '(a)') 'The value of blAdjustFac = ' // trim(adjustl(tmpstr1)) // &
                                 ' is adjusted to: blAdjustFac = ' // trim(adjustl(tmpstr2))

    CALL logmessage(info, scratchmessage)

    bladjustfac = defv_bladjustfac
  END IF

  errstatus = errnum

END FUNCTION checkcontrolfileinputs

!================================================================================

!----------------------------------------------------------------
! F U N C T I O N   L O A D  I N T  V A R
!----------------------------------------------------------------
!----------------------------------------------------------------
INTEGER FUNCTION loadintvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)

  IMPLICIT NONE

  INTEGER, INTENT(IN)  :: ninp, nout
  REAL(sz), INTENT(IN) :: vinp(ninp)
  INTEGER, INTENT(OUT) :: vout(nout)

  INTEGER              :: i, ic
```

```
    !-----------------------------------------------------------------------
    !  Load INTEGER variable with input values.
    !-----------------------------------------------------------------------

    ! If not all values are provided for variable, assume the last value
    ! for the rest of the array.
    ic = 0
    IF (ninp <= nout) THEN
      DO i = 1, ninp
        ic = ic + 1
        vout(i) = int(vinp(i))
      END DO
      DO i = ninp + 1, nout
        ic = ic + 1
        vout(i) = int(vinp(ninp))
      END DO
    ELSE
      DO i = 1, nout
        ic = ic + 1
        vout(i) = int(vinp(i))
      END DO
    END IF

    nvalsout = ic

    RETURN

  END FUNCTION loadintvar

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   L O A D  L O G  V A R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION loadlogvar (nInp, vInp, nOut, vOut) RESULT(nValsOut)

    IMPLICIT NONE

    INTEGER, INTENT(IN)           :: ninp, nout
    CHARACTER(LEN=*), INTENT(IN)  :: vinp(ninp)
    LOGICAL, INTENT(OUT)          :: vout(nout)

    INTEGER                       :: i, ic

    !-----------------------------------------------------------------------
    !  Load INTEGER variable with input values.
    !-----------------------------------------------------------------------

    !  If not all values are provided for variable, assume the last value
    !  for the rest of the array.
    ic = 0
    IF (ninp <= nout) THEN
      DO i = 1, ninp
        ic = ic + 1
        IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
          vout(i) = .true.
        ELSE
          vout(i) = .false.
        END IF
      END DO
      DO i = ninp + 1, nout
        ic = ic + 1
        IF ((vinp(ninp)(1:1) == 'T') .OR. (vinp(ninp)(1:1) == 't')) THEN
          vout(i) = .true.
        ELSE
          vout(i) = .false.
        END IF
      END DO
    ELSE
      DO i = 1, nout
        ic = ic + 1
        IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
          vout(i) = .true.
        ELSE
          vout(i) = .false.
        END IF
      END DO
    END IF

```

```
    nvalsout = ic

    RETURN

  END FUNCTION loadlogvar

!==============================================================================

  !------------------------------------------------------------------
  ! F U N C T I O N   L O A D  R E A L  V A R
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  INTEGER FUNCTION loadrealvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)

    IMPLICIT NONE

    INTEGER, INTENT(IN)    :: ninp, nout
    REAL(sz), INTENT(IN)   :: vinp(ninp)
    REAL(sz), INTENT(OUT)  :: vout(nout)

    INTEGER                :: i, ic

    !---------------------------------------------------------------------
    !  Load INTEGER variable with input values.
    !---------------------------------------------------------------------

    !  If not all values are provided for variable, assume the last value
    !  for the rest of the array.
    ic = 0
    IF (ninp <= nout) THEN
      DO i = 1, ninp
        ic = ic + 1
        vout(i) = vinp(i)
      END DO
      DO i = ninp + 1, nout
        ic = ic + 1
        vout(i) = vinp(ninp)
      END DO
    ELSE
      DO i = 1, nout
        ic = ic + 1
        vout(i) = vinp(i)
      END DO
    END IF

    nvalsout = ic

    RETURN

  END FUNCTION loadrealvar

!==============================================================================

  !------------------------------------------------------------------
  ! F U N C T I O N   T O  L O W E R  C A S E
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  PURE FUNCTION tolowercase(inpString) RESULT(outString)

    IMPLICIT NONE

    CHARACTER(*), INTENT(IN)   :: inpstring

    INTEGER, PARAMETER         :: duc = ichar('A') - ichar('a')
    CHARACTER(LEN(inpString)) :: outstring
    CHARACTER                  :: ch
    INTEGER                    :: i

    DO i = 1, len(inpstring)
      ch = inpstring(i:i)
      IF ((ch >= 'A') .AND. (ch <= 'Z')) ch = char(ichar(ch) - duc)
      outstring(i:i) = ch
    END DO

    RETURN

  END FUNCTION tolowercase

!==============================================================================

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

```
  ! F U N C T I O N   T O  U P P E R  C A S E
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  PURE FUNCTION touppercase(inpString) RESULT(outString)

    IMPLICIT NONE

    CHARACTER(*), INTENT(IN)  :: inpstring

    INTEGER, PARAMETER        :: duc = ichar('A') - ichar('a')
    CHARACTER(LEN(inpString)) :: outstring
    CHARACTER                 :: ch
    INTEGER                   :: i

    DO i = 1, len(inpstring)
      ch = inpstring(i:i)
      IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
      outstring(i:i) = ch
    END DO

    RETURN

  END FUNCTION touppercase

!================================================================================

  !------------------------------------------------------------------
  ! F U N C T I O N   C O N V  L O N
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  REAL(sz) function convlon(inplon) result (myvalout)

    IMPLICIT NONE

    REAL(sz) :: inplon

    myvalout = mod(inplon + 180.0_sz, 360.0_sz) - 180.0_sz

    RETURN

  END FUNCTION convlon

!================================================================================

  !------------------------------------------------------------------
  !  S U B R O U T I N E   G E O  TO  C P P  S C A L A R
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  SUBROUTINE geotocpp_scalar(lat, lon, lat0, lon0, x, y)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)  :: lat
    REAL(SZ), INTENT(IN)  :: lon
    REAL(SZ), INTENT(IN)  :: lat0
    REAL(SZ), INTENT(IN)  :: lon0
    REAL(SZ), INTENT(OUT) :: x
    REAL(SZ), INTENT(OUT) :: y

    x = deg2rad * rearth * (lon - lon0) * cos(lat0)
    y = deg2rad * rearth * lat

  END SUBROUTINE geotocpp_scalar

!================================================================================

  !------------------------------------------------------------------
  !  S U B R O U T I N E   G E O  TO  C P P  1D
  !------------------------------------------------------------------
  !------------------------------------------------------------------
  SUBROUTINE geotocpp_1d(lat, lon, lat0, lon0, x, y)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)  :: lat(:)
    REAL(SZ), INTENT(IN)  :: lon(:)
    REAL(SZ), INTENT(IN)  :: lat0
```

```
    REAL(SZ), INTENT(IN)  :: lon0
    REAL(SZ), INTENT(OUT) :: x(:)
    REAL(SZ), INTENT(OUT) :: y(:)

    x = deg2rad * rearth * (lon - lon0) * cos(lat0)
    y = deg2rad * rearth * lat

  END SUBROUTINE geotocpp_1d

!================================================================================

  !----------------------------------------------------------------------
  !  S U B R O U T I N E   C P P  T O  G E O  S C A L A R
  !----------------------------------------------------------------------
  !----------------------------------------------------------------------
  SUBROUTINE cpptogeo_scalar(x, y, lat0, lon0, lat, lon)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)   :: x
    REAL(SZ), INTENT(IN)   :: y
    REAL(SZ), INTENT(IN)   :: lat0
    REAL(SZ), INTENT(IN)   :: lon0
    REAL(SZ), INTENT(OUT)  :: lat
    REAL(SZ), INTENT(OUT)  :: lon

    lat = y / (deg2rad * rearth)
    lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))

  END SUBROUTINE cpptogeo_scalar

!================================================================================

  !----------------------------------------------------------------------
  !  S U B R O U T I N E   C P P  T O  G E O  1D
  !----------------------------------------------------------------------
  !----------------------------------------------------------------------
  SUBROUTINE cpptogeo_1d(x, y, lat0, lon0, lat, lon)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)   :: x(:)
    REAL(SZ), INTENT(IN)   :: y(:)
    REAL(SZ), INTENT(IN)   :: lat0
    REAL(SZ), INTENT(IN)   :: lon0
    REAL(SZ), INTENT(OUT)  :: lat(:)
    REAL(SZ), INTENT(OUT)  :: lon(:)

    lat = y / (deg2rad * rearth)
    lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))

  END SUBROUTINE cpptogeo_1d

!================================================================================

  ! ----------------------------------------------------------------------
  !  F U N C T I O N   S P H E R I C A L   D I S T A N C E
  ! ----------------------------------------------------------------------
  !----------------------------------------------------------------------
  REAL(sz) function sphericaldistance_scalar(lat1, lon1, lat2, lon2) result(myvalout)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: lat1    ! latitude of point 1 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon1    ! longitude of point 1 on the sphere (degrees east)
    REAL(sz), INTENT(IN) :: lat2    ! latitude of point 2 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon2    ! longitude of point 2 on the sphere (degrees east)

    REAL(sz)             :: phi1, phi2, lamda1, lamda2, dphi, dlamda, dsigma

    phi1   = deg2rad * lat1
    phi2   = deg2rad * lat2
    dphi   = abs(phi2 - phi1)

    lamda1 = deg2rad * lon1
```

```
    lamda2 = deg2rad * lon2
    dlamda = abs(lamda2 - lamda1)

    ! Vincenty formula to calculate a distance along a sphere
    dsigma = atan(sqrt((cos(phi2) * sin(dlamda))**2 + &
                       (cos(phi1) * sin(phi2) - sin(phi1) * cos(phi2) * cos(dlamda))**2))
    dsigma = dsigma / (sin(phi1) * sin(phi2) + cos(phi1) * cos(phi2) * cos(dlamda))

    ! This is the great-circle distance; REARTH in meters
    myvalout = rearth * dsigma

    RETURN

  END FUNCTION sphericaldistance_scalar

!================================================================================

  ! ----------------------------------------------------------------
  !  F U N C T I O N   S P H E R I C A L   D I S T A N C E _ 1 D
  ! ----------------------------------------------------------------
  !----------------------------------------------------------------
  FUNCTION sphericaldistance_1d(lats, lons, lat0, lon0) RESULT(myValOut)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    ! Global variables
    REAL(sz), INTENT(IN) :: lats(:) ! latitude of point 1 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lons(:) ! longitude of point 1 on the sphere (degrees east)
    REAL(sz), INTENT(IN) :: lat0    ! latitude of point 2 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon0    ! longitude of point 2 on the sphere (degrees east)

    REAL(sz), DIMENSION(:), ALLOCATABLE :: myvalout

    ! Local variables
    REAL(sz), DIMENSION(:), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
    REAL(sz)                            :: phi0, lamda0
    INTEGER                             :: status, n1


    CALL setmessagesource("SphericalDistance_1D")

    IF (SIZE(lats) /= SIZE(lons)) THEN
      WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
      CALL allmessage(error, scratchmessage)

      CALL terminate()
    END IF

    n1 = SIZE(lats, 1)
    ALLOCATE(myvalout(n1), stat = status)
    ALLOCATE(phis(n1), lamdas(n1), dphi(n1), dlamda(n1), dsigma(n1), stat = status)

    IF (status /= 0) THEN
      WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
      CALL allmessage(error, scratchmessage)

      CALL terminate()
    END IF

    phis   = deg2rad * lats
    phi0   = deg2rad * lat0
    dphi   = abs(phi0 - phis)

    lamdas = deg2rad * lons
    lamda0 = deg2rad * lon0
    dlamda = abs(lamda0 - lamdas)

    ! Vincenty formula to calculate a distance along a sphere
    dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
                       (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
    dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))

    ! This is the great-circle distance; REARTH in meters
    myvalout = rearth * dsigma

    DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)

    CALL unsetmessagesource()

```

```
    RETURN

  END FUNCTION sphericaldistance_1d

!================================================================================

  ! ----------------------------------------------------------------
  !  F U N C T I O N   S P H E R I C A L   D I S T A N C E _ 2 D
  ! ----------------------------------------------------------------
  !----------------------------------------------------------------
  FUNCTION sphericaldistance_2d(lats, lons, lat0, lon0) RESULT(myValOut)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    ! Global variables
    REAL(sz), INTENT(IN) :: lats(:, :) ! latitude of point 1 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lons(:, :) ! longitude of point 1 on the sphere (degrees east)
    REAL(sz), INTENT(IN) :: lat0       ! latitude of point 2 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon0       ! longitude of point 2 on the sphere (degrees east)

    REAL(sz), DIMENSION(:, :), ALLOCATABLE :: myvalout

    ! Local variables
    REAL(sz), DIMENSION(:, :), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
    REAL(sz)                               :: phi0, lamda0
    INTEGER                                :: status, n1, n2


    CALL setmessagesource("SphericalDistance_2D")

    IF (SIZE(lats) /= SIZE(lons)) THEN
      WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      CALL terminate()
    END IF

    n1 = SIZE(lats, 1)
    n2 = SIZE(lats, 2)
    ALLOCATE(myvalout(n1, n2), stat = status)
    ALLOCATE(phis(n1, n2), lamdas(n1, n2), dphi(n1, n2), dlamda(n1, n2), dsigma(n1, n2), stat = status)

    IF (status /= 0) THEN
      WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      CALL terminate()
    END IF

    phis   = deg2rad * lats
    phi0   = deg2rad * lat0
    dphi   = abs(phi0 - phis)

    lamdas = deg2rad * lons
    lamda0 = deg2rad * lon0
    dlamda = abs(lamda0 - lamdas)

    ! Vincenty formula to calculate a distance along a sphere
    dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
                       (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
    dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))

    ! This is the great-circle distance; REARTH in meters
    myvalout = rearth * dsigma

    DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)

    CALL unsetmessagesource()

    RETURN

  END FUNCTION sphericaldistance_2d

!================================================================================

```

```
  ! ----------------------------------------------------------------
  !  F U N C T I O N   S P H E R I C A L   D I S T A N C E  H A R V
  ! ----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sz) function sphericaldistanceharv(lat1, lon1, lat2, lon2) result(myvalout)

    USE pahm_global, ONLY : rearth, deg2rad

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: lat1    ! latitude of point 1 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon1    ! longitude of point 1 on the sphere (degrees east)
    REAL(sz), INTENT(IN) :: lat2    ! latitude of point 2 on the sphere (degrees north)
    REAL(sz), INTENT(IN) :: lon2    ! longitude of point 2 on the sphere (degrees east)

    REAL(sz)             :: phi1, phi2, lamda1, lamda2, dphi, dlamda, dsigma

    phi1   = deg2rad * lat1
    phi2   = deg2rad * lat2
    dphi   = abs(phi2 - phi1)

    lamda1 = deg2rad * lon1
    lamda2 = deg2rad * lon2
    dlamda = abs(lamda2 - lamda1)

    ! Haversine formula formula to calculate a distance along a sphere
    dsigma = sqrt(sin(dphi / 2.0_sz)**2 + cos(phi1) * cos(phi2) * sin(dlamda / 2.0_sz)**2)
    dsigma = 2.0_sz * asin(dsigma)

    ! This is the great-circle distance; REARTH in meters
    myvalout = rearth * dsigma

    RETURN

  END FUNCTION sphericaldistanceharv

!================================================================================

!DEL ! ----------------------------------------------------------------
!DEL !  F U N C T I O N   S P H E R I C A L   D I S T A N C E  A D C I R C
!DEL ! ----------------------------------------------------------------
!DEL !  jgf49.1001 PV to be deleted
!DEL !> Function to get the distance along the surface of
!DEL !> a sphere (the earth's surface in this case).
!DEL ! ----------------------------------------------------------------
!DEL REAL(SZ) FUNCTION SphericalDistanceADCIRC(dx, dy, y1, y2) RESULT(myValOut)

!DEL   USE PaHM_Global, ONLY : REARTH, DEG2RAD

!DEL   IMPLICIT NONE

!DEL   REAL(SZ), INTENT(IN) :: dx    ! longitude distance in radians
!DEL   REAL(SZ), INTENT(IN) :: dy    ! latitude distance in radians
!DEL   REAL(SZ), INTENT(IN) :: y1    ! degrees latitude of starting point
!DEL   REAL(SZ), INTENT(IN) :: y2    ! degrees latitude of ending point

!DEL   ! compute the distances based on haversine formula for
!DEL   ! distance along a sphere
!DEL   myValOut = SQRT(SIN(dy / 2.0_SZ)**2 +                                    &
!DEL                   COS(y1 * DEG2RAD) * COS(y2 * DEG2RAD) * SIN(dx / 2.0_SZ)**2)

!DEL   ! This is the great-circle distance; REARTH in meters
!DEL   myValOut = REARTH * (2.0_SZ * ASIN(myValOut))

!DEL   RETURN

!DEL END FUNCTION SphericalDistanceADCIRC

!DEL================================================================================

  ! ----------------------------------------------------------------
  !  S U B R O U T I N E   S P H E R I C A L  F R A C  P O I N T
  ! ----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE sphericalfracpoint(lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)

    USE pahm_global, ONLY : rearth, deg2rad, rad2deg

    IMPLICIT NONE

    ! Global variables
```

```
    REAL(SZ), INTENT(IN)            :: lat1        ! latitude of point 1 on the sphere (degrees north)
    REAL(SZ), INTENT(IN)            :: lon1        ! longitude of point 1 on the sphere (degrees east)
    REAL(SZ), INTENT(IN)            :: lat2        ! latitude of point 2 on the sphere (degrees north)
    REAL(SZ), INTENT(IN)            :: lon2        ! longitude of point 2 on the sphere (degrees east)
    REAL(SZ), INTENT(IN)            :: fraction    ! distance fraction of the indermediate point (0 <= f <= 1)
    REAL(SZ), INTENT(OUT)           :: latf, lonf  ! the calculated latitude and longitude of the
                                                   ! intermediate point
    REAL(SZ), OPTIONAL, INTENT(OUT) :: distf       ! the distance between point 1 and the intermediate point
    REAL(SZ), OPTIONAL, INTENT(OUT) :: dist12      ! the distance between point 1 and point 2

    ! Local variables
    REAL(SZ)                        :: myFrac
    REAL(SZ)                        :: phi1, phi2, lamda1, lamda2, delta
    REAL(SZ)                        :: aa, bb, xx, yy, zz
    REAL(SZ) :: myDist12, myDistF


    myfrac = fraction
    IF (myfrac < 0) myfrac = 0.0_sz
    IF (myfrac > 1) myfrac = 1.0_sz

    ! Calculate the great circle distance between points 1 and 2
    mydist12 = sphericaldistance(lat1, lon1, lat2, lon2)

    ! Distance is in meters (REARTH in meters). If myDist12 < 0.01_SZ
    ! the two points are coincident
    IF (mydist12 < 0.01_sz) THEN
      latf   = lat1
      lonf   = lon1
      IF (PRESENT(distf))  distf  = 0.0_sz
      IF (PRESENT(dist12)) dist12 = 0.0_sz

      RETURN
    END IF

    phi1   = deg2rad * lat1
    phi2   = deg2rad * lat2
    lamda1 = deg2rad * lon1
    lamda2 = deg2rad * lon2

    delta = mydist12 / rearth

    aa = sin((1.0_sz - myfrac) * delta) / sin(delta)
    bb = sin(myfrac * delta) / sin(delta)

    xx = aa * cos(phi1) * cos(lamda1) + bb * cos(phi2) * cos(lamda2)
    yy = aa * cos(phi1) * sin(lamda1) + bb * cos(phi2) * sin(lamda2)
    zz = aa * sin(phi1) + bb * sin(phi2)

    ! The (lat, lon) values of the intermidiate point
    latf = rad2deg * atan2(zz, sqrt(xx * xx + yy * yy))
    lonf = rad2deg * atan2(yy, xx)

    ! This is the great-circle distance; REARTH in meters
    mydistf  = sphericaldistance(lat1, lon1, latf, lonf)

    IF (PRESENT(distf))  distf  = mydistf
    IF (PRESENT(dist12)) dist12 = mydist12

    RETURN

  END SUBROUTINE sphericalfracpoint

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   G E T  L O C  A N D  R A T I O
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE getlocandratio(val, arrVal, idx1, idx2, wtRatio)

    IMPLICIT NONE

    ! Global variables
    REAL(SZ), INTENT(IN)  :: val          ! value to search for
    REAL(SZ), INTENT(IN)  :: arrVal(:)    ! search array (1D)
    INTEGER, INTENT(OUT)  :: idx1         ! the index of the lowest bound
    INTEGER, INTENT(OUT)  :: idx2         ! the index of the highest bound
    REAL(SZ), INTENT(OUT) :: wtRatio      ! the ratio factor that used in the linear interpolation
```

```
                                         ! calculations: F = F(idx1) + wtRatio * (F(idx2) - F(idx1))
                                         ! 0 <= wtRatio <= 1.0

    ! Local variables
    INTEGER                  :: nn, jl, jl1, jl2
    REAL(SZ)                 :: diffVal


    idx1 = -1
    idx2 = -1
    wtratio = 0.0_sz

    nn = SIZE(arrval, 1)
    jl = minloc(abs(val - arrval), 1)

    !---------- Check if we got an exact bin value
    IF (comparereals(val - arrval(jl), 0.0_sz, 0.0001_sz) == 0) THEN
      idx1 = jl
      idx2 = jl
      wtratio = 0.0_sz

      RETURN
    END IF
    !----------

    !---------- Checking the values at the two edges of the arrVal
    IF ((jl == 1) .OR. (jl == nn)) THEN
      IF (jl == 1) THEN
        jl1 = jl
        jl2 = jl + 1
      ELSE
        jl1 = jl - 1
        jl2 = jl
      END IF

      diffval = arrval(jl2) - arrval(jl1)

      IF (comparereals(diffval, 0.0_sz, 0.0001_sz) == 0) THEN
        idx1 = jl1
        idx2 = jl1
        wtratio = 0.0_sz
      ELSE
        IF (comparereals(val - arrval(jl1), 0.0_sz) * &
            comparereals(val - arrval(jl2), 0.0_sz) < 0) THEN
          idx1 = jl1
          idx2 = jl2
          wtratio = (val - arrval(jl1)) / diffval
        END IF
      END IF

      RETURN
    END IF
    !----------

    IF (comparereals(val - arrval(jl - 1), 0.0_sz) * &
        comparereals(val - arrval(jl), 0.0_sz) < 0) THEN
      jl1 = jl - 1
      jl2 = jl

      diffval = arrval(jl2) - arrval(jl1)

      idx1 = jl1
      idx2 = jl2
      wtratio = (val - arrval(jl1)) / diffval
    ELSE IF (comparereals(val - arrval(jl), 0.0_sz) * &
             comparereals(val - arrval(jl + 1), 0.0_sz) < 0) THEN

      jl1 = jl
      jl2 = jl + 1

      diffval = arrval(jl2) - arrval(jl1)

      idx1 = jl1
      idx2 = jl2
      wtratio = (val - arrval(jl1)) / diffval
    END IF

    RETURN

  END SUBROUTINE getlocandratio

```

```
!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   C H A R  U N I Q U E
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION charunique(inpVec, outVec, idxVec) RESULT (myRec)

    IMPLICIT NONE

    CHARACTER(LEN=*), INTENT(IN)                :: inpvec(:)
    CHARACTER(LEN=*), INTENT(OUT)               :: outvec(:)
    INTEGER, ALLOCATABLE, INTENT(OUT)           :: idxvec(:)

    CHARACTER(LEN=LEN(inpVec(1))), ALLOCATABLE :: chkstr(:)
    INTEGER, ALLOCATABLE                        :: chkint(:)
    INTEGER :: nels
    INTEGER :: icnt, jcnt   ! counters


    nels = SIZE(inpvec, 1)

    ALLOCATE(chkstr(nels))
    ALLOCATE(chkint(nels))


    jcnt = 1
    DO icnt = 1, nels
      IF (trim(inpvec(icnt)) == '')    cycle
      IF (any(chkstr == inpvec(icnt))) cycle

      ! No match found so add it to the output
      chkstr(jcnt) = inpvec(icnt)
      chkint(jcnt) = icnt
      jcnt = jcnt + 1
    END DO

    myrec  = jcnt - 1
    outvec = chkstr
    idxvec = chkint

    DEALLOCATE(chkstr)
    DEALLOCATE(chkint)

    RETURN

  END FUNCTION charunique

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   V A L  S T R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sp) function valstr(string) result(myval)

    IMPLICIT NONE

    ! Dummy arguments
    CHARACTER(LEN=*), INTENT(IN) :: string

    ! Local variables
    INTEGER  :: i
    REAL(sp) :: v

    i = realscan(string,1,v)
    myval = v

    RETURN

  END FUNCTION valstr

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   D  V A L  S T R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(hp) function dvalstr(string) result(myval)

    IMPLICIT NONE
```

```

    ! Dummy arguments
    CHARACTER(LEN=*), INTENT(IN) :: string

    ! Local variables
    INTEGER  :: i
    REAL(hp) :: v

    i = drealscan(string,1,v)
    myval = v

    RETURN

  END FUNCTION dvalstr

!==================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   I N T  V A L  S T R
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION intvalstr(String) Result(myVal)

    IMPLICIT NONE

    ! Dummy arguments
    CHARACTER(LEN=*), INTENT(IN) :: string

    ! Local variables
    INTEGER :: i
    INTEGER :: v

    i = intscan(string,1,.true.,v)
    myval = v

    RETURN

  END FUNCTION intvalstr

!==================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   R E A L  S C A N
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION realscan(String, Pos, Value) Result(myVal)

    IMPLICIT NONE

    ! Dummy arguments
    INTEGER, INTENT(IN)          :: pos
    CHARACTER(LEN=*), INTENT(IN) :: string
    REAL(sp), INTENT(OUT)        :: value

    ! Local variables
    INTEGER :: fract, intg, kfract, pmsign, power, ptr

    ! CHECK POS.
    myval = pos
    Value = 0.0
    IF(pos < 1 .OR. len(string) < pos)RETURN

    ! SET UP WORKING VARIABLES.
    intg = 0
    fract = 0
    kfract = 0
    power = 0
    DO WHILE (.true.)
       ! SKIP LEADING BLANKS.
       IF(string(myval:myval) == ' ') THEN
          myval = myval + 1
          IF(myval > len(string))RETURN
          cycle
       END IF

       ! LOOK FOR SIGN.
       ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
       pmsign = 0
       IF(string(myval:myval) == '+') THEN
          pmsign = +1
       ELSE IF(string(myval:myval) == '-') THEN
```

```
          pmsign = -1
       END IF
       IF(pmsign.NE.0)myval = myval + 1

       ! LOOK FOR INTEGER PART.
       myval = intscan(string,myval,.false.,intg)

       ! LOOK FOR FRACTION PART.
       IF(myval.LE.len(string)) THEN
          IF(myval > pos+abs(pmsign)) THEN
             ! DETERMINE IF FIRST FORM OR SECOND FORM.
             ! HANDLE FIRST FORM:  D+ ['.' D*]
             IF(string(myval:myval) == '.') THEN
                myval = myval + 1
                IF(myval.LE.len_trim(string)) THEN
                   IF(string(myval:myval).NE.' ') THEN
                      ptr = intscan(string,myval,.false.,fract)
                      kfract = ptr - myval
                      myval = ptr
                   END IF
                END IF
             END IF
          ! HANDLE SECOND FORM:  '.' D+
          ELSE IF(string(myval:myval).NE.'.') THEN
             ! IF '.' MISSING, THEN WE HAVE NOTHING.
             myval = pos
             RETURN
          ELSE
             myval = myval + 1
             ptr = intscan(string,myval,.false.,fract)
             kfract = ptr - myval
             IF(kfract == 0) THEN
                ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
                myval = pos
                RETURN
             ELSE
                myval = ptr
             END IF
          END IF

          ! LOOK FOR EXPONENT PART.
          IF(myval.LE.len(string)) THEN
             IF((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e')) THEN
                myval = myval + 1
                ptr = intscan(string,myval,.true.,power)
                IF(ptr == myval) THEN
                   ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
                   ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
                   ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
                   myval = myval - 1
                   Value = intg + fract/10.0**kfract
                   IF(pmsign == -1)Value = -Value
                   RETURN
                ELSE
                   myval = ptr
                END IF
             END IF
          END IF
       END IF

       ! COMPUTE REAL VALUE FROM ITS PARTS.
       IF(kfract.NE.0) THEN
          Value = (intg+fract/10.0**kfract)*10.0**power
       ELSE
          Value = intg*10.0**power
       END IF
       IF(pmsign == -1)Value = -Value
       EXIT
    END DO

    RETURN

  END FUNCTION realscan

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   D R E A L  S C A N
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION drealscan(String,Pos,Value) RESULT(myVal)
```

```
    IMPLICIT NONE

    ! Dummy arguments
    INTEGER, INTENT(IN)          :: pos
    CHARACTER(LEN=*), INTENT(IN) :: string
    REAL(hp), INTENT(OUT)        :: value

    ! Local variables
    INTEGER :: fract, intg, kfract, pmsign, power, ptr

    ! CHECK POS.
    myval = pos
    Value = 0.0
    IF(pos < 1 .OR. len(string) < pos)RETURN

    ! SET UP WORKING VARIABLES.
    intg = 0
    fract = 0
    kfract = 0
    power = 0
    DO WHILE (.true.)
       ! SKIP LEADING BLANKS.
       IF(string(myval:myval) == ' ') THEN
          myval = myval + 1
          IF(myval > len(string))RETURN
          cycle
       END IF

       ! LOOK FOR SIGN.
       ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
       pmsign = 0
       IF(string(myval:myval) == '+') THEN
          pmsign = +1
       ELSE IF(string(myval:myval) == '-') THEN
          pmsign = -1
       END IF
       IF(pmsign.NE.0)myval = myval + 1

       ! LOOK FOR INTEGER PART.
       myval = intscan(string,myval,.false.,intg)

       ! LOOK FOR FRACTION PART.
       IF(myval.LE.len(string)) THEN
          IF(myval > pos+abs(pmsign)) THEN
             ! DETERMINE IF FIRST FORM OR SECOND FORM.
             ! HANDLE FIRST FORM:  D+ ['.' D*]
             IF(string(myval:myval) == '.') THEN
                myval = myval + 1
                IF(myval.LE.len_trim(string)) THEN
                   IF(string(myval:myval).NE.' ') THEN
                      ptr = intscan(string,myval,.false.,fract)
                      kfract = ptr - myval
                      myval = ptr
                   END IF
                END IF
             END IF
          ! HANDLE SECOND FORM:  '.' D+
          ELSE IF(string(myval:myval).NE.'.') THEN
             ! IF '.' MISSING, THEN WE HAVE NOTHING.
             myval = pos
             RETURN
          ELSE
             myval = myval + 1
             ptr = intscan(string,myval,.false.,fract)
             kfract = ptr - myval
             IF(kfract == 0) THEN
                ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
                myval = pos
                RETURN
             ELSE
                myval = ptr
             END IF
          END IF

          ! LOOK FOR EXPONENT PART.
          IF(myval.LE.len(string)) THEN
             IF((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e') .OR. &
                (string(myval:myval) == 'D') .OR. (string(myval:myval) == 'd')) THEN
                myval = myval + 1
                ptr = intscan(string,myval,.true.,power)
```

```
                IF(ptr == myval) THEN
                   ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
                   ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
                   ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
                   myval = myval - 1
                   Value = intg + fract/10.0**kfract
                   IF(pmsign == -1)Value = -Value
                   RETURN
                ELSE
                   myval = ptr
                END IF
             END IF
          END IF
       END IF

       ! COMPUTE REAL VALUE FROM ITS PARTS.
       IF(kfract.NE.0) THEN
          Value = (intg+fract/10.0**kfract)*10.0**power
       ELSE
          Value = intg*10.0**power
       END IF
       IF(pmsign == -1)Value = -Value
       EXIT
    END DO

    RETURN

    END FUNCTION drealscan

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   I N T  S C A N
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  INTEGER FUNCTION intscan(String, Pos, Signed, Value) Result(myVal)

    IMPLICIT NONE

    ! Dummy arguments
    INTEGER, INTENT(IN)           :: pos
    LOGICAL, INTENT(IN)           :: signed
    CHARACTER(LEN=*), INTENT(IN)  :: string
    INTEGER, INTENT(OUT)          :: value

    ! Local variables
    INTEGER(KIND=4) :: digit,pmsign

    ! CHECK POS.
    myval = pos
    Value = 0
    IF(pos < 1 .OR. len(string) < pos)RETURN
    DO WHILE (.true.)

       ! SKIP LEADING BLANKS.
       IF(string(myval:myval) == ' ') THEN
          myval = myval + 1
          IF(myval > len(string))RETURN
          cycle
       END IF

       ! IF SIGNED, CHECK FOR SIGN.
       pmsign = 0
       IF(signed) THEN
          IF(string(myval:myval) == '+') THEN
             pmsign = +1
          ELSE IF(string(myval:myval) == '-') THEN
             pmsign = -1
          END IF
          IF(pmsign.NE.0)myval = myval + 1

          ! IF sign is the last char in the field (with no integer following it)
          ! myVal value is left as POS or at the end of leading blanks.
          IF(myval > len_trim(string)) THEN
             myval = myval - 1
             RETURN
          END IF
       END IF

       ! PROCESS DIGIT STRING.
       DO myval = myval,len(string)
```

```
            digit = ichar(string(myval:myval)) - ichar('0')
            IF(digit < 0 .OR. 9 < digit) GO TO 10
            Value = Value*10 + digit
        END DO
        ! Explicitly defined intscn to avoid possible compiler dependences (TWB. 930223)
        myval = len(string) + 1
        EXIT
      END DO

      ! ADJUST SIGN.
      10 IF(signed.AND.pmsign == -1)Value = -Value

      RETURN

      END FUNCTION intscan

!==============================================================================

END MODULE utilities

```

9.40 vortex.F90 File Reference

Modules

- module pahm_vortex

Functions/Subroutines

- subroutine pahm_vortex::calcintensitychange (var, times, calcInt, status, order)
  This subroutine calculates the intensity time change of a variable using second order mumerical accuracy and uneven spacing.
- subroutine pahm_vortex::uvtrans (lat, lon, times, u, v, status, order)
  This subroutine calculates the translational velocity of a moving hurricane using second order mumerical accuracy and uneven spacing.
- subroutine pahm_vortex::uvtranspoint (lat1, lon1, lat2, lon2, time1, time2, u, v)
  This subroutine calculates the translational velocity of a moving hurricane.
- subroutine pahm_vortex::newvortex (pinf, p0, lat, lon, vm)
  Create a new Vortex object.
- subroutine pahm_vortex::newvortexfull (pinf, p0, lat, lon, vm)
  A new vortex is created for the full gradient wind balance.
- subroutine pahm_vortex::setvortex (pinf, p0, lat, lon)
  Set basic parameter for a new Vortex object.
- subroutine pahm_vortex::setrmaxes (rMaxW)
- subroutine pahm_vortex::getrmaxes (rMaxW)
- subroutine pahm_vortex::calcrmaxes ()
  Calculate the radius of maximum winds for all storm quadrants.
- subroutine pahm_vortex::calcrmaxesfull ()
  Calculate the radius of maximum winds for all storm quadrants. Solving the full gradient wind equation without the assumption of cyclostrohpic balance.
- subroutine pahm_vortex::fitrmaxes ()
  Calculates the coefficients that fit the given radius of maximum winds for all storm quadrants.
- subroutine pahm_vortex::fitrmaxes4 ()
- subroutine pahm_vortex::setvmaxesbl (vMaxW)
- subroutine pahm_vortex::getvmaxesbl (vMaxW)

- subroutine pahm_vortex::setusevmaxesbl (u)
- subroutine pahm_vortex::setshapeparameter (param)
- real(sz) function pahm_vortex::getshapeparameter ()
- real(sz) function, dimension(4) pahm_vortex::getshapeparameters ()
- real(sz) function, dimension(4) pahm_vortex::getphifactors ()
- subroutine pahm_vortex::setisotachradii (ir)
- subroutine pahm_vortex::setisotachwindspeeds (vrQ)
- subroutine pahm_vortex::setusequadrantvr (u)
- logical function pahm_vortex::getusequadrantvr ()
- real(sz) function pahm_vortex::spinterp (angle, dist, opt)

  Spatial Interpolation function based on angle and r.
- real(sz) function pahm_vortex::interpr (quadVal, quadSel, quadDis)
- real(sz) function pahm_vortex::rmw (angle)

  Calculate the radius of maximum winds.
- subroutine pahm_vortex::uvp (lat, lon, uTrans, vTrans, u, v, p)

  Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.
- subroutine pahm_vortex::uvpr (iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, uTrans, vTrans, geof, u, v, p)

  Calculate (u, v) wind components and surface pressure from an asymmetric hurricane wind model.
- real(sz) function pahm_vortex::fang (r, rmx)

  Compute a wind angle to parameterize frictional inflow across isobars.
- subroutine pahm_vortex::rotate (x, y, angle, whichWay, xr, yr)

  Rotate a 2D vector (x, y) by an angle.
- real(sz) function pahm_vortex::getlatestrmax ()
- real(sz) function pahm_vortex::getlatestangle ()
- real(sz) function pahm_vortex::vhwithcorifull (testRMax)

  External function f(x) = 0 for which a root is sought using Brent's root-finding method.
- real(sz) function pahm_vortex::vhwithcori (testRMax)

  External function f(x) = 0 for which a root is sought using Brent's root-finding method.
- real(sz) function pahm_vortex::vhnocori (testRMax)
- real(sz) function pahm_vortex::findroot (func, x1, x2, dx, a, b)

  Use brute-force marching to find a root the interval [x1,x2].

Variables

- integer, parameter pahm_vortex::nquads = 4
- integer, parameter pahm_vortex::npoints = NQUADS + 2
- real(sz), dimension(npoints) pahm_vortex::rmaxes
- real(sz), dimension(npoints, 4) pahm_vortex::rmaxes4
- real(sz) pahm_vortex::pn
- real(sz) pahm_vortex::pc
- real(sz) pahm_vortex::clat
- real(sz) pahm_vortex::clon
- real(sz) pahm_vortex::vmax
- real(sz) pahm_vortex::b
- real(sz) pahm_vortex::corio
- real(sz) pahm_vortex::vr
- real(sz) pahm_vortex::phi
- real(sz), dimension(npoints) pahm_vortex::phis

- real(sz), dimension(npoints, 4) pahm_vortex::phis4
- real(sz), dimension(npoints) pahm_vortex::bs
- real(sz), dimension(npoints, 4) pahm_vortex::bs4
- real(sz), dimension(npoints) pahm_vortex::vmbl
- real(sz), dimension(npoints, 4) pahm_vortex::vmbl4
- integer, dimension(npoints, 4) pahm_vortex::quadflag4
- real(sz), dimension(npoints, 4) pahm_vortex::quadir4
- real(sz), dimension(nquads) pahm_vortex::vrquadrant
- real(sz), dimension(nquads) pahm_vortex::radius
- integer pahm_vortex::quad
- real(sz) pahm_vortex::latestrmax
- real(sz) pahm_vortex::latestangle
- logical pahm_vortex::usequadrantvr
- logical pahm_vortex::usevmaxesbl

9.40.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Note

Adopted from the ADCIRC source code.

Definition in file vortex.F90.

9.41 vortex.F90

Go to the documentation of this file.

```
!----------------------------------------------------------------
!               M O D U L E   V O R T E X
!----------------------------------------------------------------
!----------------------------------------------------------------

MODULE pahm_vortex

  USE pahm_sizes
  USE pahm_messages

  IMPLICIT NONE
  SAVE

  INTEGER, PARAMETER               :: nquads  = 4           ! Number of quadrants for which wind radii are
 provided
  INTEGER, PARAMETER               :: npoints = nquads + 2  ! Number of (theta, rMax) points for curve fit
  REAL(sz), DIMENSION(NPOINTS)     :: rmaxes                ! Radius of maximum winds
  REAL(sz), DIMENSION(NPOINTS, 4)  :: rmaxes4               ! (nautical miles)

  REAL(sz)                         :: pn                    ! Ambient surface pressure (mb) !PV global
 var?
  REAL(sz)                         :: pc                    ! Surface pressure at center of storm (mb) !PV
 global var?
  REAL(sz)                         :: clat                  ! Latitude  of storm center (degrees north)
 !PV global var?
  REAL(sz)                         :: clon                  ! Longitude of storm center (degrees east )
 !PV global var?
```

```
REAL(sz)                          :: vmax                  ! Max sustained wind velocity in storm (knots) !PV global var?

REAL(sz)                          :: b                     ! Exponential shape parameter
REAL(sz)                          :: corio                 ! Coriolis force (1/s)
REAL(sz)                          :: vr                    ! Velocity @ wind radii (knots)
REAL(sz)                          :: phi
REAL(sz), DIMENSION(NPOINTS)      :: phis                  ! Correction factor to B and vh
REAL(sz), DIMENSION(NPOINTS, 4)   :: phis4                 ! Correction factor to B and vh

REAL(sz), DIMENSION(NPOINTS)      :: bs
REAL(sz), DIMENSION(NPOINTS, 4)   :: bs4
REAL(sz), DIMENSION(NPOINTS)      :: vmbl
REAL(sz), DIMENSION(NPOINTS, 4)   :: vmbl4
INTEGER,  DIMENSION(NPOINTS, 4)   :: quadflag4
REAL(sz), DIMENSION(NPOINTS, 4)   :: quadir4
REAL(sz), DIMENSION(NQUADS)       :: vrquadrant
REAL(sz), DIMENSION(NQUADS)       :: radius                ! Wind radii - the distance

INTEGER                           :: quad                  ! Quadrant counter

REAL(sz)                          :: latestrmax            ! most recently calculated value of fitted rmax
REAL(sz)                          :: latestangle           ! angle of the most recently calculated node w.r.t. the storm location
LOGICAL                           :: usequadrantvr
LOGICAL                           :: usevmaxesbl


CONTAINS


!----------------------------------------------------------------
! S U B R O U T I N E   C A L C   I N T E N S I T Y   C H A N G E
!----------------------------------------------------------------
!----------------------------------------------------------------
SUBROUTINE calcintensitychange(var, times, calcInt, status, order)

  USE pahm_global, ONLY : deg2rad
  USE utilities, ONLY : sphericaldistance

  IMPLICIT NONE

  REAL(SZ), DIMENSION(:), INTENT(IN)  :: var, times
  INTEGER, OPTIONAL, INTENT(IN)       :: order

  REAL(SZ), DIMENSION(:), INTENT(OUT) :: calcInt
  INTEGER, INTENT(OUT)                :: status

  INTEGER                             :: ordAcur
  REAL(SZ)                            :: dt1, dt2
  LOGICAL                             :: dt1OK, dt2OK
  REAL(SZ)                            :: val1, val2
  INTEGER                             :: iCnt, maxCnt

  status = 0
  maxcnt = 0

  CALL setmessagesource("CalcIntensityChange")

  IF ((SIZE(shape(var)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
    WRITE(scratchmessage, '(a)') 'The rank of arrays var and times should be equal to 1 (vectors)'
    CALL allmessage(error, scratchmessage)

    CALL unsetmessagesource()

    status = 1

    RETURN
  ELSE
    maxcnt = SIZE(var)
  END IF

  ordacur = 2
  IF (PRESENT(order)) THEN
    IF (order <= 1) ordacur = 1
    IF (order  > 1) ordacur = 2
  END IF
  IF (SIZE(var) < 3) ordacur = 1

  ! Case 1st orded accuracy using backward differences
```

```
    IF (ordacur == 1 )THEN
      DO icnt = 2, maxcnt
        dt1 = times(icnt) - times(icnt - 1)
          dt1ok = (comparereals(dt1, 0.0_sz) /=0)

        val1 = 0.0_sz
        IF (dt1ok) val1 = (var(icnt) - var(icnt - 1)) / dt1

        calcint(icnt) = val1
      END DO
      calcint(1) = calcint(2)

      CALL unsetmessagesource()

      RETURN
    END IF

    ! Case 2nd order accuracy using Forward differences for the first point,
    ! backward differences for the last point and central differences in
    ! between points. Temporal spacing assumed to be uneven (general case).
    ! Forward, backward and central differences are all 2nd order accurate
    ! approximations.

    !----- Forward differences (first point)
    icnt = 1
    dt1 = times(icnt + 1) - times(icnt)
      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
    dt2 = times(icnt + 2) - times(icnt + 1)
      dt2ok = (comparereals(dt2, 0.0_sz) /=0)

    val1 = 0.0_sz
    IF (dt1ok) val1 = (var(icnt + 1) -  var(icnt)) / dt1

    val2 = 0.0_sz
    IF (dt2ok) val2 = (var(icnt + 2) -  var(icnt + 1)) / dt2

    IF (dt1ok .AND. dt2ok) THEN
      calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * val1 - (dt1 / (dt1 + dt2)) * val2
    ELSE IF (.NOT. dt1ok) THEN
      calcint(icnt) = val1
    ELSE
      calcint(icnt) = 2.0_sz * val1 - val2
    END IF
    !----- Forward differences (first point)

    !----- Central differences
    DO icnt = 2, maxcnt - 1
      ! Forward
      dt1 = times(icnt + 1) - times(icnt)
        dt1ok = (comparereals(dt1, 0.0_sz) /=0)
      ! Backward
      dt2 = times(icnt) - times(icnt - 1)
        dt2ok = (comparereals(dt2, 0.0_sz) /=0)

      val1 = 0.0_sz
      IF (dt1ok) val1 = (var(icnt + 1) -  var(icnt)) / dt1

      val2 = 0.0_sz
      IF (dt2ok) val2 = (var(icnt) -  var(icnt - 1)) / dt2

      IF (dt1ok .AND. dt2ok) THEN
        calcint(icnt) = (dt2 / (dt1 + dt2)) * val1 + (dt1 / (dt1 + dt2)) * val2
      ELSE IF (.NOT. dt1ok) THEN
        calcint(icnt) = val1
      ELSE
        calcint(icnt) = val2
      END IF
    END DO
    !----- Central differences

    !----- Backward differences (last point)
    icnt = maxcnt
    dt1 = times(icnt) - times(icnt - 1)
      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
    dt2 = times(icnt - 1) - times(icnt - 2)
      dt2ok = (comparereals(dt2, 0.0_sz) /=0)

    val1 = 0.0_sz
    IF (dt1ok) val1 = (var(icnt) -  var(icnt - 1)) / dt1

    val2 = 0.0_sz
```

```
    IF (dt2ok) val2 = (var(icnt - 1) -  var(icnt - 2)) / dt2

    IF (dt1ok .AND. dt2ok) THEN
      calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * val1 - (dt1 / (dt1 + dt2)) * val2
    ELSE IF (.NOT. dt1ok) THEN
      calcint(icnt) = val1
    ELSE
      calcint(icnt) = 2.0_sz * val1 - val2
    END IF
    !----- Backward differences (last point)

    CALL unsetmessagesource()

  END SUBROUTINE calcintensitychange

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   U V  T R A N S
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE uvtrans(lat, lon, times, u, v, status, order)

    USE pahm_global, ONLY : deg2rad
    USE utilities, ONLY : sphericaldistance

    IMPLICIT NONE

    REAL(SZ), DIMENSION(:), INTENT(IN)  :: lat, lon, times
    INTEGER, OPTIONAL, INTENT(IN)       :: order

    REAL(SZ), DIMENSION(:), INTENT(OUT) :: u, v
    INTEGER, INTENT(OUT)                :: status

    INTEGER                             :: ordAcur
    REAL(SZ)                            :: dx1, dy1, dx2, dy2
    REAL(SZ)                            :: dt1, dt2
    LOGICAL                             :: dt1OK, dt2OK
    REAL(SZ)                            :: u1, u2, v1, v2
    INTEGER                             :: iCnt, maxCnt

    status = 0
    maxcnt = 0

    CALL setmessagesource("UVTrans")

    IF ((SIZE(shape(lat)) /= 1) .OR. (SIZE(shape(lon)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
      WRITE(scratchmessage, '(a)') 'The rank of arrays lat, lon and times should be equal to 1 (vectors)'
      CALL allmessage(error, scratchmessage)

      CALL unsetmessagesource()

      status = 1

      RETURN
    ELSE
      maxcnt = SIZE(lat)
    END IF

    ordacur = 2
    IF (PRESENT(order)) THEN
      IF (order <= 1) ordacur = 1
      IF (order  > 1) ordacur = 2
    END IF
    IF (SIZE(lat) < 3) ordacur = 1

    ! Case 1st orded accuracy using backward differences
    IF (ordacur == 1 )THEN
      DO icnt = 2, maxcnt
        dx1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
        dy1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
        dt1 = abs(times(icnt) - times(icnt - 1))
          dt1ok = (comparereals(dt1, 0.0_sz) /=0)

        u1 = 0.0_sz
        v1 = 0.0_sz
        IF (dt1ok) THEN
          u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
          v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
        END IF

```

```
      u(icnt) = u1
      v(icnt) = v1
    END DO
    u(1) = u(2)
    v(1) = v(2)

    CALL unsetmessagesource()

    RETURN
  END IF

  ! Case 2nd order accuracy using Forward differences for the first point,
  ! backward differences for the last point and central differences in
  ! between points. Temporal spacing assumed to be uneven (general case).
  ! Forward, backward and central differences are all 2nd order accurate
  ! approximations.

  !----- Forward differences (first point)
  icnt = 1
  dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
  dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
  dt1 = abs(times(icnt + 1) - times(icnt))
    dt1ok = (comparereals(dt1, 0.0_sz) /=0)

  dx2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 1), lon(icnt + 2))
  dy2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 2), lon(icnt + 1))
  dt2 = abs(times(icnt + 2) - times(icnt + 1))
    dt2ok = (comparereals(dt2, 0.0_sz) /=0)

  u1 = 0.0_sz
  v1 = 0.0_sz
  IF (dt1ok) THEN
    u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
    v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
  END IF

  u2 = 0.0_sz
  v2 = 0.0_sz
  IF (dt2ok) THEN
    u2 = sign(dx2 / dt2, (lon(icnt + 2) - lon(icnt + 1)))
    v2 = sign(dy2 / dt2, (lat(icnt + 2) - lat(icnt + 1)))
  END IF

  IF (dt1ok .AND. dt2ok) THEN
    u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
    v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
  ELSE IF (.NOT. dt1ok) THEN
    u(icnt) = u1
    v(icnt) = v1
  ELSE
    u(icnt) = 2.0_sz * u1 - u2
    v(icnt) = 2.0_sz * v1 - v2
  END IF
  !----- Forward differences (first point)

  !----- Central differences
  DO icnt = 2, maxcnt - 1
    ! Forward
    dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
    dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
    dt1 = abs(times(icnt + 1) - times(icnt))
      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
    ! Backward
    dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
    dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
    dt2 = abs(times(icnt) - times(icnt - 1))
      dt2ok = (comparereals(dt2, 0.0_sz) /=0)

    u1 = 0.0_sz
    v1 = 0.0_sz
    IF (dt1ok) THEN
      u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
      v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
    END IF

    u2 = 0.0_sz
    v2 = 0.0_sz
    IF (dt2ok) THEN
      u2 = sign(dx2 / dt2, (lon(icnt) - lon(icnt - 1)))
      v2 = sign(dy2 / dt2, (lat(icnt) - lat(icnt - 1)))
    END IF
```

```

      IF (dt1ok .AND. dt2ok) THEN
        u(icnt) = (dt2 / (dt1 + dt2)) * u1 + (dt1 / (dt1 + dt2)) * u2
        v(icnt) = (dt2 / (dt1 + dt2)) * v1 + (dt1 / (dt1 + dt2)) * v2
      ELSE IF (.NOT. dt1ok) THEN
        u(icnt) = u1
        v(icnt) = v1
      ELSE
        u(icnt) = u2
        v(icnt) = v2
      END IF
    END DO
    !----- Central differences

    !----- Backward differences (last point)
    icnt = maxcnt
    dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt - 1))
    dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt - 1), lon(icnt))
    dt1 = abs(times(icnt) - times(icnt - 1))
      dt1ok = (comparereals(dt1, 0.0_sz) /=0)

    dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt - 2))
    dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 2), lon(icnt - 1))
    dt2 = abs(times(icnt - 1) - times(icnt - 2))
      dt2ok = (comparereals(dt2, 0.0_sz) /=0)

    u1 = 0.0_sz
    v1 = 0.0_sz
    IF (dt1ok) THEN
      u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
      v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
    END IF

    u2 = 0.0_sz
    v2 = 0.0_sz
    IF (dt2ok) THEN
      u2 = sign(dx2 / dt2, (lon(icnt - 1) - lon(icnt - 2)))
      v2 = sign(dy2 / dt2, (lat(icnt - 1) - lat(icnt - 2)))
    END IF

    IF (dt1ok .AND. dt2ok) THEN
      u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
      v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
    ELSE IF (.NOT. dt1ok) THEN
      u(icnt) = u1
      v(icnt) = v1
    ELSE
      u(icnt) = 2.0_sz * u1 - u2
      v(icnt) = 2.0_sz * v1 - v2
    END IF
    !----- Backward differences (last point)

    CALL unsetmessagesource()

  END SUBROUTINE uvtrans

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   U V  T R A N S  P O I N T
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE uvtranspoint(lat1, lon1, lat2, lon2, time1, time2, u, v)

    USE pahm_global, ONLY : deg2rad
    USE utilities, ONLY : sphericaldistance

    IMPLICIT NONE

    ! Global variables
    REAL(SZ), INTENT(IN)   :: lat1, lon1, lat2, lon2
    REAL(SZ), INTENT(IN)   :: time1, time2
    REAL(SZ), INTENT(OUT) :: u, v

    ! Local variables
    REAL(SZ) :: dx, dy, dt
    LOGICAL  :: dtOK

    dx = sphericaldistance(lat1, lon1, lat1, lon2)
    dy = sphericaldistance(lat1, lon1, lat2, lon1)
    dt = abs(time2 - time1)
```

```
      dtok = (comparereals(dt, 0.0_sz) /=0)

    u = 0.0_sz
    v = 0.0_sz
    IF (dtok) THEN
      u = sign(dx / dt, (lon2 - lon1))
      v = sign(dy / dt, (lat2 - lat1))
    END IF

  END SUBROUTINE uvtranspoint

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   N E W  V O R T E X
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  SUBROUTINE newvortex(pinf, p0, lat, lon, vm)

    USE pahm_global, ONLY : rhoair, deg2rad, omega, mb2pa, kt2ms

    IMPLICIT NONE

    REAL(SZ), INTENT(IN) :: pinf
    REAL(SZ), INTENT(IN) :: p0
    REAL(SZ), INTENT(IN) :: lat
    REAL(SZ), INTENT(IN) :: lon
    REAL(SZ), INTENT(IN) :: vm

    ! set instance variables
    pn = pinf
    pc = p0
    clat = lat
    clon = lon
    vmax = vm
!PV Check conversions
    ! evaluate basic physical params
    corio = 2.0_sz * omega * sin(deg2rad * clat)
    b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
    b = max(min(b, 2.0_sz), 1.0_sz) ! limit B to range 1.0->2.5
!PV Data already have been converted
    ! added for compatibility of CalcRMaxes to use with simplified nws20
    bs(1:6) = b
    vmbl(1:6) = vmax

  END SUBROUTINE newvortex

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   N E W  V O R T E X  F U L L
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE newvortexfull(pinf, p0, lat, lon, vm)

    USE pahm_global, ONLY : rhoair, deg2rad, kt2ms, omega, mb2pa

    IMPLICIT NONE

    REAL(SZ), INTENT(IN) :: pinf
    REAL(SZ), INTENT(IN) :: p0
    REAL(SZ), INTENT(IN) :: lat
    REAL(SZ), INTENT(IN) :: lon
    REAL(SZ), INTENT(IN) :: vm

    ! set instance variables
    pn   = pinf
    pc   = p0
    clat = lat
    clon = lon
    vmax = vm

    ! evaluate basic physical params
    corio = 2.0_sz * omega * sin(deg2rad * clat)
    b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
    phi       = 1.0_sz
    bs(1:6)   = b
    phis(1:6) = phi
    vmbl(1:6) = vmax

```

```
    ! Jie 2013.01
    ! B = MAX(MIN(B, 2.0_SZ), 1.0_SZ) ! limit B to range 1.0->2.5

  END SUBROUTINE newvortexfull

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T  V O R T E X
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  SUBROUTINE setvortex(pinf, p0, lat, lon)

    USE pahm_global, ONLY : deg2rad, omega

    IMPLICIT NONE

    REAL(SZ), INTENT(IN) :: pinf
    REAL(SZ), INTENT(IN) :: p0
    REAL(SZ), INTENT(IN) :: lat
    REAL(SZ), INTENT(IN) :: lon

    ! set instance variables
    pn   = pinf
    pc   = p0
    clat = lat
    clon = lon

    ! evaluate basic physical params
    corio = 2.0_sz * omega * sin(deg2rad * clat)

  END SUBROUTINE setvortex

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T   R M A X E S
  !----------------------------------------------------------------
  SUBROUTINE setrmaxes(rMaxW)

    IMPLICIT NONE

    REAL(SZ), DIMENSION(4), INTENT(IN) :: rMaxW
    INTEGER :: i

    DO i = 1, 4
       rmaxes(i + 1) = rmaxw(i)
    END DO

  END SUBROUTINE setrmaxes

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   G E T  R M A X E S
  !----------------------------------------------------------------
  SUBROUTINE getrmaxes(rMaxW)

    IMPLICIT NONE

    REAL(SZ), DIMENSION(4), INTENT(OUT) :: rMaxW

    INTEGER :: i

    DO i = 1, 4
      rmaxw(i) = rmaxes(i + 1)
    END DO

  END SUBROUTINE getrmaxes

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   C A L C  R M A X E S
  !----------------------------------------------------------------
  !  Jie 2014.07 Modified with quadrant-varying vmBL, which not only
  !  works for nws19 but for the simplified nws20
  !----------------------------------------------------------------
  SUBROUTINE calcrmaxes()

    IMPLICIT NONE
```

```

    REAL(SZ)              :: root         ! Radius of maximum winds
    REAL(SZ), PARAMETER :: INNERRADIUS = 1.0_sz
    REAL(SZ), PARAMETER :: OUTERRADIUS = 400.0_sz
    REAL(SZ), PARAMETER :: ACCURACY    = 0.0001_sz
    REAL(SZ), PARAMETER :: ZOOM        = 0.01_sz
    INTEGER , PARAMETER :: ITERMAX     = 3
    REAL(SZ)              :: r1, r2, r3, r4, dr
    REAL(SZ)              :: vicinity
    INTEGER               :: n, iter

    !----------------------------
    ! Loop over quadrants of storm
    !----------------------------
    DO n = 1, nquads
      ! set B and vMax values for each quadrant
      ! for nws19, B and vMax are constant
      ! for simplified nws20, B is constant, while vMax is not
      b = bs(n + 1)
      vmax = vmbl(n + 1)

      quad = n
      root = -1.0_sz
      r1   = innerradius
      r2   = outerradius
      dr   = 1.0_sz
      DO iter = 1, itermax
        root = findroot(vhwithcori, r1, r2, dr, r3, r4)
        r1   = r3
        r2   = r4
        dr   = dr * zoom
      END DO

      ! determine if rMax is actually in the vicinity of the
      ! isotach radius that we are using to solve for rMax,
      ! and if so, take another shot at finding the
      ! rMax using the gradient wind balance that neglects
      ! coriolis (and is appropriate in the vicinity of rMax)
      vicinity = abs(root - radius(quad)) / root
      IF ((root < 0.0_sz) .OR. (vicinity <= 0.0_sz)) THEN
        r1 = innerradius
        r2 = outerradius
        dr = 1.0_sz
        DO iter = 1, itermax
          root = findroot(vhnocori, r1, r2, dr, r3, r4)
          r1 = r3
          r2 = r4
          dr = dr * zoom
        END DO
      END IF

      rmaxes(n + 1) = root
    END DO

  END SUBROUTINE calcrmaxes

!==============================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   C A L C  R M A X E S  F U L L
  !----------------------------------------------------------------
  ! Jie 2013.02 added looping procedures to calculate bs and phis
  !----------------------------------------------------------------
  SUBROUTINE calcrmaxesfull()

    USE pahm_global, ONLY : rhoair, nm2m, kt2ms, mb2pa

    IMPLICIT NONE

    REAL(SZ)              :: root            ! Radius of maximum winds
    REAL(SZ), PARAMETER :: INNERRADIUS = 1.0_sz
    REAL(SZ), PARAMETER :: OUTERRADIUS = 500.0_sz
    REAL(SZ), PARAMETER :: ACCURACY    = 0.0001_sz
    REAL(SZ), PARAMETER :: ZOOM        = 0.01_sz
    INTEGER , PARAMETER :: ITERMAX     = 3
    REAL(SZ)              :: r1, r2, r3, r4, dr
    INTEGER               :: n, iter, noRootFlag
    REAL(SZ)              :: bNew, bNew1
    REAL(SZ)              :: phiNew
    INTEGER, PARAMETER   :: cont = 400      ! Max # of iterations
    INTEGER               :: iCont, ibCont  ! iteration counter
```

```

211 FORMAT(a7, x ,i2, x, a38)

!-----------------------------
! Loop over quadrants of storm
!-----------------------------
DO n = 1, nquads
  norootflag = 0

  ! initialize B and phi values for each quadrant
  b    = bs(n + 1)
  phi  = phis(n + 1)
  vmax = vmbl(n + 1)

  ! Loop the root-solving process to converge B, for in the
  ! new wind formulation, B is a function of rMax, vMax, f, and phi
  DO icont = 1, cont ! logical expre. is at the end to exit the loop
    norootflag = 0
    quad = n
    root = -1.0_sz
    r1 = innerradius
    r2 = outerradius
    dr = 1.0_sz
    DO iter = 1, itermax
      root = findroot(vhwithcorifull, r1, r2, dr, r3, r4)
      r1 = r3
      r2 = r4
      dr = dr * zoom
    END DO

    ! avoid invalid B value when root is not found
    IF (root < 0.0_sz) THEN
    !  r1 = INNERRADIUS
    !  r2 = OUTERRADIUS
    !  dr = 1.0_SZ
    !  DO iter = 1, ITERMAX
    !    root = FindRoot(VhNoCori, r1, r2, dr, r3, r4)
    !    r1 = r3
    !    r2 = r4
    !    dr = dr * ZOOM
    !  END DO
      root = 1.0 * radius(quad)
      norootflag = 1
    END IF

    rmaxes(n + 1) = root

    ! Jie 2013.02
    ! determine if B converges, if yes, break loop and assign
    ! values to rMaxes, if not, continue the loop to re-calculate
    ! root and re-evaluate bs
    phinew = 1 + vmax * kt2ms * root * nm2m * corio /                          &
             (b * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio))
    bnew   = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) *       &
              rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa)
    DO ibcont = 1, cont
      bnew1 = bnew
      phinew = 1 + vmax * kt2ms * root * nm2m * corio /                        &
               (bnew * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio))
      bnew   = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) *     &
                rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa)

      IF (abs(bnew - bnew1) <= 0.01_sz) EXIT
    END DO

    ! debug with aswip
    !IF (ibCont >= cont) THEN
    !  WRITE(1111, 211) "iquad=", n, "bNew did not fully converge, procede"
    !END IF
    ! end debug with aswip

    IF (abs(b - bnew) <= 0.01_sz) EXIT

    ! update B and phi for next iteration
    ! warning: modifications made here also affect other subroutines
    b   = bnew
    phi = phinew
  END DO !iCont = 1, cont

  ! update to the latest values for aswip output
  bs(n + 1)   = bnew
```

```
      phis(n +1 ) = phinew

      ! debug with aswip
      !IF (iCont >= cont) THEN
      !  WRITE(1111, 211) "iquad=", n, "B did not fully converge, procede"
      !END IF
      ! end debug with aswip

      ! determine if rMax is actually in the vicinity of the
      ! isotach radius that we are using to solve for rMax,
      ! and if so, take another shot at finding the
      ! rMax using the gradient wind equation that neglects
      ! coriolis (and is appropriate in the vicinity of rMax)
      ! Jie 2013.01
      !vicinity = ABS(root - radius(quad)) / root
      IF (norootflag == 1) THEN
         WRITE(*, *) "iquad=", n, "No root found, return dist. to Isotach"
      END IF
    END DO  !n = 1, nQuads

  END SUBROUTINE calcrmaxesfull

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   F I T   R M A X E S
  !----------------------------------------------------------------
  !  RJW 07 - 2009
  !----------------------------------------------------------------
  SUBROUTINE fitrmaxes()

    IMPLICIT NONE

    ! Generate 2 additional (theta, rMax) points for curve-fit
    rmaxes(1) = rmaxes(5)
    rmaxes(6) = rmaxes(2)

  END SUBROUTINE fitrmaxes

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   F I T   R M A X E S  4
  !----------------------------------------------------------------
  SUBROUTINE fitrmaxes4()

    IMPLICIT NONE

    ! Generate 2 additional points for curve-fit
    quadflag4(1, 1:4) = quadflag4(5, 1:4)
    quadflag4(6, 1:4) = quadflag4(2, 1:4)

    quadir4(1, 1:4) = quadir4(5, 1:4)
    quadir4(6, 1:4) = quadir4(2, 1:4)

    rmaxes4(1, 1:4) = rmaxes4(5, 1:4)
    rmaxes4(6, 1:4) = rmaxes4(2, 1:4)

    bs4(1, 1:4) = bs4(5, 1:4)
    bs4(6, 1:4) = bs4(2, 1:4)

    phis4(1, 1:4) = phis4(5, 1:4)
    phis4(6, 1:4) = phis4(2, 1:4)

    vmbl4(1, 1:4) = vmbl4(5, 1:4)
    vmbl4(6, 1:4) = vmbl4(2, 1:4)

  END SUBROUTINE fitrmaxes4

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T   V M A X E S   B L
  !----------------------------------------------------------------
  SUBROUTINE setvmaxesbl(vMaxW)

    IMPLICIT NONE

    REAL(SZ), DIMENSION(4), INTENT(IN) :: vMaxW

    INTEGER :: i
```

```

  DO i = 1, 4
    vmbl(i + 1) = vmaxw(i)
  END DO

END SUBROUTINE setvmaxesbl

!==============================================================================

  !---------------------------------------------------------------
  ! S U B R O U T I N E   G E T  V M A X E S  B L
  !---------------------------------------------------------------
SUBROUTINE getvmaxesbl(vMaxW)

  IMPLICIT NONE

  REAL(SZ), DIMENSION(4), INTENT(OUT) :: vMaxW

  INTEGER :: i

  DO i = 1, 4
    vmaxw(i) = vmbl(i + 1)
  END DO

END SUBROUTINE getvmaxesbl

!==============================================================================

  !---------------------------------------------------------------
  ! S U B R O U T I N E   S E T  U S E  V M A X E S  B L
  !---------------------------------------------------------------
SUBROUTINE setusevmaxesbl(u)

  IMPLICIT NONE

  LOGICAL, INTENT(IN) :: u

  usevmaxesbl = u

END SUBROUTINE setusevmaxesbl

  !---------------------------------------------------------------
  ! S U B R O U T I N E   S E T  S H A P E  P A R A M E T E R
  !---------------------------------------------------------------
SUBROUTINE setshapeparameter(param)

  IMPLICIT NONE

  REAL(SZ) :: param

  b = param

END SUBROUTINE setshapeparameter

!==============================================================================

  !---------------------------------------------------------------
  ! F U N C T I O N   G E T  S H A P E  P A R A M E T E R
  !---------------------------------------------------------------
REAL(sz) function getshapeparameter() result(myvalout)

  IMPLICIT NONE

  myvalout = b

  RETURN

END FUNCTION getshapeparameter

!==============================================================================

  !---------------------------------------------------------------
  ! F U N C T I O N   G E T  S H A P E  P A R A M E T E R S
  !---------------------------------------------------------------
FUNCTION getshapeparameters() RESULT(myValOut)

  IMPLICIT NONE

  REAL(sz), DIMENSION(4) :: myvalout

  INTEGER :: i
```

```

    DO i = 1, 4
      myvalout(i) = bs(i + 1)
    END DO

    RETURN

  END FUNCTION getshapeparameters

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G E T  P H I  F A C T O R S
  !----------------------------------------------------------------
  FUNCTION getphifactors() RESULT(myValOut)

    IMPLICIT NONE

    REAL(sz), DIMENSION(4) :: myvalout

    INTEGER :: i

    DO i = 1, 4
      myvalout(i) = phis(i + 1)
    END DO

    RETURN

  END FUNCTION getphifactors

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T  I S O T A C H  R A D I I
  !----------------------------------------------------------------
  SUBROUTINE setisotachradii(ir)

    IMPLICIT NONE

    REAL(SZ), DIMENSION(4), INTENT(IN) :: ir

    radius(:) = ir(:)

  END SUBROUTINE setisotachradii

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T  I S O T A C H  W I N D  S P E E D S
  !----------------------------------------------------------------
  SUBROUTINE setisotachwindspeeds(vrQ)

    IMPLICIT NONE

    REAL(SZ), DIMENSION(4), INTENT(IN) :: vrQ

    vrquadrant(:) = vrq(:)

  END SUBROUTINE setisotachwindspeeds

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   S E T  U S E  Q U A D R A N T  V R
  !----------------------------------------------------------------
  SUBROUTINE setusequadrantvr(u)

    IMPLICIT NONE

    LOGICAL, INTENT(IN) :: u

    usequadrantvr = u

  END SUBROUTINE setusequadrantvr

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G E T  L A T E S T  A N G L E
  !----------------------------------------------------------------
  LOGICAL FUNCTION getusequadrantvr() RESULT(myValOut)
```

```
    IMPLICIT NONE

    myvalout = usequadrantvr

  END FUNCTION getusequadrantvr

!==============================================================================

  !---------------------------------------------------------------
  ! F U N C T I O N   S P I N T E R P
  !---------------------------------------------------------------
  !
  !---------------------------------------------------------------
  REAL(sz) function spinterp(angle, dist, opt) result(myvalout)

    IMPLICIT NONE

    REAL(sz), INTENT(IN)             :: angle, dist
    INTEGER, INTENT(IN)              :: opt
    REAL(sz), DIMENSION(NPOINTS, 4)  :: param
    REAL(sz)                         :: temp1, temp2
    REAL(sz)                         :: deltaangle
    INTEGER                          :: iquad

    IF (opt == 1) THEN
      param = rmaxes4
    ELSE IF (opt == 2) THEN
      param = bs4
    ELSE IF (opt == 3) THEN
      param = vmbl4
    END IF

    IF (angle <= 45.0_sz) THEN
      iquad = 5
      deltaangle = 45.0_sz + angle
    ELSE IF (angle <= 135.0_sz) THEN
      iquad = 2
      deltaangle = angle - 45.0_sz
    ELSE IF (angle <= 225.0_sz) THEN
      iquad = 3
      deltaangle = angle - 135.0_sz
    ELSE IF (angle <= 315.0_sz) THEN
      iquad = 4
      deltaangle = angle - 225.0_sz
    ELSE IF (angle > 315.0_sz) THEN
      iquad = 5
      deltaangle = angle - 315.0_sz
    END IF

    ! nearest neighbor weighted interpolation
    IF ( deltaangle < 1.0_sz ) THEN
      myvalout = interpr(param, iquad, dist)
    ELSE IF (deltaangle > 89.0_sz) THEN
      myvalout = interpr(param, iquad + 1, dist)
    ELSE
      temp1 = interpr(param, iquad, dist)
      temp2 = interpr(param, iquad + 1, dist)
      myvalout = (temp1 / deltaangle**2 + temp2 / (90.0 - deltaangle)**2) /    &
              (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
    END IF

  END FUNCTION spinterp

!==============================================================================

  !---------------------------------------------------------------
  ! F U N C T I O N   I N T E R P R
  !---------------------------------------------------------------
  REAL(sz) function interpr(quadval, quadsel, quaddis) result(myvalout)

    IMPLICIT NONE

    REAL(sz), DIMENSION(NPOINTS, 4), INTENT(IN)  :: quadval
    INTEGER, INTENT(IN)                          :: quadsel
    REAL(sz), INTENT(IN)                         :: quaddis

    REAL(sz)                                     :: fac
    INTEGER                                      :: totalisot

    totalisot = sum(quadflag4(quadsel, :))
```

```
    SELECT CASE(totalisot)
      CASE(1)
        myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))
      CASE(2)
        IF (quaddis > quadir4(quadsel, 1)) THEN
          myvalout = quadval(quadsel, 1)
        ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
          fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
          myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
        ELSE
           myvalout = quadval(quadsel, 2)
        END IF
      CASE(3)
        IF (quaddis > quadir4(quadsel, 1)) THEN
          myvalout = quadval(quadsel, 1)
        ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
          fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
          myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
        ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
          fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
          myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
        ELSE
          myvalout = quadval(quadsel, 3)
        END IF
      CASE(4)
        IF (quaddis > quadir4(quadsel, 1)) THEN
          myvalout = quadval(quadsel, 1)
        ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
          fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
          myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
        ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
          fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
          myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
        ELSE IF (quaddis > quadir4(quadsel, 4)) THEN
          fac = (quaddis - quadir4(quadsel, 4)) / (quadir4(quadsel, 3) - quadir4(quadsel, 4))
          myvalout = quadval(quadsel, 3) * fac + quadval(quadsel, 4) * (1 - fac)
        ELSE
          myvalout = quadval(quadsel, 4)
        END IF
      CASE default
        ! For whatever reason if our algorithm fails, add the following
        ! line to avoid run-time errors
        myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))
        WRITE(*, *) "ERROR: InterpR failed in nws20get." !PV remove it of modify it?
    END SELECT

  END FUNCTION interpr

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   R M W
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  REAL(sz) function rmw(angle) result(myvalout)

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: angle
    INTEGER              :: basequadrant
    REAL(sz)             :: deltaangle

    IF (angle <= 45.0_sz) THEN
      basequadrant = 5
      deltaangle = 45.0_sz + angle
    ELSE IF (angle <= 135.0_sz) THEN
      basequadrant = 2
      deltaangle = angle - 45.0_sz
    ELSE IF (angle <= 225.0_sz) THEN
      basequadrant = 3
      deltaangle = angle - 135.0_sz
    ELSE IF (angle <= 315.0_sz) THEN
      basequadrant = 4
      deltaangle = angle - 225.0_sz
    ELSE IF (angle > 315.0_sz) THEN
      basequadrant = 5
      deltaangle = angle - 315.0_sz
    END IF

    ! nearest neighbor weighted interpolation
```

```
    IF ( deltaangle < 1.0_sz ) THEN
      myvalout = rmaxes(basequadrant)    ! avoid div by zero
    ELSE IF ( deltaangle > 89.0_sz ) THEN
      myvalout = rmaxes(basequadrant + 1) ! avoid div by zero
    ELSE
      myvalout = (rmaxes(basequadrant) / deltaangle**2 +                         &
                  rmaxes(basequadrant + 1) / (90.0 - deltaangle)**2) /           &
                 (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
    END IF

    ! linearly interpolate
    !myValOut = (deltaAngle / 90.0_SZ) *                               &
    !           (rMaxes(baseQuadrant + 1) - rMaxes(baseQuadrant)) +    &
    !           rMaxes(baseQuadrant)

  END FUNCTION rmw

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   U V P
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  SUBROUTINE uvp(lat, lon, uTrans, vTrans, u, v, p)

    USE pahm_global, ONLY : windreduction, one2ten, deg2rad, rad2deg, mb2pa, kt2ms, nm2m, m2nm, rearth

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)  :: lat
    REAL(SZ), INTENT(IN)  :: lon
    REAL(SZ), INTENT(IN)  :: uTrans
    REAL(SZ), INTENT(IN)  :: vTrans

    REAL(SZ), INTENT(OUT) :: u
    REAL(SZ), INTENT(OUT) :: v
    REAL(SZ), INTENT(OUT) :: p

    REAL(SZ)              :: transSpdX  !NWS8-style translation speed
    REAL(SZ)              :: transSpdY  !NWS8-style translation speed

    REAL(SZ)              :: dx
    REAL(SZ)              :: dy
    REAL(SZ)              :: dist
    REAL(SZ)              :: rmx
    REAL(SZ)              :: angle
    REAL(SZ)              :: speed
    REAL(SZ)              :: uf
    REAL(SZ)              :: vf
    REAL(SZ)              :: percentCoriolis
    REAL(SZ)              :: speedAtRMax
    REAL(SZ)              :: vMaxFactor

    !------------------------------------------------------
    ! Calculate distance and angle between eye of hurricane
    ! and input nodal point
    !------------------------------------------------------
    dx = deg2rad * rearth * (lon - clon) * cos(deg2rad * clat)
    dy = deg2rad * rearth * (lat - clat)
    dist = sqrt(dx * dx + dy * dy)

    !--------------------------------------
    ! Handle special case at eye of hurricane
    ! in eye velocity is zero not translational velocity
    !--------------------------------------
    IF (dist < 1.0_sz) THEN
      u = 0.0_sz
      v = 0.0_sz
      p = pc * mb2pa

      RETURN
    END IF

    dist = m2nm * dist

    angle = 360.0_sz + rad2deg * atan2(dx, dy)
    IF (angle > 360.0_sz) angle = angle - 360.0_sz

    latestangle = angle
    rmx = rmw(angle)
```

```
    latestrmax = rmx

    !--------------------------------------------------
    ! Compute (u,v) wind velocity components from the
    ! asymmetric hurricane vortex.
    !
    ! Note: the vortex winds are valid at the top of the
    ! surface layer, so reduce the winds to the surface.
    ! Also convert the winds from max sustained 1-minute
    ! averages to 10-minute averages for the storm surge
    ! model.
    !--------------------------------------------------
    percentcoriolis = 1.0_sz
    speed = sqrt((vmax * kt2ms)**2 * (rmx / dist)**b * exp(1.0_sz - (rmx / dist)**b) +   &
                 (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2)                    &
               - nm2m * dist * percentcoriolis * corio / 2.0_sz

    ! calculate the wind speed (m/s) at rMax, using
    ! equation that includes full coriolis
    speedatrmax = sqrt((vmax * kt2ms)**2 * exp(0.0_sz) +                        &
                       (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2)    &
                     - nm2m * dist * percentcoriolis * corio / 2.0_sz

    ! calculate a factor to place the velocity profile so that
    ! it hits vMax
    vmaxfactor = vmax * kt2ms / speedatrmax

    ! jgf20111007: Calculate NWS8-like translation speed
    transspdx = (abs(speed / speedatrmax)) * utrans * kt2ms
    transspdy = (abs(speed / speedatrmax)) * vtrans * kt2ms

    speed = speed * vmaxfactor

    ! now reduce the wind speed to the surface
    speed = speed * windreduction

    u = -speed * cos(deg2rad * angle)
    v =  speed * sin(deg2rad * angle)

    ! Alter wind direction by adding a frictional inflow angle
    CALL rotate(u, v, fang(dist, rmx), clat, uf, vf)
    u = uf
    v = vf
    !
    ! jgf20111007: Add in the translation velocity
    u = u + transspdx
    v = v + transspdy
    !
    ! convert from 1 minute averaged winds to 10 minute averaged
    ! winds for use in ADCIRC
    u = u * one2ten
    v = v * one2ten

    ! Compute surface pressure from asymmetric hurricane vortex
    p = mb2pa * (pc + (pn - pc) * exp(-(rmx / dist)**b))

    ! cut off the vortex field after 401nm
    ! TODO: 401nm should be replaced with something less
    ! arbitrary ... and find a better way to blend this
    !IF ( dist > 401.0_SZ ) THEN
    !  u = 0.0_SZ
    !  v = 0.0_SZ
    !  p = MB2PA * pn
    !END IF

  END SUBROUTINE uvp

!================================================================================


  !----------------------------------------------------------------
  ! S U B R O U T I N E   U V P R
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  SUBROUTINE uvpr(iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, &
                  uTrans, vTrans, geof, u, v, p)

    USE pahm_global, ONLY : windreduction, one2ten, deg2rad, mb2pa, kt2ms, nm2m

    IMPLICIT NONE
```

```
      REAL(SZ), INTENT(IN)  :: iDist
      REAL(SZ), INTENT(IN)  :: iAngle
      REAL(SZ), INTENT(IN)  :: iRmx
      REAL(SZ), INTENT(IN)  :: iRmxTrue
      REAL(SZ), INTENT(IN)  :: iB
      REAL(SZ), INTENT(IN)  :: iVm
      REAL(SZ), INTENT(IN)  :: iPhi
      REAL(SZ), INTENT(IN)  :: uTrans
      REAL(SZ), INTENT(IN)  :: vTrans
      INTEGER , INTENT(IN)  :: geof

      REAL(SZ), INTENT(OUT) :: u
      REAL(SZ), INTENT(OUT) :: v
      REAL(SZ), INTENT(OUT) :: p

      REAL(SZ)              :: transSpdX  !NWS8-style translation speed
      REAL(SZ)              :: transSpdY  !NWS8-style translation speed
      REAL(SZ)              :: rmx
      REAL(SZ)              :: speed
      REAL(SZ)              :: uf
      REAL(SZ)              :: vf
      REAL(SZ)              :: percentCoriolis

      rmx  = irmx
      b    = ib
      vmax = ivm
      phi  = iphi

      !----------------------------------------
      ! Handle special case at eye of hurricane
      ! in eye velocity is zero not translational velocity
      !----------------------------------------
      IF (idist < 1.0_sz) THEN
        u = 0.0_sz
        v = 0.0_sz
        p = pc * mb2pa

        RETURN
      END IF

      !--------------------------------------------------
      ! Compute (u, v) wind velocity components from the
      ! asymmetric hurricane vortex.
      !
      ! Note: the vortex winds are valid at the top of the
      ! surface layer, so reduce the winds to the surface.
      ! Also convert the winds from max sustained 1-minute
      ! averages to 10-minute averages for the storm surge
      ! model.
      !--------------------------------------------------
      percentcoriolis = 1.0_sz
      ! Jie 2014.07
      IF (geof == 1) THEN
        speed = sqrt(((vmax * kt2ms)**2 + vmax * kt2ms * rmx * nm2m * percentcoriolis * corio) *   &
                      (rmx / idist)**b * exp(phi * (1.0_sz - (rmx / idist)**b)) +                  &
                      (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) -                      &
                     nm2m * idist * percentcoriolis * corio / 2.0_sz
      ELSE
        speed = sqrt((vmax * kt2ms)**2 * (rmx / idist)**b * exp(1.0_sz - (rmx / idist)**b) +   &
                      (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) -   &
                     nm2m * idist * percentcoriolis * corio / 2.0_sz
      ENDIF

      ! jgf20111007: Calculate NWS8-like translation speed
      transspdx = (abs(speed / (vmax * kt2ms))) * utrans * kt2ms
      transspdy = (abs(speed / (vmax * kt2ms))) * vtrans * kt2ms

      ! now reduce the wind speed to the surface
      speed = speed * windreduction

      u = -speed * cos(deg2rad * iangle)
      v =  speed * sin(deg2rad * iangle)
      !
      ! Alter wind direction by adding a frictional inflow angle
      CALL rotate(u, v, fang(idist, irmxtrue), clat, uf, vf)
      u = uf
      v = vf
      !
      ! jgf20111007: Add in the translation velocity
      u = u + transspdx
```

```
    v = v + transspdy

    ! convert from 1 minute averaged winds to 10 minute averaged
    ! winds for use in ADCIRC
    u = u * one2ten
    v = v * one2ten

    ! Compute surface pressure from asymmetric hurricane vortex
    IF (geof == 1) THEN
      p = mb2pa * (pc + (pn - pc) * exp( - phi * (rmx / idist)**b))
    ELSE
      p = mb2pa * (pc + (pn - pc) * exp(-(rmx / idist)**b))
    ENDIF

    ! cut off the vortex field after 401nm
    ! TODO: 401nm should be replaced with something less
    ! arbitrary ... and find a better way to blend this
    !if ( dist > 401.0_SZ ) then
    !u = 0.0_SZ
    !v = 0.0_SZ
    !p = MB2PA * pn
    !endif

  END SUBROUTINE uvpr

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   F A N G
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  REAL(sz) function fang(r, rmx) result(myvalout)

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: r
    REAL(sz), INTENT(IN) :: rmx

    IF ((0.0_sz <= r) .AND. (r < rmx)) THEN
      myvalout = 10.0_sz * r / rmx
    ELSE IF ((rmx <= r) .AND. (r < 1.0_sz * rmx)) THEN
      myvalout = 10.0_sz + 75.0_sz * (r / rmx - 1.0_sz)
    ELSE IF (r >= 1.0_sz * rmx) THEN
      myvalout = 25.0_sz
    ELSE
      myvalout = 0.0_sz
    END IF

  END FUNCTION fang

!================================================================================

  !----------------------------------------------------------------
  ! S U B R O U T I N E   R O T A T E
  !----------------------------------------------------------------
  !
  !----------------------------------------------------------------
  SUBROUTINE rotate(x, y, angle, whichWay, xr, yr)

    USE pahm_global, ONLY : deg2rad

    IMPLICIT NONE

    REAL(SZ), INTENT(IN)  :: x
    REAL(SZ), INTENT(IN)  :: y
    REAL(SZ), INTENT(IN)  :: angle
    REAL(SZ), INTENT(IN)  :: whichWay

    REAL(SZ), INTENT(OUT) :: xr
    REAL(SZ), INTENT(OUT) :: yr

    REAL(SZ)              :: A, cosA, sinA

    a = sign(1.0_sz, whichway) * deg2rad * angle
    cosa = cos(a)
    sina = sin(a)

    xr = x * cosa - y * sina
    yr = x * sina + y * cosa

```

```
  END SUBROUTINE rotate

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G E T  L A T E S T  R M A X
  !----------------------------------------------------------------
  REAL(sz) function getlatestrmax() result(myvalout)

    IMPLICIT NONE

    myvalout = latestrmax

  END FUNCTION getlatestrmax

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   G E T  L A T E S T  A N G L E
  !----------------------------------------------------------------
  REAL(sz) function getlatestangle() result(myvalout)

    IMPLICIT NONE

    myvalout = latestangle

  END FUNCTION getlatestangle

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   V H  W I T H  C O R I  F U L L
  !----------------------------------------------------------------
  ! Jie 2013.02 Modified to use the full gradient wind eq.
  !----------------------------------------------------------------
  REAL(sz) function vhwithcorifull(testrmax) result(myvalout)

    USE pahm_global, ONLY : nm2m, kt2ms, ms2kt

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: testrmax

    REAL(sz)             :: thisvr ! the radial wind speed we've been given
    REAL(sz)             :: vh

    !-------------------------
    ! func(x = rMax) = vh - vr
    !-------------------------
    IF (getusequadrantvr() .EQV. .true.) THEN
      thisvr = vrquadrant(quad)
    ELSE
      thisvr = vr
    END IF

    ! Jie 2013.02
    vh = ms2kt * (sqrt(((vmax * kt2ms)**2 + vmax * kt2ms * testrmax * nm2m * corio) *     &
                       (testrmax / radius(quad))**b *                                     &
                       exp(phi * (1.0_sz - (testrmax / radius(quad))**b)) +               &
                       (nm2m * radius(quad) * corio / 2.0_sz)**2) -                       &
                  nm2m * radius(quad) * corio / 2.0_sz)

    myvalout = vh - thisvr

    RETURN

  END FUNCTION vhwithcorifull

!================================================================================

  !----------------------------------------------------------------
  ! F U N C T I O N   V H  W I T H  C O R I
  !----------------------------------------------------------------
  !----------------------------------------------------------------
  REAL(sz) function vhwithcori(testrmax) result(myvalout)

    USE pahm_global, ONLY : nm2m, kt2ms, ms2kt

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: testrmax
```

```

    REAL(sz)                 :: thisvr ! the radial wind speed we've been given
    REAL(sz)                 :: vh

    !-------------------------
    ! func(x = rMax) = vh - vr
    !-------------------------
    IF (getusequadrantvr() .EQV. .true.) THEN
      thisvr = vrquadrant(quad)
    ELSE
      thisvr = vr
    END IF

    vh = ms2kt * (sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b *                 &
                                              exp(1.0_sz - (testrmax / radius(quad))**b) +      &
                       (nm2m * radius(quad) * corio / 2.0_sz)**2) -                       &
                  nm2m * radius(quad) * corio / 2.0_sz)

    myvalout = vh - thisvr

    RETURN

  END FUNCTION vhwithcori

!================================================================================

  !--------------------------------------------------------------
  ! F U N C T I O N   V H  N O  C O R I
  !--------------------------------------------------------------
  REAL(sz) function vhnocori(testrmax) result(myvalout)

    USE pahm_global, ONLY : kt2ms, ms2kt

    IMPLICIT NONE

    REAL(sz), INTENT(IN) :: testrmax

    REAL(sz) :: thisvr ! the radial wind speed we've been given

    IF (getusequadrantvr() .EQV. .true.) THEN
      thisvr = vrquadrant(quad)
    ELSE
      thisvr = vr
    END IF

    myvalout = abs(ms2kt * sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b *      &
                                exp(1 - (testrmax / radius(quad))**b))) - thisvr

    RETURN

  END FUNCTION vhnocori

!================================================================================

  !--------------------------------------------------------------
  ! F U N C T I O N   F I N D  R O O T
  !--------------------------------------------------------------
  !--------------------------------------------------------------
  REAL(sz) function findroot(func, x1, x2, dx, a, b) result(myroot)
!PV Need to check for the x2 variable is not used anywhere next
    IMPLICIT NONE

    REAL(sz), EXTERNAL      :: func
    REAL(sz), INTENT(IN)    :: x1, x2           ! Search interval [x1,x2]
    REAL(sz), INTENT(IN)    :: dx               ! Marching increment
    REAL(sz), INTENT(OUT)   :: a, b             ! x values that bracket root

    INTEGER , PARAMETER     :: itermax = 400    ! Max # of iterations
    INTEGER                 :: iter             ! iteration counter
    REAL(sz)                :: fa, fb           ! function values f(x)

    ! Initialize left side of interval
    a  = x1
    fa = func(a)

    ! March along interval until root is found
    ! or solution diverges.
    myroot = a
    DO iter = 1, itermax
      b  = x1 + iter * dx
      fb = func(b)
```

```
      ! Check progress
      IF ((fa * fb < 0.0_sz) .OR. (abs(fb) > abs(fa))) THEN
        ! Assign root
        IF (abs(fb) > abs(fa)) THEN
          myroot = a
        ELSE
          myroot = b
        END IF

        EXIT
      END IF

      ! Move right search interval values to left side
      ! for next iteration.
      a  = b
      fa = fb
    END DO

    IF (iter >= itermax) THEN
      print *, "FUNCTION FindRoot: exceeded max # of iterations"
      myroot = -99999.0
    END IF

    RETURN

  END FUNCTION findroot

!================================================================================

END MODULE pahm_vortex
```

Index