

PaHM

Generated by Doxygen 1.9.2

1 PaHM	1
1.1 Table of Contents	1
1.1.1 Modeling System Description	1
1.1.2 Installation guide	2
1.1.3 User guide	2
1.1.4 Best practices	2
1.1.5 Developer's corner	2
1.1.6 PaHM code	2
2 PaHM code	2
2.1 Class list	3
2.2 File list	3
2.3 Todo list	3
2.4 Deprecated list	3
3 Modeling System Description	3
3.1 Introduction	3
3.2 Purposes	3
3.3 Technical features list	3
3.3.1 Modeling features	3
3.3.2 Numerical features	3
3.3.3 System and validation features	3
3.3.4 Third-party libraries	3
3.3.5 Compilation tools	3
4 Modules Index	3
4.1 Modules List	3
5 Data Type Index	4
5.1 Data Types List	4
6 File Index	5
6.1 File List	5
7 Module Documentation	6
7.1 csv_module Module Reference	6
7.1.1 Function/Subroutine Documentation	7
7.1.2 Variable Documentation	8
7.2 csv_parameters Module Reference	10
7.2.1 Variable Documentation	10
7.3 csv_utilities Module Reference	12

7.3.1 Function/Subroutine Documentation	12
7.4 pahm_drivermod Module Reference	14
7.4.1 Function/Subroutine Documentation	15
7.4.2 Variable Documentation	18
7.5 pahm_global Module Reference	19
7.5.1 Function/Subroutine Documentation	21
7.5.2 Variable Documentation	21
7.6 pahm_mesh Module Reference	38
7.6.1 Function/Subroutine Documentation	39
7.6.2 Variable Documentation	41
7.7 pahm_messages Module Reference	44
7.7.1 Function/Subroutine Documentation	45
7.7.2 Variable Documentation	51
7.8 pahm_netcdfio Module Reference	54
7.8.1 Detailed Description	54
7.8.2 Function/Subroutine Documentation	54
7.8.3 Variable Documentation	60
7.9 pahm_sizes Module Reference	60
7.9.1 Function/Subroutine Documentation	61
7.9.2 Variable Documentation	64
7.10 pahm_vortex Module Reference	66
7.10.1 Function/Subroutine Documentation	68
7.10.2 Variable Documentation	84
7.11 parwind Module Reference	89
7.11.1 Detailed Description	90
7.11.2 Function/Subroutine Documentation	90
7.11.3 Variable Documentation	99
7.12 sortutils Module Reference	99
7.12.1 Function/Subroutine Documentation	101
7.13 timedateutils Module Reference	114
7.13.1 Function/Subroutine Documentation	115
7.13.2 Variable Documentation	130
7.14 utilities Module Reference	133
7.14.1 Function/Subroutine Documentation	134
7.14.2 Variable Documentation	158
8 Data Type Documentation	159
8.1 pahm_messages::allmessage Interface Reference	159
8.1.1 Detailed Description	159

8.1.2 Member Function/Subroutine Documentation	159
8.2 sortutils::arraycopy Interface Reference	160
8.2.1 Detailed Description	160
8.2.2 Member Function/Subroutine Documentation	160
8.3 sortutils::arrayequal Interface Reference	163
8.3.1 Detailed Description	164
8.3.2 Member Function/Subroutine Documentation	164
8.4 sortutils::arth Interface Reference	165
8.4.1 Detailed Description	166
8.4.2 Member Function/Subroutine Documentation	166
8.5 parwind::besttrackdata_t Type Reference	168
8.5.1 Detailed Description	169
8.5.2 Member Data Documentation	169
8.6 pahm_sizes::comparereals Interface Reference	175
8.6.1 Detailed Description	175
8.6.2 Member Function/Subroutine Documentation	175
8.7 utilities::cpptogeo Interface Reference	177
8.7.1 Detailed Description	177
8.7.2 Member Function/Subroutine Documentation	177
8.8 csv_module::csv_file Type Reference	179
8.8.1 Detailed Description	180
8.8.2 Member Function/Subroutine Documentation	180
8.8.3 Member Data Documentation	184
8.9 csv_module::csv_string Type Reference	186
8.9.1 Detailed Description	186
8.9.2 Member Data Documentation	186
8.10 pahm_netcdfio::filedata_t Type Reference	187
8.10.1 Detailed Description	187
8.10.2 Member Data Documentation	187
8.11 pahm_sizes::fixnearwholereal Interface Reference	188
8.11.1 Detailed Description	189
8.11.2 Member Function/Subroutine Documentation	189
8.12 utilities::geotocpp Interface Reference	190
8.12.1 Detailed Description	191
8.12.2 Member Function/Subroutine Documentation	191
8.13 timedateutils::gregtojulday Interface Reference	192
8.13.1 Detailed Description	193
8.13.2 Member Function/Subroutine Documentation	193
8.14 parwind::hollanddata_t Type Reference	197

8.14.1 Detailed Description	198
8.14.2 Member Data Documentation	198
8.15 sortutils::indexx Interface Reference	202
8.15.1 Detailed Description	203
8.15.2 Member Function/Subroutine Documentation	203
8.16 pahm_messages::logmessage Interface Reference	208
8.16.1 Detailed Description	208
8.16.2 Member Function/Subroutine Documentation	209
8.17 pahm_messages::screenmessage Interface Reference	209
8.17.1 Detailed Description	210
8.17.2 Member Function/Subroutine Documentation	210
8.18 utilities::sphericaldistance Interface Reference	211
8.18.1 Detailed Description	211
8.18.2 Member Function/Subroutine Documentation	211
8.19 timedateutils::splitdatetimestring Interface Reference	214
8.19.1 Detailed Description	214
8.19.2 Constructor & Destructor Documentation	214
8.19.3 Member Function/Subroutine Documentation	215
8.20 sortutils::swap Interface Reference	216
8.20.1 Detailed Description	217
8.20.2 Member Function/Subroutine Documentation	217
8.21 timedateutils::timeconv Interface Reference	221
8.21.1 Detailed Description	221
8.21.2 Member Function/Subroutine Documentation	221
8.22 pahm_netcdfio::timedata_t Type Reference	224
8.22.1 Detailed Description	224
8.22.2 Member Data Documentation	224
9 File Documentation	225
9.1 dev_doc.md File Reference	225
9.2 mainpage.md File Reference	225
9.3 model.md File Reference	225
9.4 csv_module.F90 File Reference	225
9.4.1 Detailed Description	226
9.5 csv_module.F90	227
9.6 csv_parameters.F90 File Reference	243
9.6.1 Detailed Description	244
9.7 csv_parameters.F90	244
9.8 csv_utilities.F90 File Reference	244

9.8.1 Detailed Description	245
9.9 csv_utilities.F90	245
9.10 doxy_sample.F90 File Reference	247
9.11 doxy_sample.F90	247
9.12 driver_mod.F90 File Reference	248
9.12.1 Detailed Description	248
9.13 driver_mod.F90	249
9.14 global.F90 File Reference	251
9.14.1 Detailed Description	253
9.15 global.F90	253
9.16 mesh.F90 File Reference	257
9.16.1 Detailed Description	258
9.17 mesh.F90	258
9.18 messages.F90 File Reference	262
9.18.1 Detailed Description	263
9.19 messages.F90	264
9.20 netcdfio-nems.F90 File Reference	268
9.20.1 Macro Definition Documentation	269
9.21 netcdfio-nems.F90	269
9.22 netcdfio-orig.F90 File Reference	280
9.22.1 Macro Definition Documentation	281
9.23 netcdfio-orig.F90	281
9.24 netcdfio.F90 File Reference	291
9.24.1 Detailed Description	292
9.24.2 Macro Definition Documentation	292
9.25 netcdfio.F90	293
9.26 pahm.F90 File Reference	303
9.26.1 Detailed Description	304
9.26.2 Function/Subroutine Documentation	304
9.27 pahm.F90	305
9.28 parwind-orig.F90 File Reference	306
9.29 parwind-orig.F90	307
9.30 parwind.F90 File Reference	326
9.30.1 Detailed Description	327
9.31 parwind.F90	327
9.32 sizes.F90 File Reference	346
9.32.1 Detailed Description	347
9.33 sizes.F90	348
9.34 sortutils.F90 File Reference	351

9.34.1 Detailed Description	352
9.34.2 Function/Subroutine Documentation	352
9.35 sortutils.F90	353
9.36 timedateutils.F90 File Reference	372
9.36.1 Detailed Description	374
9.37 timedateutils.F90	374
9.38 utilities.F90 File Reference	388
9.38.1 Detailed Description	389
9.39 utilities.F90	390
9.40 vortex.F90 File Reference	421
9.40.1 Detailed Description	423
9.41 vortex.F90	423
Index	445

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 development 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 practices

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:

csv_module	6
csv_parameters	10
csv_utilities	12
pahm_drivermod	14
pahm_global	19
pahm_mesh	38
pahm_messages	44
pahm_netcdfio	54
pahm_sizes	60
pahm_vortex	66
parwind	89
sortutils	99
timedateutils	114
utilities	133

5 Data Type Index

5.1 Data Types List

Here are the data types with brief descriptions:

pahm_messages::allmessage	159
sortutils::arraycopy	160
sortutils::arrayequal	163
sortutils::arth	165
parwind::besttrackdata_t	168
pahm_sizes::comparereals	175
utilities::cpptogeo	177
csv_module::csv_file	179
csv_module::csv_string	186
pahm_netcdfio::filedata_t	187
pahm_sizes::fixnearwholereal	188

utilities::geotocpp	190
timedateutils::gregtojulday	192
parwind::hollanddata_t	197
sortutils::indexx	202
pahm_messages::logmessage	208
pahm_messages::screenmessage	209
utilities::sphericaldistance	211
timedateutils::splitdatetimestring	214
sortutils::swap	216
timedateutils::timeconv	221
pahm_netcdfio::timedata_t	224

6 File Index

6.1 File List

Here is a list of all files with brief descriptions:

csv_module.F90	
For reading and writing CSV files	225
csv_parameters.F90	
Various parameters	243
csv_utilities.F90	
Utility routines	244
doxy_sample.F90	247
driver_mod.F90	248
global.F90	251
mesh.F90	
Contains all the mesh related utilities	257
messages.F90	262
netcdfio-nems.F90	268
netcdfio-orig.F90	280
netcdfio.F90	291

pahm.F90	
Main PaHM program, calls Init, Run and Finalize procedures	303
parwind-orig.F90	306
parwind.F90	326
sizes.F90	
Contains the definitions of various number types and utilities used in PaHM	346
sortutils.F90	351
timedateutils.F90	372
utilities.F90	388
vortex.F90	421

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)]].

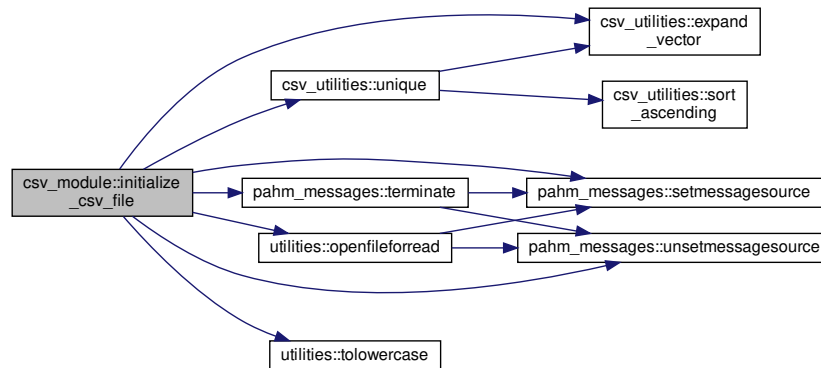
Parameters

<i>me</i>	The output csv_file structure
<i>quote</i>	Can only be one character (optional, default is ")
<i>delimiter</i>	Can only be one character (optional, default is ,)
<i>enclose_strings_in_quotes</i>	Logical flag; if true, all string cells will be enclosed in quotes (optional, default is T)
<i>enclose_all_in_quotes</i>	Logical flag; if true, <i>all</i> cells will be enclosed in quotes (optional, default is F)
<i>logical_true_string</i>	Logical flag; when writing a logical <code>true</code> value to a CSV file, this is the string to use (optional, default is T)
<i>logical_false_string</i>	Logical flag; when writing a logical <code>false</code> value to a CSV file, this is the string to use (optional, default is T)
<i>chunk_size</i>	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 ≤ 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.

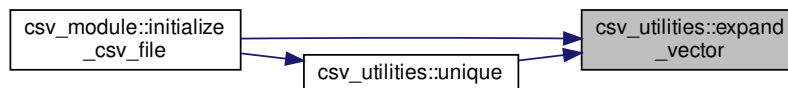
Parameters

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

```

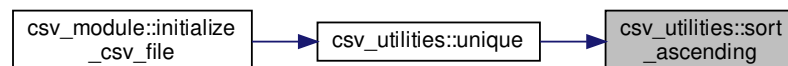
Sorts an integer array *ivec* in increasing order. Uses a basic recursive quicksort (with insertion sort for partitions with ≤ 20 elements).

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)

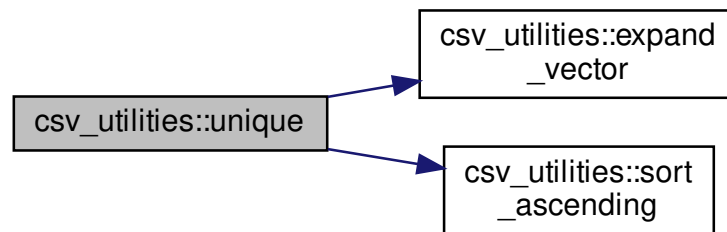
Returns only the unique elements of the vector.

Definition at line 103 of file [csv_utilities.F90](#).

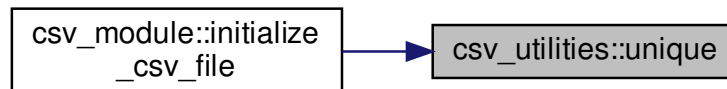
References [expand_vector\(\)](#), and [sort_ascending\(\)](#).

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`

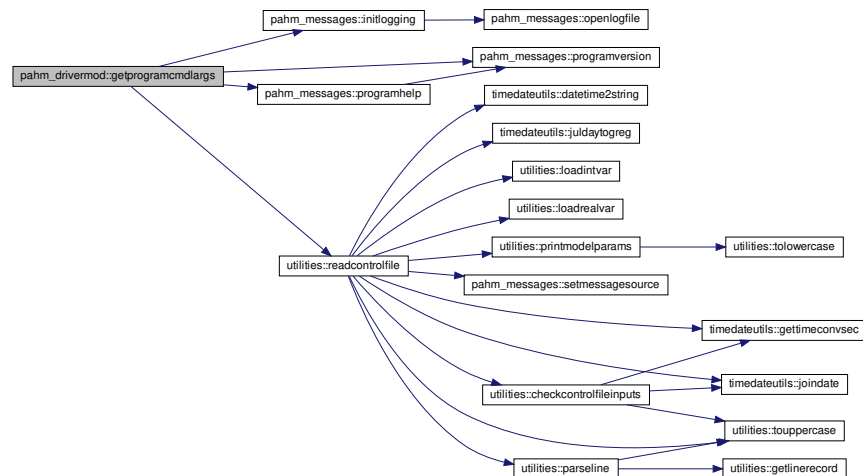
Prints on the screen the help system of the PaHM program.

Definition at line 40 of file [driver_mod.F90](#).

References [pahm_global::controlfilename](#), [pahm_messages::initlogging\(\)](#), [pahm_messages::programhelp\(\)](#), [pahm_messages::programversion](#) 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`

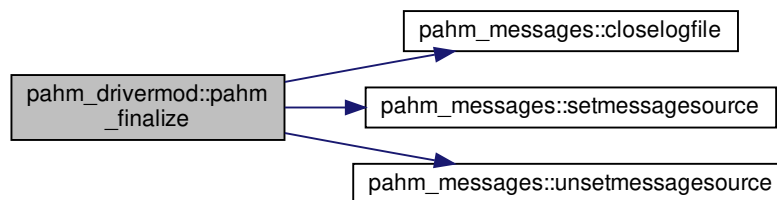
Subroutine to finalize a PaHM run.

Definition at line 194 of file `driver_mod.F90`.

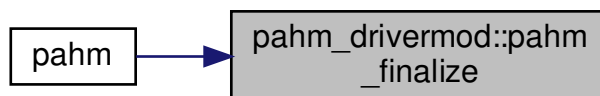
References `pahm_messages::closelogfile()`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

Referenced by `pahm()`.

Here is the call graph for this function:



Here is the caller graph for this function:



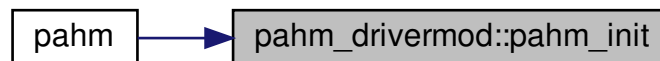
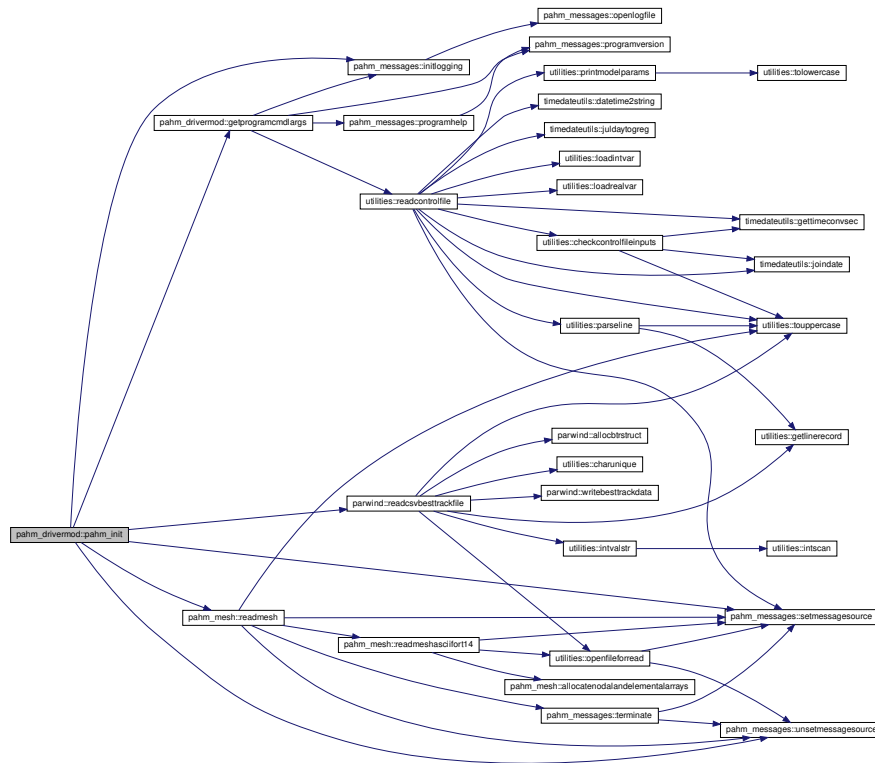
7.4.1.3 `pahm_init()` subroutine `pahm_drivermod::pahm_init`

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::unsetmessagesource()`.

Referenced by `pahm()`.

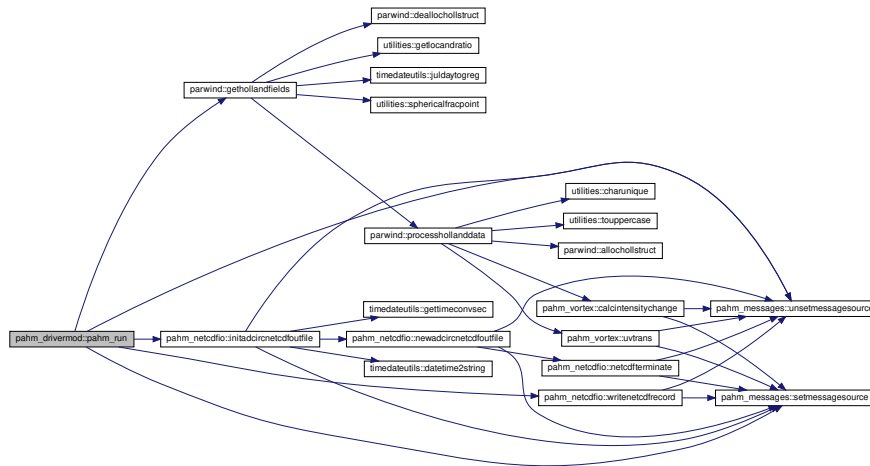


Subroutine to run PaHM (timestepping).

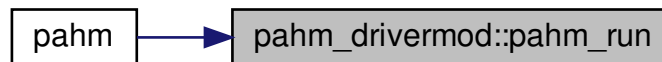
References `cnttimebegin`, `cnttimeend`, `pahm_messages::error`, `parwind::gethollandfields()`, `pahm_netcdfio::initadcircnetcdfoutfile()`, `pahm_global::modeltype`, `pahm_global::outfilename`, `pahm_global::outfilenamespecified`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, `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) [logfile](#) = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) [controlfile](#) = 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) `outfile` = BLANK
- integer `ncshuff` = 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

<i>atmT</i>	Air temperature ($^{\circ}C$)
<i>atmP</i>	Atmospheric pressure (<i>mbar</i>)
<i>relHum</i>	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(s2) 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::enddate_time = 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 = RMISV`

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::logmessage_2\(\)](#), 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 = RMISV`

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 = RMISV`

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 = RMISV`

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 = RMISV`

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::timeconvisev\(\)](#), 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.

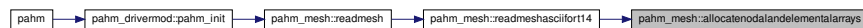
Mesh related memory allocation for any array that is dimensioned by the number of nodes in the mesh or the number of elements in the mesh.

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.

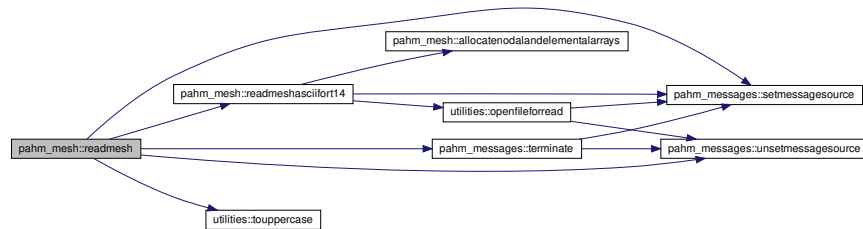
Read the mesh file for the specified model type (meshFileType) and in ASCII or NetCDF format (if applicable).

Definition at line 69 of file [mesh.F90](#).

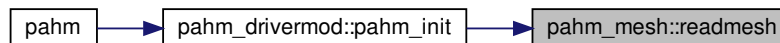
References [pahm_messages::error](#), [pahm_global::meshfileform](#), [pahm_global::meshfilename](#), [pahm_global::meshfilenamespecified](#), [pahm_global::meshfiletype](#), [readmeshasciifort14\(\)](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::terminate\(\)](#), [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.

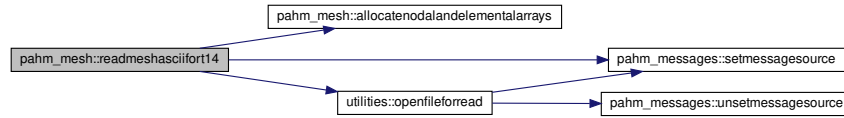
Reads the ADCIRC fort.14 mesh file and sets all mesh variables and arrays.

Definition at line 170 of file [mesh.F90](#).

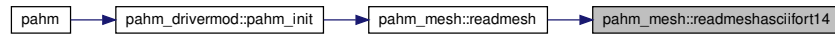
References [agrid](#), [allocatenodalandelementalarrays\(\)](#), [pahm_messages::info](#), [ismeshok](#), [pahm_global::lun_inp](#), [maxfacenodes](#), [pahm_global::meshfilename](#), [ne](#), [np](#), [utilities::openfileforread\(\)](#), [pahm_messages::setmessagesource\(\)](#), [sfea](#), [sfea0](#), [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 writing.
- 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 [sourcenum](#)
- 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`

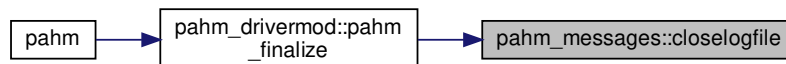
Closes an opened log file.

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](#).

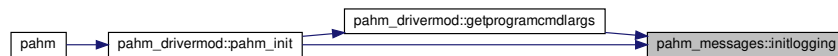
References [loginitcalled](#), [loglevelnames](#), [openlogfile\(\)](#), and [sourcenumber](#).

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 `UnsetMessageSource` 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 writing.

Definition at line 113 of file [messages.F90](#).

References [error](#), [pahm_global::logfile](#), [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](#).

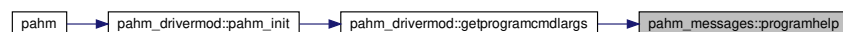
References [pahm_global::lun_screen](#), and [programversion\(\)](#).

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`

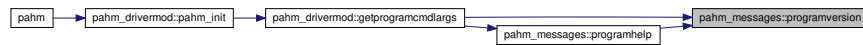
Prints on the screen the versioning information of the program.

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 UnsetMessageSource 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](#).

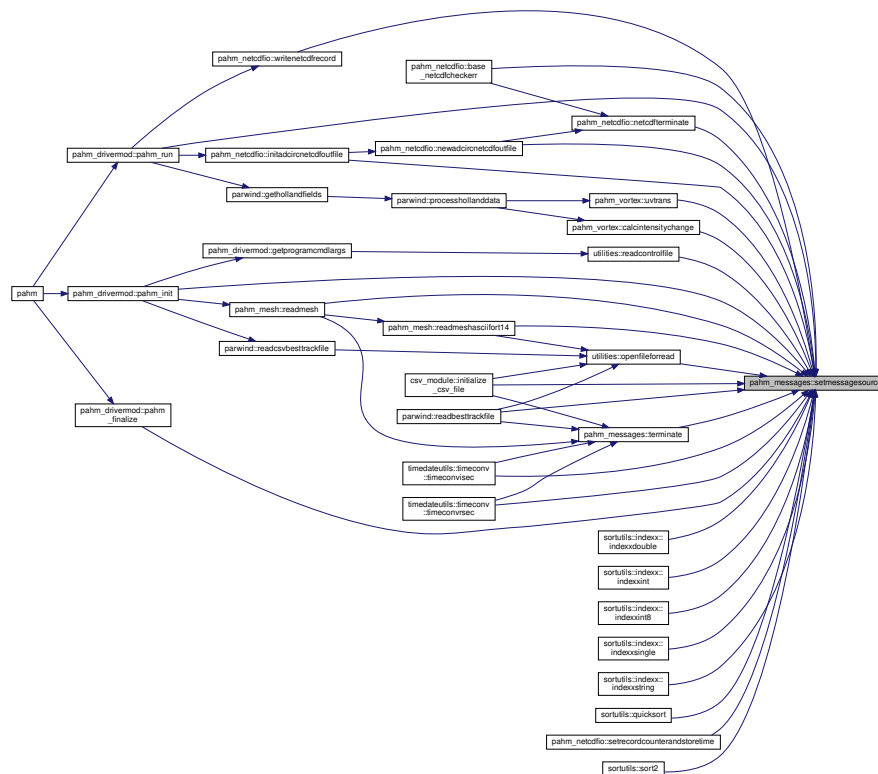
Sets the name of the subroutine that is writing log and/or screen messages.

Definition at line 349 of file messages.F90.

References [messagesources](#), and [sourcenumbers](#).

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::newadcircnetcdfoutfile()`, `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`

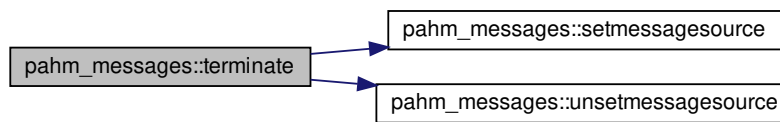
Terminates the calling program when a fatal error is encountered.

Definition at line 452 of file [messages.F90](#).

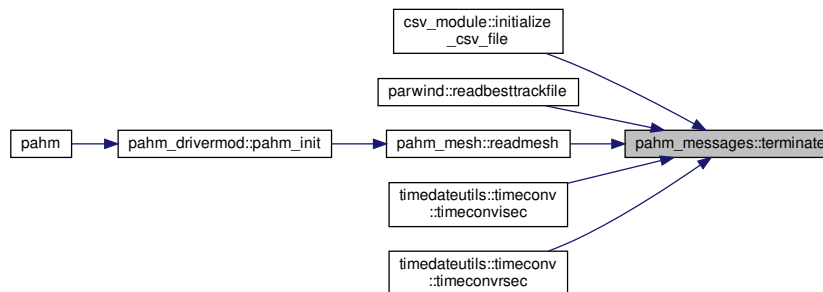
References [error](#), [setmessagesource\(\)](#), and [unsetmessagesource\(\)](#).

Referenced by [csv_module::initialize_csv_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [pahm_mesh::readmesh\(\)](#), [timedateutils::timeconv::timeconv](#) 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`

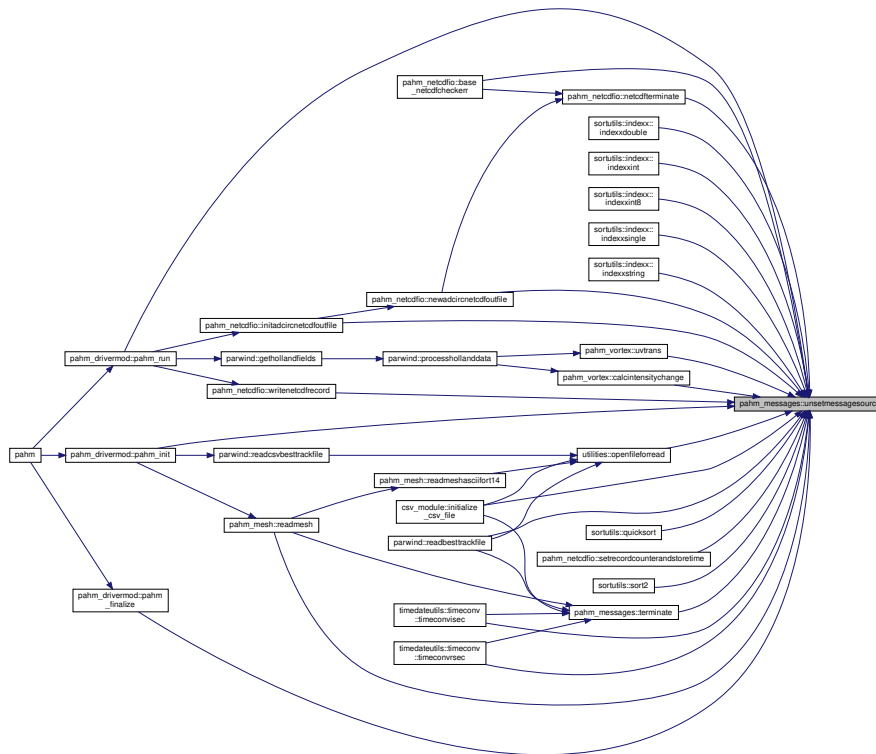
Removes the name of the subroutine that is no longer active.

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](#).

Here is the caller graph for this function:



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::timeconvsec\(\)](#), [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::timeconvise\(\)](#), [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

<i>ierr</i>	The error status from a NetCDF library call
<i>file</i>	The name of the file the error occurred
<i>line</i>	The line number of the file the error occurred

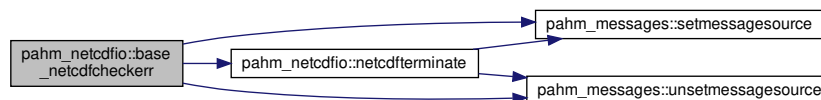
Returns

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

<i>adcircOutFile</i>	The name of the file to be initialized. The file is first created by calling <code>NewAdcircNetCDFOutFile</code> .
----------------------	--

Returns

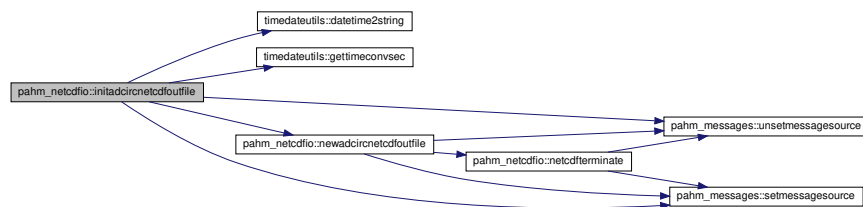
`adcircOutFile`: The renamed input file.

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

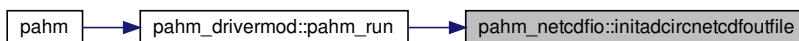
References `pahm_mesh::agrid`, `timedateutils::datetime2string()`, `timedateutils::gettimeconvsec()`, `pahm_sizes::imissv`, `pahm_global::ncdeflate`, `pahm_global::ncdlevel`, `pahm_global::ncshuffle`, `pahm_global::ncvarnam_pres`, `pahm_global::ncvarnam_wndx`, `pahm_global::ncvarnam_wndy`, `pahm_mesh::ne`, `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`

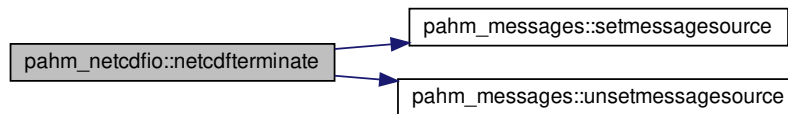
Terminates the program on NetCDF error.

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

<code>ncID</code>	The NetCDF ID of the file to be created (output)
<code>adcircOutFile</code>	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

<i>ncID</i>	The ID of the NetCDF file
<i>f</i>	The file structure
<i>t</i>	The time structure

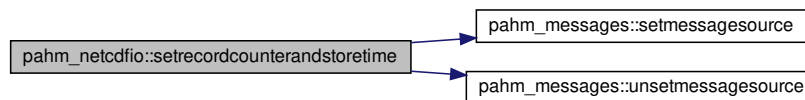
Returns

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::setmessagesource](#) 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

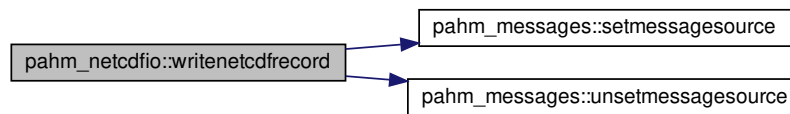
<i>adcircOutFile</i>	The name of the NetCDF file
<i>timeLoc</i>	The time record to write

Definition at line 763 of file [netcdfio-nems.F90](#).

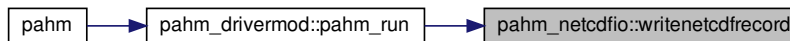
References [mytime](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::unsetmessagesource\(\)](#), [pahm_global::wpress](#), [pahm_global::wvelx](#), and [pahm_global::wvely](#).

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 [compare_reals](#)
- interface [fix_near_wholereal](#)

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

<code>rVal1</code>	The first value (double precision number) in the comparison
<code>rVal2</code>	The second value (double precision number) in the comparison
<code>eps</code>	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

<i>rVal1</i>	The first value (single precision number) in the comparison
<i>rVal2</i>	The second value (single precision number) in the comparison
<i>eps</i>	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

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

Returns

myValOut : Either **rVal** or its nearest integer **iVal** 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

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

Returns

myValOut : Either **rVal** or its nearest integer **iVal** 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::timeconviseec\(\)](#), 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 numerical 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 numerical 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 cyclostrophic 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 [interp](#) (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 numerical 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

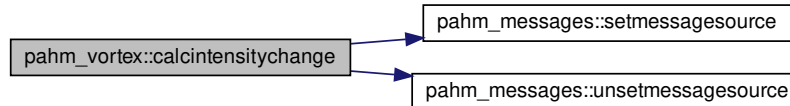
On output: calcInt the calculated intensity change (df/dt) status error status (0 means no error)

Definition at line 79 of file [vortex.F90](#).

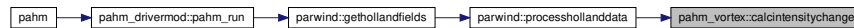
References [pahm_global::deg2rad](#), [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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.

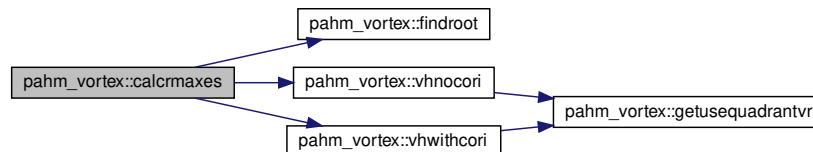
On input: none

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 [vmb1](#).

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 cyclostrophic balance.

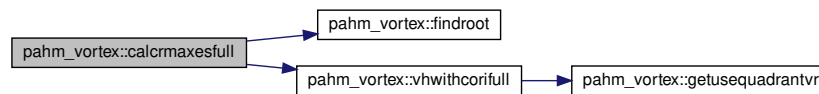
On input: none

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 [vmb1](#).

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.

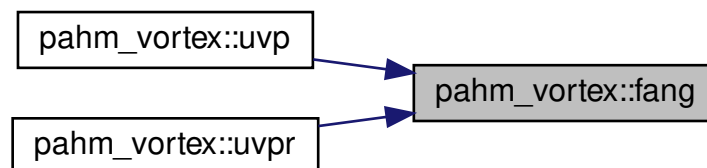
On input: r distance from center of storm rmx radius of maximum winds

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 sought x1 left side of interval x2 right side of interval dx x increment for march

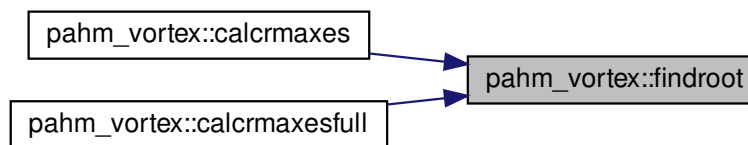
On output: a left side of interval that brackets the root b right side of interval that brackets the root FindRoot root returned

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 [vmb14](#).

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](#).

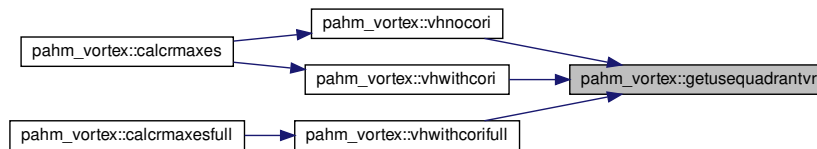
7.10.1.14 getusequadrantvr() `logical function pahm_vortex::getusequadrantvr`

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 [vmbi](#).

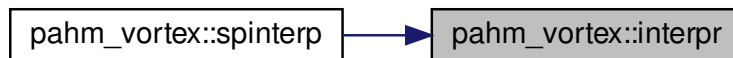
7.10.1.16 **interp()** `real(sz) function pahm_vortex::interp (`
 `real(sz), dimension(npoints, 4), intent(in) quadVal,`
 `integer, intent(in) quadSel,`
 `real(sz), intent(in) quadDis)`

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

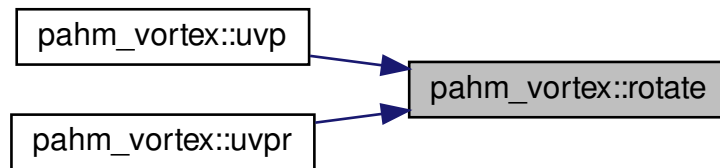
On output: xr x component of rotated vector yr y component of rotated vector

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](#), [interp\(\)](#), [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)

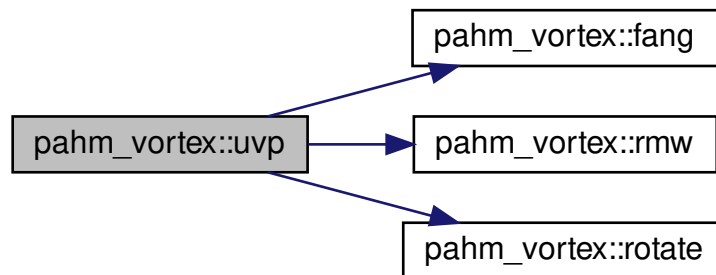
Internal parameters: dampRadii How far out (# of rMax radii) to begin damping out the translational velocity

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)

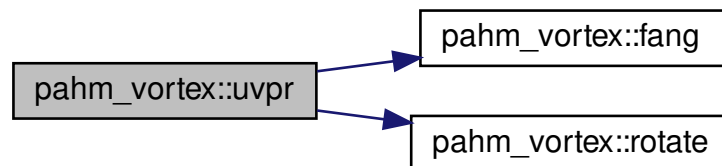
Internal parameters: dampRadii How far out (# of rMax radii) to begin damping out the translational velocity

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 numerical 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

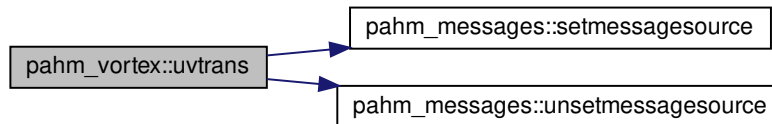
On output: u x component of the translational velocities (m/s) v y component of the translational velocities (m/s) status error status (0 means no error)

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) time2 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 )

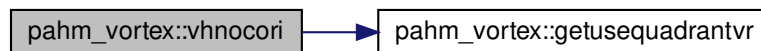
```

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(npnts) 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 vmb14 `real(sz), dimension(npoints, 4) pahm_vortex::vmb14`

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 [deallocchollstruct](#) (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

Panagiotis Velissariou 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

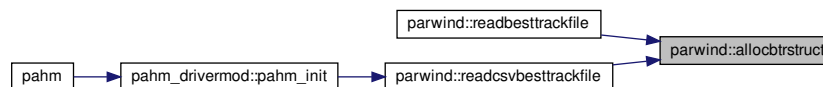
Parameters

<i>str</i>	The best track structure of type <code>BestTrackData_T</code>
<i>nRec</i>	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

<i>str</i>	The holland structure of type HollandData_T
<i>nRec</i>	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

<i>str</i>	The best track structure of type BestTrackData_T
------------	--

Definition at line 1390 of file [parwind-orig.F90](#).

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

Subroutine to deallocate memory of an allocated holland structure.

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

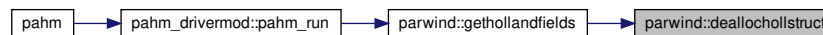
Parameters

<i>str</i>	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.

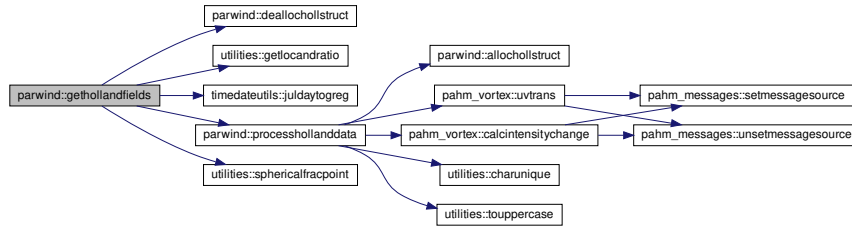
Assumes geographical coordinates.

Definition at line 897 of file [parwind-orig.F90](#).

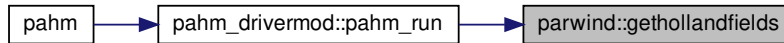
References [pahm_global::backgroundatmpress](#), [pahm_global::basee](#), [pahm_global::besttrackfilename](#), [pahm_global::bladjustfac](#), [deallochoollstruct\(\)](#), [pahm_global::deg2rad](#), [utilities::getlocandratio\(\)](#), [pahm_global::gravity](#), [pahm_mesh::ismeshok](#), [timedateutils::juldaytogreg\(\)](#), [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

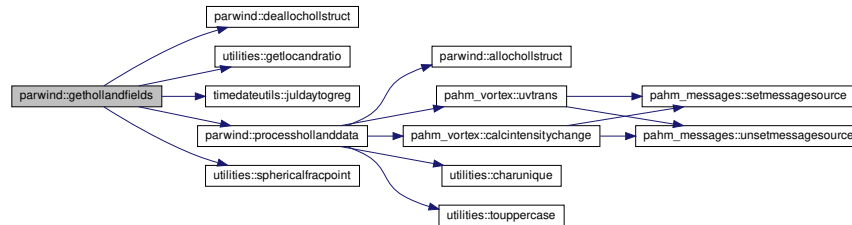
<i>timeIDX</i>	The time location to generate the fields for
----------------	--

Definition at line 947 of file [parwind.F90](#).

References [pahm_global::backgroundatmpress](#), [pahm_global::basee](#), [pahm_global::besttrackfilename](#), [pahm_global::bladjustfac](#), [deallocollstruct\(\)](#), [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

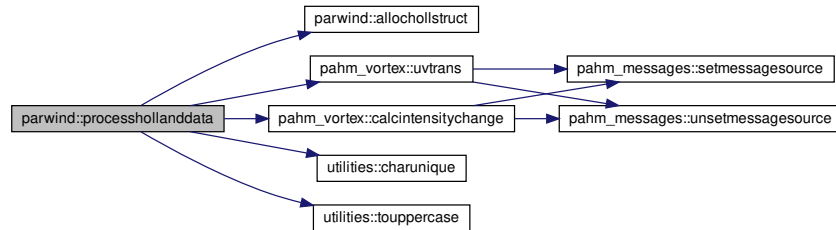
<i>idTrFile</i>	The ID of the input track file (1, 2, ...)
<i>strOut</i>	The HollandData_T structure that stores all Holland model generated data (output)
<i>status</i>	Error status, 0 = no error (output)

Definition at line 678 of file [parwind-orig.F90](#).

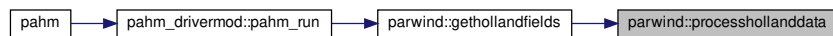
References [allochollstruct\(\)](#), [besttrackdata](#), [pahm_vortex::calcintensitychange\(\)](#), [utilities::charunique\(\)](#), [pahm_global::kt2ms](#), [pahm_global::nbtrfiles](#), [pahm_global::nm2m](#), [utilities::touppercase\(\)](#), and [pahm_vortex::uvtrans\(\)](#).

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.

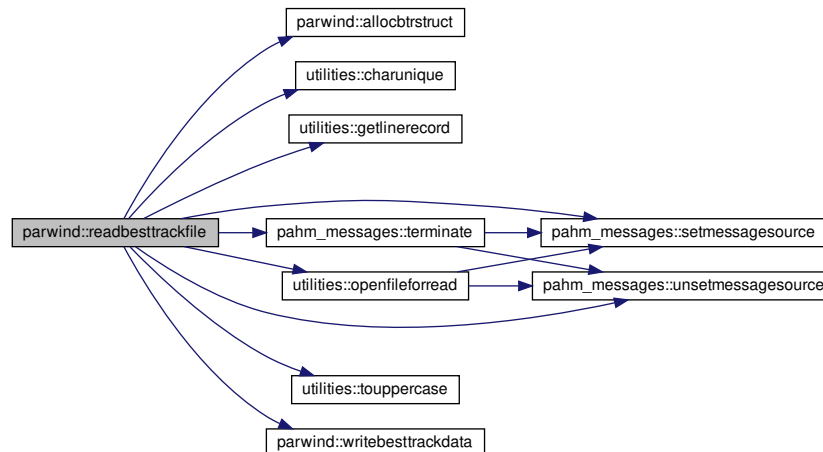
It uses fortran format statements (old approach) to read the ATCF formatted track files as follows:

- 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

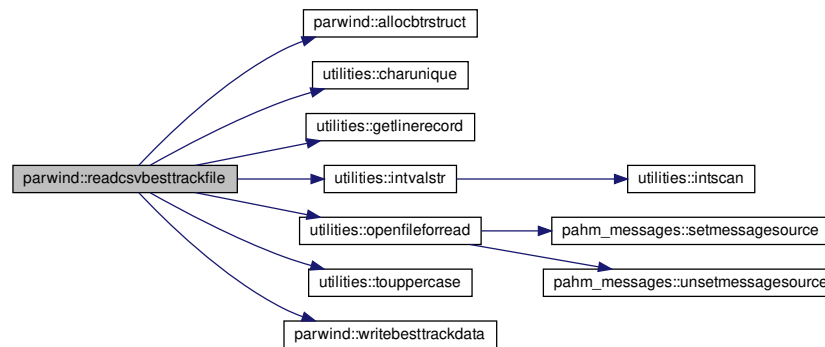
- Skips lines that are time repeats. ???PV check
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 409 of file [parwind-orig.F90](#).

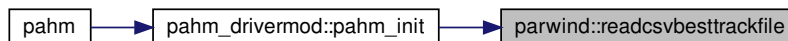
References [allocbtrstruct\(\)](#), [besttrackdata](#), [pahm_global::besttrackfilename](#), [utilities::charunique\(\)](#), [utilities::getlinerecord\(\)](#), [utilities::intvalstr\(\)](#), [pahm_global::nbtrfiles](#), [utilities::openfileforread\(\)](#), [utilities::touppercase\(\)](#), and [writebesttrackdata\(\)](#).

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-processed best track data to file.

On Input:

inpFile The name of the input best track 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

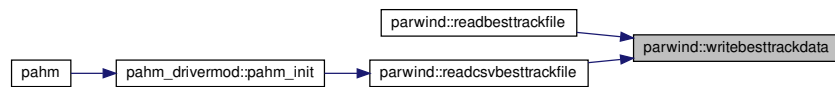
<i>inpFile</i>	The name of the input best track file
<i>btrStruc</i>	The "adjusted" best track data structure that corresponds to the <i>inpFile</i>
<i>suffix</i>	The suffix (optional) to be appended to the <i>inpFile</i> (default '_adj')

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) [arhint](#) (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

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

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

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

<i>arr1</i>	The first array in the comparison (double precision)
<i>arr2</i>	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

<i>arr1</i>	The first array in the comparison (integer)
<i>arr2</i>	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

<i>arr1</i>	The first array in the comparison (single precision)
<i>arr2</i>	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

<i>first</i>	The value of the first term (double precision)
<i>increment</i>	The value of the increment (double precision)
<i>n</i>	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

<i>first</i>	The value of the first term (integer)
<i>increment</i>	The value of the increment (integer)
<i>n</i>	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

<i>first</i>	The value of the first term (single precision)
<i>increment</i>	The value of the increment (single precision)
<i>n</i>	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

<i>arr1D</i>	The array to be indexed (double precision)
<i>idx1D</i>	The array of "indexed" indexes of arr1D (output)
<i>status</i>	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

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

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

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

<i>arr1D</i>	The array to be indexed (string)
<i>idx1D</i>	The array of "indexed" indexes of <code>arr1D</code> (output)
<i>status</i>	The error status, no error: <code>status = 0</code> (output)
<i>caseSens</i>	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

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

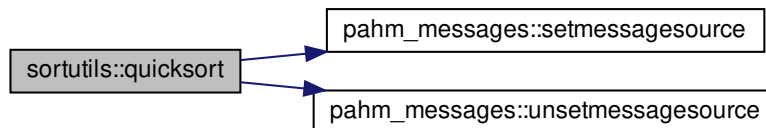
Note

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

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

Returns

The two sorted arrays *arr1D* and *slv1D*

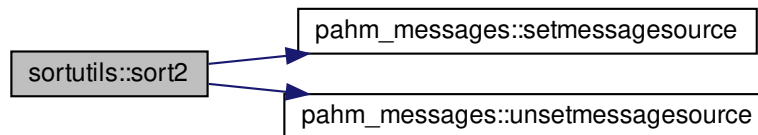
Note

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

<i>str1</i>	The first string in the comparison
<i>str2</i>	The second string in the comparison
<i>mSensitive</i>	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](#).

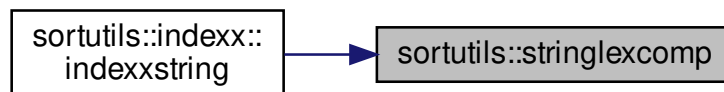
References [utilities::touppercase\(\)](#).

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

<i>a</i>	The first value to be swapped (double precision)
<i>b</i>	The second value to be swapped (double precision)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (double precision)
<i>b</i>	The second 1D array to be swapped (double precision)
<i>mask</i>	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

<i>a</i>	The first value to be swapped (integer)
<i>b</i>	The second value to be swapped (integer)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (integer)
<i>b</i>	The second 1D array to be swapped (integer)
<i>mask</i>	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

<i>a</i>	The first value to be swapped (single precision)
<i>b</i>	The second value to be swapped (single precision)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (single precision)
<i>b</i>	The second 1D array to be swapped (single precision)
<i>mask</i>	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 timdateutils Module Reference

Data Types

- interface [gregtojulday](#)
- interface [splitdatetimestring](#)
- interface [timeconv](#)

Functions/Subroutines

- subroutine [timeconvsec](#) (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 timdateutils::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

<i>year</i>	The year (YYYY)
<i>month</i>	The month of the year (MM)
<i>day</i>	The day of the month (DD)
<i>hour</i>	The hour of the day (hh) (optional - 0 is substituted if not supplied)
<i>min</i>	The minute of the hour (mm) (optional - 0 is substituted if not supplied)
<i>sec</i>	The second of the minute (ss) (optional - 0 is substituted if not supplied)
<i>sep</i>	The separation character between the date part and the time part
<i>units</i>	The units part to be prepended to the datetime string in the form '<units> since'
<i>zone</i>	The timezone to use (default none/UTC, optional)
Optional error status	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

<i>iYear</i>	The year (YYYY, integer, $1582 \leq YYYY$)
<i>iMonth</i>	The month of the year (MM, integer, $1 \leq MM \leq 12$)
<i>iDay</i>	The day of the month (DD, integer, $1 \leq DD \leq 31$)

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 `timdateutils::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

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

Definition at line 1011 of file [timdateutils.F90](#).

References [juldaytogreg\(\)](#).

Here is the call graph for this function:



7.13.1.4 elapsedsecs() real(sz) function `timdateutils::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

<i>inTime1</i>	The start time (real)
<i>inTime2</i>	The end time (real)

Parameters

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

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](#).

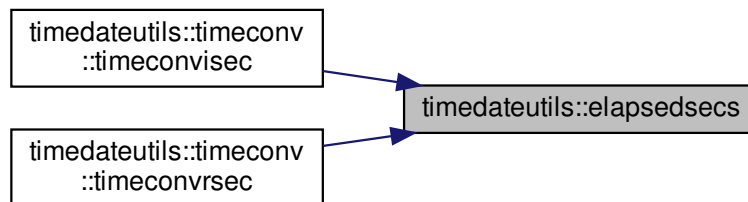
References [gettimeconvsec\(\)](#).

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

Parameters

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

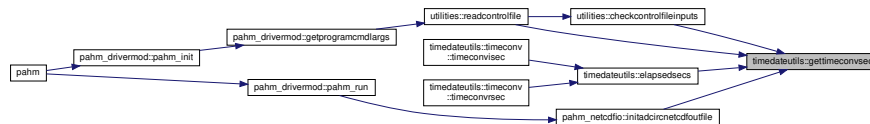
Returns

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 GregToJulDaylSEC but the seconds number is real to allow for second fractions.

Parameters

<i>iDate</i>	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)
<i>iTime</i>	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

<i>mJD</i>	<p>Flag to use a modified julian day number or not</p> <p>To use a modified julian day number use: <code>mJD >= 1</code> otherwise use: <code>mJD < 1</code> default: <code>mJD = 0</code></p> <p>The modified julian day number (MJD) was defined in the mid 1950's in the interests of astronomy and space science as <code>MJD = JD - 2400000.5</code>. 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.</p>
------------	---

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 useful to compute differences between dates.

Parameters

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

Parameters

<i>mJD</i>	<p>Flag to use a modified julian day number or not</p> <p>To use a modified julian day number use: <code>mJD >= 1</code> otherwise use: <code>mJD < 1</code> default: <code>mJD = 0</code></p> <p>The modified julian day number (MJD) was defined in the mid 1950's in the interests of astronomy and space science as <code>MJD = JD - 2400000.5</code>. 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.</p>
------------	---

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 useful to compute differences between dates.

Similar to `GregToJulDayISEC` but the seconds number is real to allow for second fractions.

Parameters

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

Parameters

<i>rSec</i>	The second of the minute (ss, real, 0 <= ss <= 59)
<i>mJD</i>	<p>Flag to use a modified julian day number or not</p> <p>To use a modified julian day number use: mJD >= 1 otherwise use: mJD < 1 default: mJD = 0</p> <p>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.</p>

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

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

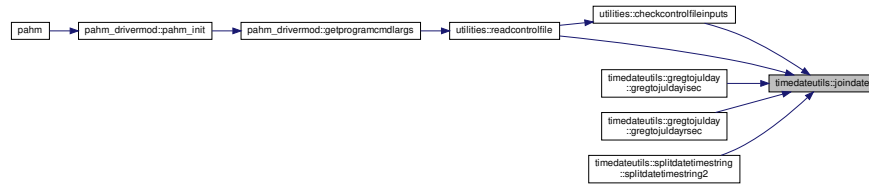
Returns

myValOut: The integer date (YYYYMMDD)

Definition at line [1241](#) of file [timedateutils.F90](#).

Referenced by `utilities::checkcontrolfileinputs()`, `timedateutils::gregtojulday::gregtojuldaysec()`, `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 useful to compute differences between dates.

Parameters

<i>julDay</i>	The Julian day number (double).
<i>mJD</i>	Flag to use a modified julian day number or not To use a modified julian day number use: <code>mJD >= 1</code> otherwise use: <code>mJD < 1</code> default: <code>mJD = 0</code> The modified julian day number (MJD) was defined in the mid 1950's in the interests of astronomy and space science as <code>MJD = JD - 2400000.5</code> . 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.
<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY, output)
<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12, output)
<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31, output)
<i>iHour</i>	The hour of the day (hh, integer, 0 <= hh <= 23, output)
<i>iMin</i>	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
<i>iSec</i>	The second of the minute (ss, integer, 0 <= ss <= 59, output)

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

<i>iYear</i>	The year (YYYY, integer, $1582 \leq \text{YYYY}$)
--------------	--

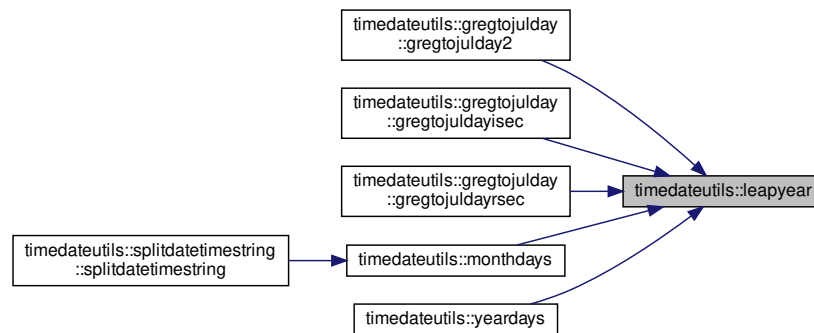
Returns

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::gregmonthdays\(\)](#), and [yeardays\(\)](#).

Here is the caller graph for this function:



7.13.1.12 monthdays() `integer function timdateutils::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

<i>iYear</i>	The year (YYYY, integer, $1582 \leq YYYY$)
<i>iMonth</i>	The month of the year (MM, integer, $1 \leq MM \leq 12$)

Returns

myVal: The days of the month

Definition at line 403 of file `timdateutils.F90`.

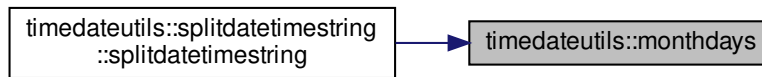
References `pahm_sizes::imissv`, and `leapyear()`.

Referenced by `timdateutils::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 timdateutils::preprocessdatetimestring(`
(
 `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

<i>inDateTime</i>	The input date string
-------------------	-----------------------

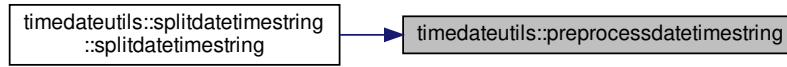
Returns

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

Definition at line 1186 of file [timdateutils.F90](#).

Referenced by [timdateutils::splitdatetimestring::splitdatetimestring\(\)](#).

Here is the caller graph for this function:



7.13.1.14 splitdate() subroutine timdateutils::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

<i>inDate</i>	The integer date (YYYYMMDD)
<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY, output)
<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12, output)
<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31, output)

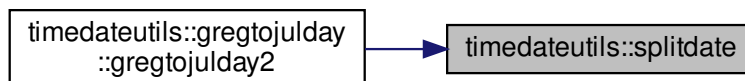
Note

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

Definition at line 1281 of file [timdateutils.F90](#).

Referenced by [timdateutils::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

<i>inDateTime</i>	The input date string: YYYYMMDDhhmmss
<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY, output)
<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12, output)
<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31, output)
<i>iHour</i>	The hour of the day (hh, integer, 0 <= hh <= 23, output)
<i>iMin</i>	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
<i>iSec</i>	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

<i>inDateTime</i>	The input date string: YYYYMMDDhhmmss
<i>iDate</i>	The integer date (YYYYMMDD, output)
<i>iTime</i>	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

<i>iYear</i>	The year (integer)
<i>iMonth</i>	The month of the year (1-12, integer)
<i>iDay</i>	The day of the month (1-31, integer)
<i>iHour</i>	The hour of the day (0-23, integer)
<i>iMin</i>	The minute of the hour (0-59, integer)
<i>iSec</i>	The second of the minute (0-59, integer)
<i>timeSec</i>	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

<i>iYear</i>	The year (integer)
<i>iMonth</i>	The month of the year (1-12, integer)
<i>iDay</i>	The day of the month (1-31, integer)
<i>iHour</i>	The hour of the day (0-23, integer)
<i>iMin</i>	The minute of the hour (0-59, integer)
<i>rSec</i>	The second of the minute (0-59, real)
<i>timeSec</i>	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

<i>iYear</i>	The year (YYYY, integer, $1582 \leq \text{YYYY}$)
--------------	--

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 timdateutils::firstgregtime = 0 * 10000 + 0 * 100 + 0

Definition at line 45 of file [timdateutils.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#).

7.13.2.3 mdjdate integer, parameter timdateutils::mdjdate = UNIXDATE

Definition at line 80 of file [timdateutils.F90](#).

7.13.2.4 mdjoffset real(hp), parameter timdateutils::mdjoffset = OFFUNIXJULDAY

Definition at line 82 of file [timdateutils.F90](#).

Referenced by [timdateutils::gregtojulday::gregtojulday2\(\)](#), [timdateutils::gregtojulday::gregtojuldayisec\(\)](#), [timdateutils::gregtojulday::gregtojuldayisec\(\)](#), [timdateutils::gregtojulday::gregtojuldayisec\(\)](#), and [juldaytogreg\(\)](#).

7.13.2.5 mdjtime integer, parameter timdateutils::mdjtime = UNIXTIME

Definition at line 81 of file [timdateutils.F90](#).

7.13.2.6 modeldate integer, parameter timdateutils::modeldate = 1990 * 10000 + 1 * 100 + 1

Definition at line 65 of file [timdateutils.F90](#).

7.13.2.7 modeltime integer, parameter timdateutils::modeltime = 0 * 10000 + 0 * 100 + 0

Definition at line 66 of file [timdateutils.F90](#).

7.13.2.8 modjuldate integer, parameter timdateutils::modjuldate = 1858 * 10000 + 11 * 100 + 17

Definition at line 53 of file [timdateutils.F90](#).

7.13.2.9 modjultime integer, parameter timdateutils::modjultime = 0 * 10000 + 0 * 100 + 0

Definition at line 54 of file [timdateutils.F90](#).

7.13.2.10 offfirstgregday real(hp), parameter timdateutils::offfirstgregday = 2299150.5_HP

Definition at line 46 of file [timdateutils.F90](#).

Referenced by [juldaytogreg\(\)](#).

7.13.2.11 offmodeljulday real(hp), parameter timdateutils::offmodeljulday = 2447892.5_HP

Definition at line 67 of file [timdateutils.F90](#).

7.13.2.12 offmodjulday real(hp), parameter timdateutils::offmodjulday = 2400000.5_HP

Definition at line 55 of file [timdateutils.F90](#).

7.13.2.13 offunixjulday real(hp), parameter timdateutils::offunixjulday = 2440587.5_HP

Definition at line 61 of file [timdateutils.F90](#).

7.13.2.14 unixdate integer, parameter timdateutils::unixdate = 1970 * 10000 + 1 * 100 + 1

Definition at line 59 of file [timdateutils.F90](#).

7.13.2.15 unixtime integer, parameter timdateutils::unixtime = 0 * 10000 + 0 * 100 + 0

Definition at line 60 of file [timdateutils.F90](#).

7.13.2.16 usemodjulday `integer, parameter timdateutils::usemodjulday = 0`

Definition at line 72 of file [timdateutils.F90](#).

Referenced by [timdateutils::gregtojulday::gregtojulday2\(\)](#), [timdateutils::gregtojulday::gregtojuldayisec\(\)](#), [timdateutils::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 `sphericalfractpoint` (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

<i>inpVec</i>	The input 1D string array
<i>outVec</i>	The output 1D string array of the unique elements (output)
<i>idxVec</i>	The 1D array of indexes of the unique elements in the <i>inpVec</i> array (output)

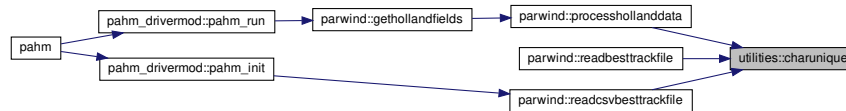
Returns

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

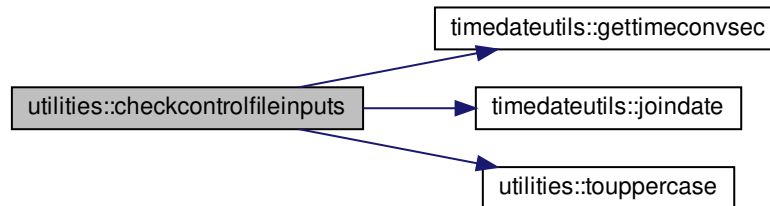
Definition at line 1140 of file [utilities.F90](#).

References [pahm_global::backgroundatmpress](#), [pahm_global::begsimspecified](#), [pahm_global::begsimtime](#), [pahm_global::besttrackfilename](#), [pahm_global::besttrackfilenameespecified](#), [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](#),

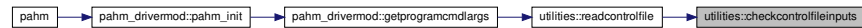
[pahm_global::outdt](#), [pahm_global::outfilename](#), [pahm_global::outfilenamespecified](#), [pahm_global::refdate](#), [pahm_global::refdatetime](#), [pahm_global::reftime](#), [pahm_global::rhoair](#), [pahm_global::rhowater](#), [touppercase\(\)](#), and [pahm_global::unittime](#).

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

<i>inpLon</i>	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

<i>x</i>	X coordinate: x (m) - real, 1D array
<i>y</i>	Y coordinate: y (m) - real, 1D array
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>lat</i>	Latitude (degrees north) - real, 1D array (output)
<i>lon</i>	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

<i>x</i>	X coordinate: x (m) - real, scalar
<i>y</i>	Y coordinate: y (m) - real, scalar
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>lat</i>	Latitude (degrees north) - real, scalar (output)
<i>lon</i>	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

<i>String</i>	The input string
<i>Pos</i>	The position in the input string where the scanning begins
<i>Value</i>	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

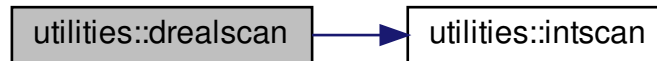
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

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

<i>String</i>	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

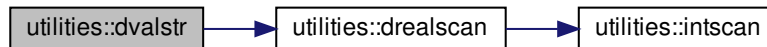
See also

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

<i>lat</i>	Latitude (degrees north) - real, 1D array
<i>lon</i>	Longitude (degrees east) - real, 1D array
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>x</i>	Calculated X coordinate: x (m) - real, 1D array (output)
<i>y</i>	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

<i>lat</i>	Latitude (degrees north) - real, scalar
<i>lon</i>	Longitude (degrees east) - real, scalar
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>x</i>	Calculated X coordinate: x (m) - real, scalar (output)
<i>y</i>	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) inLine,
    character(len=len(inline)), 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

<i>inLine</i>	The input text line
<i>lastCommFlag</i>	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
<i>outLine</i>	The output line (the left adjusted input line)

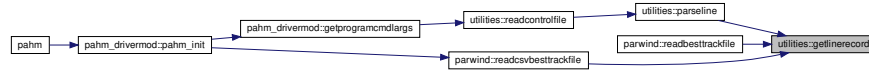
Returns

`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: $\text{VAR}(\text{estimated}) = \text{VAR}(\text{idx1}) + \text{wtRatio} * (\text{VAR}(\text{idx2}) - \text{VAR}(\text{idx1}))$.

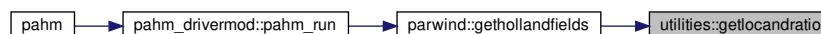
Parameters

<i>val</i>	The value to search for, such that $\text{arrVal}(\text{idx1}) \leq \text{val} \leq \text{arrVal}(\text{idx2})$
<i>arrVal</i>	The one-dimensional array to search (PV ordered in ascending order?)
<i>idx1</i>	The index of the lowest array bound such that: $\text{arrVal}(\text{idx1}) \leq \text{val}$ (output)
<i>idx2</i>	The index of the highest array bound such that: $\text{arrVal}(\text{idx2}) \geq \text{val}$ (output)
<i>wtRatio</i>	The ratio factor used in the linear interpolation calculation: $\text{VAR}(\text{estimated}) = \text{VAR}(\text{idx1}) + \text{wtRatio} * (\text{VAR}(\text{idx2}) - \text{VAR}(\text{idx1}))$ 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

<i>String</i>	The input string
<i>Pos</i>	The position in the input string where the scanning begins
<i>Signed</i>	The sign (+, -) of the numeric string, if present
<i>Value</i>	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

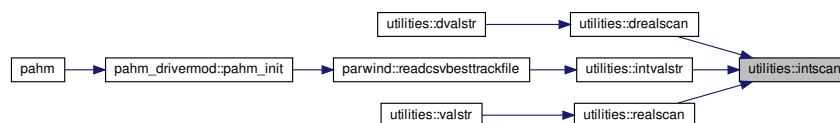
See also

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 [drealstr\(\)](#), [intvalstr\(\)](#), and [realstr\(\)](#).

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

<i>String</i>	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

<i>nInp</i>	Number of input values
<i>vInp</i>	Array of input values
<i>nOut</i>	Number of output values
<i>vOut</i>	Array of output values (integer, output)

Returns

nValsOut: Number of processed output values

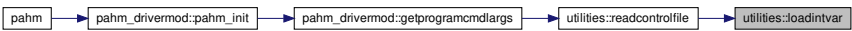
Note

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

<i>nInp</i>	Number of input values
<i>vInp</i>	Array of input values
<i>nOut</i>	Number of output values
<i>vOut</i>	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

<i>nInp</i>	Number of input values
<i>vInp</i>	Array of input values
<i>nOut</i>	Number of output values
<i>vOut</i>	Array of output values (real, output)

Returns

nValsOut: Number of processed output values

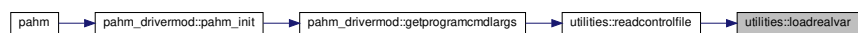
Note

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

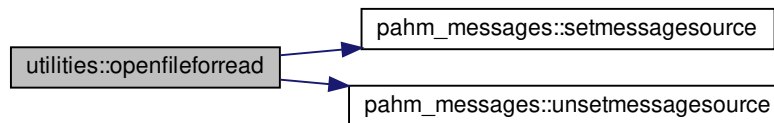
<i>lun</i>	The logical unit number (LUN) to use
<i>fileName</i>	The full pathname of the input file
<i>errorIO</i>	The error status, no error: status = 0 (output)

Definition at line 68 of file `utilities.F90`.

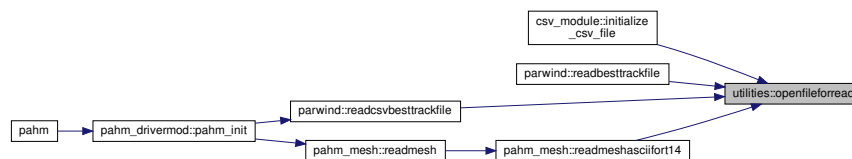
References `pahm_messages::error`, `pahm_messages::info`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

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

<i>inpLine</i>	The input text line
<i>outLine</i>	The output line, left adjusted input line (output)
<i>keyWord</i>	The keyword to extract settings for (input/output)
<i>nVal</i>	The number of values provided for the keyword (input/output)
<i>cVal</i>	String array (cVal(nVal)) that holds the string values provided for the keyword (input/output)
<i>rVal</i>	Real array (rVal(nVal)) that holds the values provided for the keyword (input/output)

Returns

myStatus: The error status, no error: status = 0

Note

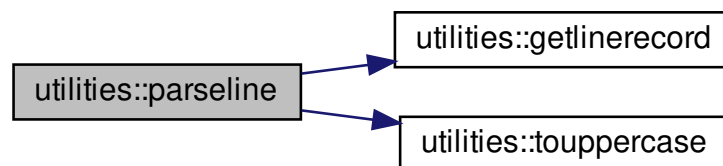
Adopted from the ROMS source (Utility/inp_par.F, decode_line)

Definition at line 869 of file [utilities.F90](#).

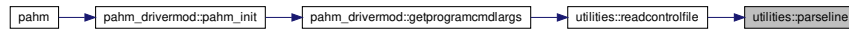
References [getlinerecord\(\)](#), and [touppercase\(\)](#).

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

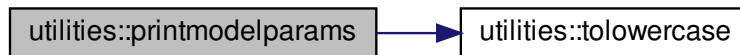
This subroutine prints on the screen the values of the program's parameters.

Definition at line 644 of file [utilities.F90](#).

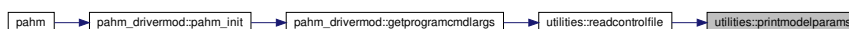
References [pahm_global::backgroundatmpress](#), [pahm_global::begdatespecified](#), [pahm_global::begdatetime](#), [pahm_global::begday](#), [pahm_global::beghour](#), [pahm_global::begmin](#), [pahm_global::begmonth](#), [pahm_global::begsec](#), [pahm_global::begsimspecified](#), [pahm_global::begsimtime](#), [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

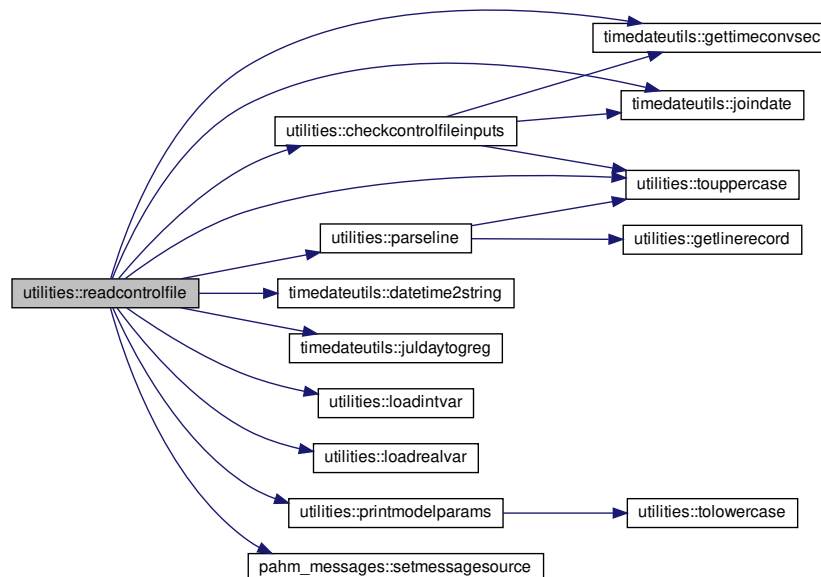
<i>inpFile</i>	The full pathname of the input file
----------------	-------------------------------------

Definition at line 153 of file utilities.F90.

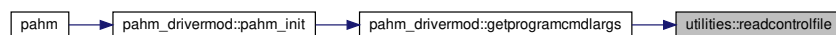
References [pahm_sizes::blank](#), [checkcontrolfileinputs\(\)](#), [closetol](#), [timedateutils::datetime2string\(\)](#), [timedateutils::gettimeconvsec\(\)](#), [pahm_sizes::imissv](#), [timedateutils::joindate\(\)](#), [timedateutils::juldaytogreg\(\)](#), [loadintvar\(\)](#), [loadrealvar\(\)](#), [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

<i>String</i>	The input string
<i>Pos</i>	The position in the input string where the scanning begins
<i>Value</i>	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

<i>lats</i>	Latitude of first points - real, 1D array
<i>lons</i>	Longitude of first points - real, 1D array
<i>lat0</i>	Latitude of second point - real, scalar
<i>lon0</i>	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

<i>lats</i>	Latitude of first points - real, 2D array
<i>lons</i>	Longitude of first points - real, 2D array
<i>lat0</i>	Latitude of second point - real, scalar
<i>lon0</i>	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

<i>lat1</i>	Latitude of first point - real, scalar
<i>lon1</i>	Longitude of first point - real, scalar
<i>lat2</i>	Latitude of second point - real, scalar
<i>lon2</i>	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

<i>lat1</i>	Latitude of first point - real, scalar
<i>lon1</i>	Longitude of first point - real, scalar
<i>lat2</i>	Latitude of second point - real, scalar
<i>lon2</i>	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

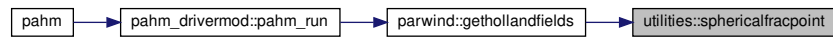
<i>lat1</i>	Latitude of the first point (degrees north)
<i>lon1</i>	Longitude of the first point (degrees east)
<i>lat2</i>	Latitude of the second point (degrees north)
<i>lon2</i>	Longitude of the second point (degrees east)
<i>fraction</i>	The fraction of the distance between points 1 and 2 where the intermediate point is located ($0 \leq \text{fraction} \leq 1$)
<i>latf</i>	The calculated latitude of the intermediate point (degrees north, output)
<i>lonf</i>	The calculated longitude of the intermediate point (degrees east, output)
<i>distf</i>	The great circle distance between the first and the intermediate point (m, output)
<i>dist12</i>	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

<i>inpString</i>	The input string
------------------	------------------

Returns

outString: The output 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

<i>inpString</i>	The input string
------------------	------------------

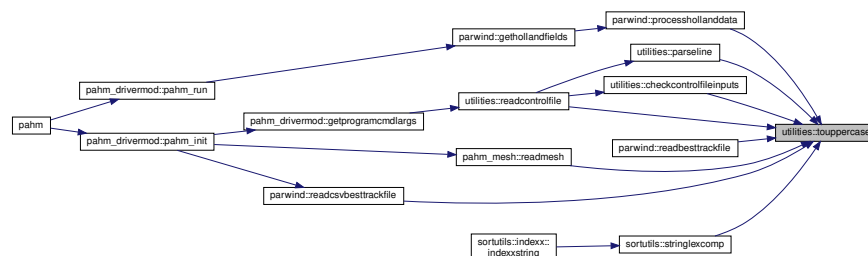
Returns

outString: The output 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

<i>String</i>	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

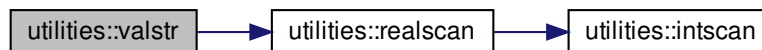
See also

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`

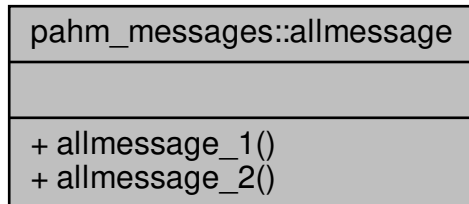
Definition at line 24 of file [utilities.F90](#).

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 )

```

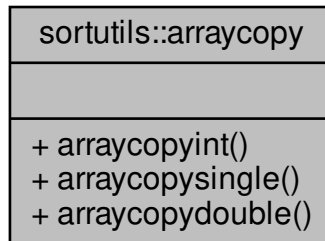
Definition at line 321 of file [messages.F90](#).

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

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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

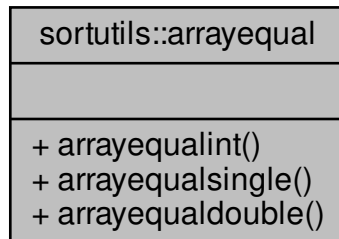
Definition at line 1208 of file [sortutils.F90](#).

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

<i>arr1</i>	The first array in the comparison (double precision)
<i>arr2</i>	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

<i>arr1</i>	The first array in the comparison (integer)
<i>arr2</i>	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

<i>arr1</i>	The first array in the comparison (single precision)
<i>arr2</i>	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.

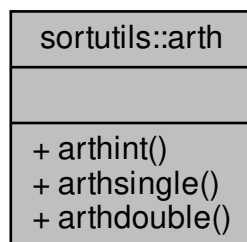
Definition at line 1329 of file [sortutils.F90](#).

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

<i>first</i>	The value of the first term (double precision)
<i>increment</i>	The value of the increment (double precision)
<i>n</i>	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

<i>first</i>	The value of the first term (integer)
<i>increment</i>	The value of the increment (integer)
<i>n</i>	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

<i>first</i>	The value of the first term (single precision)
<i>increment</i>	The value of the increment (single precision)
<i>n</i>	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 [intl](#)
- 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 cynam `integer, dimension(:), allocatable parwind::besttrackdata_t::cynam`

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`

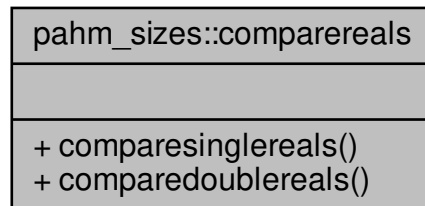
Definition at line 104 of file [parwind-orig.F90](#).

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

<i>rVal1</i>	The first value (double precision number) in the comparison
<i>rVal2</i>	The second value (double precision number) in the comparison
<i>eps</i>	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

<i>rVal1</i>	The first value (single precision number) in the comparison
<i>rVal2</i>	The second value (single precision number) in the comparison
<i>eps</i>	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)

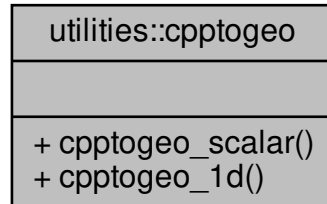
Definition at line 168 of file [sizes.F90](#).

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

<i>x</i>	X coordinate: x (m) - real, 1D array
<i>y</i>	Y coordinate: y (m) - real, 1D array
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>lat</i>	Latitude (degrees north) - real, 1D array (output)
<i>lon</i>	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

<i>x</i>	X coordinate: x (m) - real, scalar
<i>y</i>	Y coordinate: y (m) - real, scalar
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>lat</i>	Latitude (degrees north) - real, scalar (output)
<i>lon</i>	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:

csv_module::csv_file
<ul style="list-style-type: none"> + quote + quotation + character + delimiter + n_rows + n_cols + chunk_size + header + csv_data + icol + iunit + enclose_strings_in_quotes + enclose_all_in_quotes + logical_true_string + logical_false_string
<ul style="list-style-type: none"> + initialize() + read() + destroy() + variable_types() + get_header() + get_header_str() + get_header_csv_str() + get() + get_csv_data_as_str() + csv_get_value() and 15 more...

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.

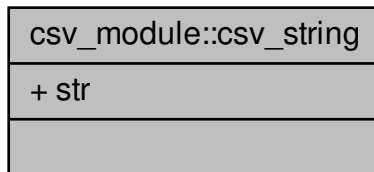
Definition at line 55 of file `csv_module.F90`.

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.

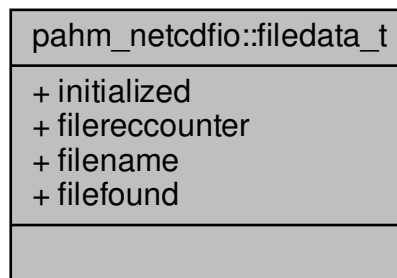
Definition at line 42 of file [csv_module.F90](#).

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.`

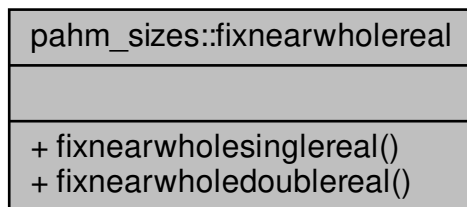
Definition at line 32 of file [netcdfio-nems.F90](#).

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

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

Returns

`myValOut` : Either **rVal** or its nearest integer **iVal** 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

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

Returns

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

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

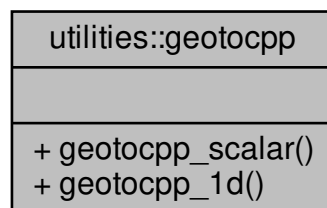
Definition at line 291 of file [sizes.F90](#).

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

<i>lat</i>	Latitude (degrees north) - real, 1D array
<i>lon</i>	Longitude (degrees east) - real, 1D array
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>x</i>	Calculated X coordinate: x (m) - real, 1D array (output)
<i>y</i>	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

<i>lat</i>	Latitude (degrees north) - real, scalar
<i>lon</i>	Longitude (degrees east) - real, scalar
<i>lat0</i>	Latitude of projection origin (degrees north) - real, scalar
<i>lon0</i>	Longitude of projection origin (degrees east) - real, scalar
<i>x</i>	Calculated X coordinate: x (m) - real, scalar (output)
<i>y</i>	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:

timedateutils::gregtojulday
+ gregtojuldayisec() + gregtojuldayrsec() + gregtojulday2()

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.

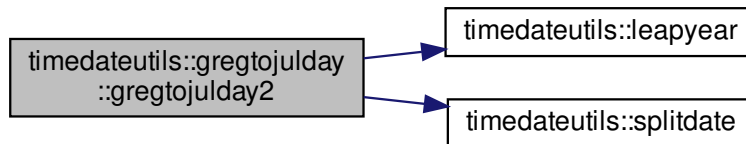
Note

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

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

Parameters

<i>mJD</i>	<p>Flag to use a modified julian day number or not</p> <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>
------------	---

Returns

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

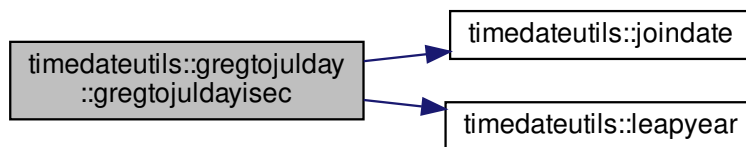
Note

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 useful to compute differences between dates.

Similar to GregToJulDay|SEC but the seconds number is real to allow for second fractions.

Parameters

<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY)
<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12)
<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31)
<i>iHour</i>	The hour of the day (hh, integer, 0 <= hh <= 23)
<i>iMin</i>	The minute of the hour (mm, integer, 0 <= mm <= 59)
<i>rSec</i>	The second of the minute (ss, real, 0 <= ss <= 59)
<i>mJD</i>	Flag to use a modified julian day number or not To use a modified julian day number use: <code>mJD >= 1</code> otherwise use: <code>mJD < 1</code> default: <code>mJD = 0</code> The modified julian day number (MJD) was defined in the mid 1950's in the interests of astronomy and space science as <code>MJD = JD - 2400000.5</code> . 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

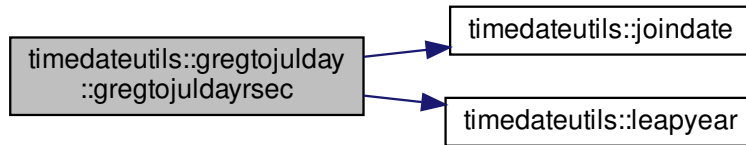
Note

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:

- [timdateutils.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`

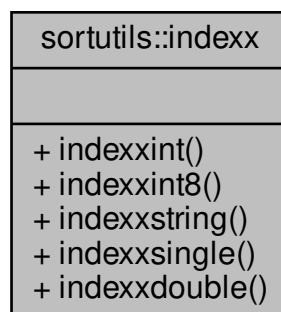
Definition at line 124 of file [parwind-orig.F90](#).

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

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

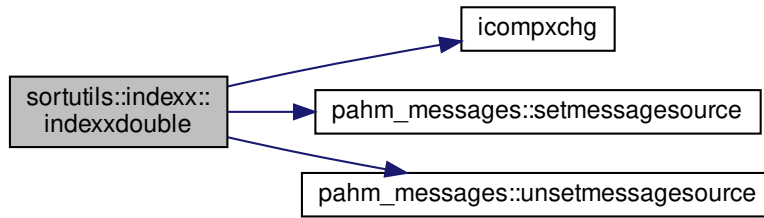
Note

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

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

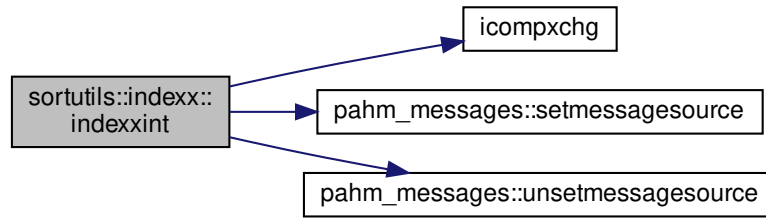
Note

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

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

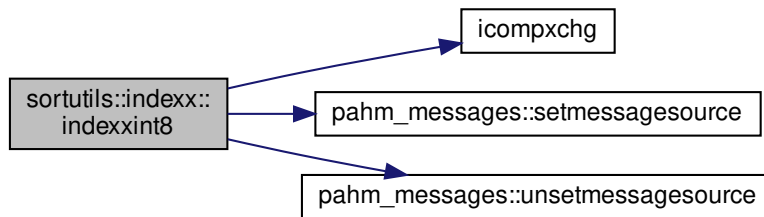
Note

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

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

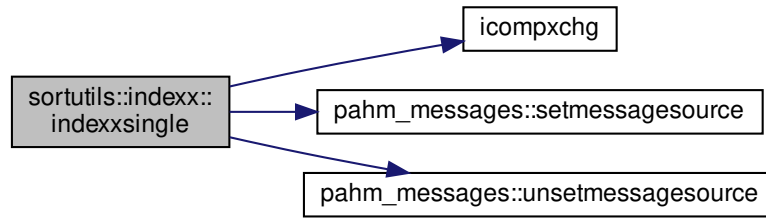
Note

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

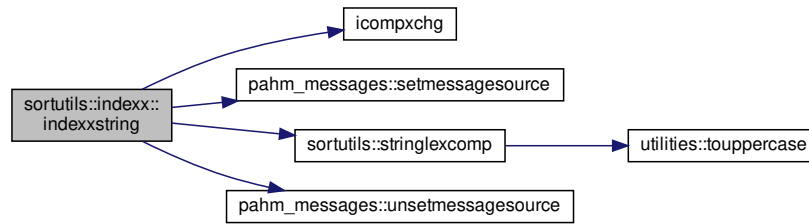
Parameters

<i>arr1D</i>	The array to be indexed (string)
<i>idx1D</i>	The array of "indexed" indexes of <i>arr1D</i> (output)
<i>status</i>	The error status, no error: <i>status</i> = 0 (output)
<i>caseSens</i>	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:

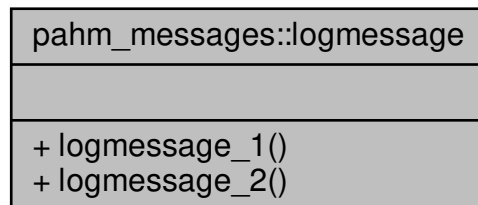


The documentation for this interface was generated from the following file:

- [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 UnsetMessageSource 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](#).

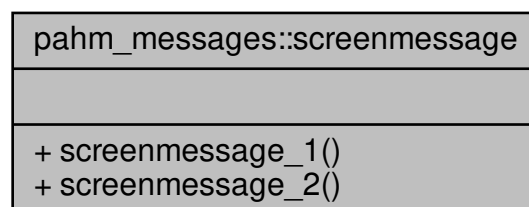
References [pahm_messages::logfileopened](#), [pahm_messages::loginitcalled](#), [pahm_messages::loglevelnames](#), [pahm_global::lun_log](#), [pahm_messages::messagesources](#), and [pahm_messages::sourcenumber](#).

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 UnsetMessageSource 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](#).

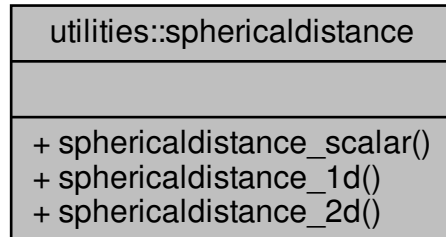
References [pahm_messages::loginitcalled](#), [pahm_messages::loglevelnames](#), [pahm_global::lun_screen](#), [pahm_messages::messagesources](#), [pahm_messages::nscreen](#), and [pahm_messages::sourcenumber](#).

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_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

<i>lats</i>	Latitude of first points - real, 1D array
<i>lons</i>	Longitude of first points - real, 1D array
<i>lat0</i>	Latitude of second point - real, scalar
<i>lon0</i>	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

<i>lats</i>	Latitude of first points - real, 2D array
<i>lons</i>	Longitude of first points - real, 2D array
<i>lat0</i>	Latitude of second point - real, scalar
<i>lon0</i>	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

<i>lat1</i>	Latitude of first point - real, scalar
<i>lon1</i>	Longitude of first point - real, scalar
<i>lat2</i>	Latitude of second point - real, scalar
<i>lon2</i>	Longitude of second point - real, scalar

Returns

myValOut: The great-circle distance in meters

Definition at line 1993 of file [utilities.F90](#).

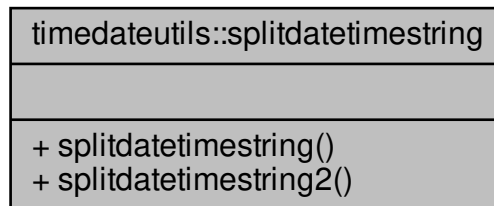
References [pahm_global::deg2rad](#), and [pahm_global::rearth](#).

The documentation for this interface was generated from the following file:

- [utilities.F90](#)

8.19 timdateutils::splitdatetimestring Interface Reference

Collaboration diagram for timdateutils::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 [timdateutils.F90](#).

8.19.2 Constructor & Destructor Documentation

8.19.2.1 splitdatetimestring() subroutine `timdateutils::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)".

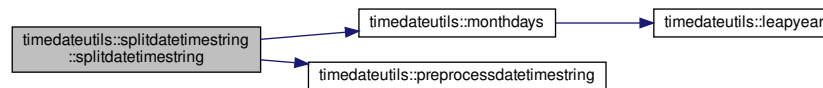
Parameters

<i>inDateTime</i>	The input date string: YYYYMMDDhhmmss
<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY, output)
<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12, output)
<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31, output)
<i>iHour</i>	The hour of the day (hh, integer, 0 <= hh <= 23, output)
<i>iMin</i>	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
<i>iSec</i>	The second of the minute (ss, integer, 0 <= ss <= 59, output)

Definition at line 1073 of file [timdateutils.F90](#).

References [timdateutils::monthdays\(\)](#), and [timdateutils::preprocessdatetimestring\(\)](#).

Here is the call graph for this function:



8.19.3 Member Function/Subroutine Documentation

8.19.3.1 splitdatetimestring2() subroutine `timdateutils::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)".

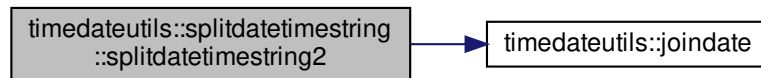
Parameters

<i>inDateTime</i>	The input date string: YYYYMMDDhhmmss
<i>iDate</i>	The integer date (YYYYMMDD, output)
<i>iTime</i>	The integer time (hhmmss, output)

Definition at line 1141 of file [timedateutils.F90](#).

References [timedateutils::joindate\(\)](#).

Here is the call graph for this function:

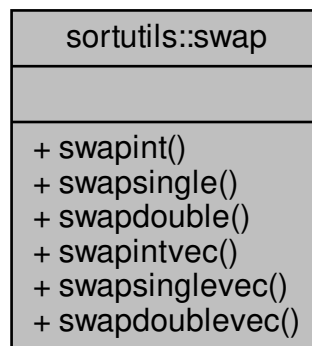


The documentation for this interface was generated from the following file:

- [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

<i>a</i>	The first value to be swapped (double precision)
<i>b</i>	The second value to be swapped (double precision)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (double precision)
<i>b</i>	The second 1D array to be swapped (double precision)
<i>mask</i>	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

<i>a</i>	The first value to be swapped (integer)
<i>b</i>	The second value to be swapped (integer)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (integer)
<i>b</i>	The second 1D array to be swapped (integer)
<i>mask</i>	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

<i>a</i>	The first value to be swapped (single precision)
<i>b</i>	The second value to be swapped (single precision)
<i>mask</i>	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

<i>a</i>	The first 1D array to be swapped (single precision)
<i>b</i>	The second 1D array to be swapped (single precision)
<i>mask</i>	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

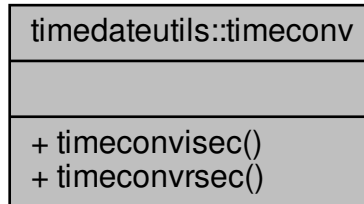
Definition at line [1717](#) of file [sortutils.F90](#).

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.

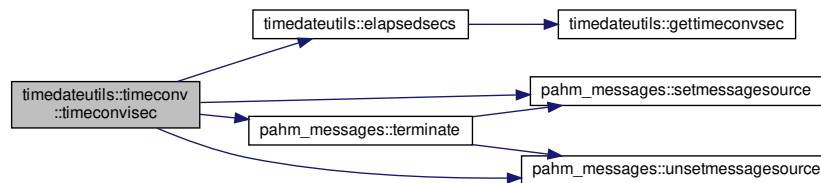
Parameters

<i>iYear</i>	The year (integer)
<i>iMonth</i>	The month of the year (1-12, integer)
<i>iDay</i>	The day of the month (1-31, integer)
<i>iHour</i>	The hour of the day (0-23, integer)
<i>iMin</i>	The minute of the hour (0-59, integer)
<i>iSec</i>	The second of the minute (0-59, integer)
<i>timeSec</i>	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.

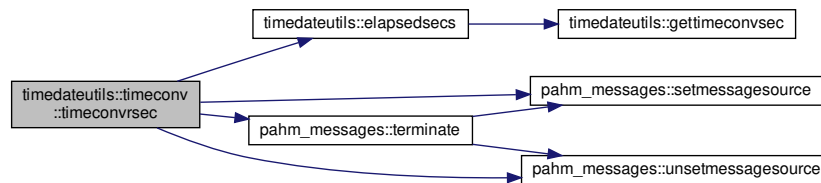
Parameters

<i>iYear</i>	The year (integer)
<i>iMonth</i>	The month of the year (1-12, integer)
<i>iDay</i>	The day of the month (1-31, integer)
<i>iHour</i>	The hour of the day (0-23, integer)
<i>iMin</i>	The minute of the hour (0-59, integer)
<i>rSec</i>	The second of the minute (0-59, real)
<i>timeSec</i>	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:

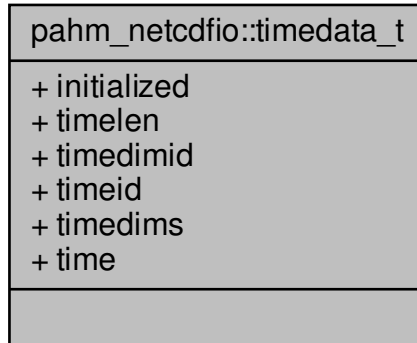


The documentation for this interface was generated from the following file:

- `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.](#)

```

00001 !-----
00002 !           M O D U L E   C S V _ M O D U L E
00003 !-----
00014 !-----
00015
00016 module csv_module
00017
00018   USE pahm_sizes, ONLY : wp, ip
00019   USE pahm_global, ONLY : lun_btrk, lun_btrk1
00020   USE pahm_messages
00021   USE utilities, ONLY : openfileforread, tolowercase
00022   use csv_utilities
00023   use csv_parameters
00024
00025   implicit none
00026
00027   private
00028
00029   ! the different types of variables that can be in a CSV file.
00030   integer,parameter,public :: csv_type_string = 1
00031   integer,parameter,public :: csv_type_double = 2
00032   integer,parameter,public :: csv_type_integer = 3
00033   integer,parameter,public :: csv_type_logical = 4
00034
00035   real(wp),parameter :: zero = 0.0_wp
00036
00037   type,public :: csv_string
00042     character(len=:),allocatable :: str
00043 end type csv_string
00044
00045   type,public :: csv_file
00046
00052
00053   private
00054
00055   character(len=1) :: quote = '"'
00056   character(len=1) :: delimiter = ','
00057
00058   ! for reading a csv file:
00059   integer,public :: n_rows = 0 ! number of rows in the file
00060   integer,public :: n_cols = 0 ! number of columns in the file
00061   integer :: chunk_size = 100 ! for expanding vectors
00062   type(csv_string),dimension(:),allocatable :: header ! the header
00063   type(csv_string),dimension(:,),allocatable :: csv_data ! the data in the file
00064
00065   ! for writing a csv file:
00066   integer :: icol = 0 ! last column written in current row
00067   integer :: iunit = lun_btrk ! file unit for writing
00068   logical :: enclose_strings_in_quotes = .true. ! if true, all string cells
00069                                           ! will be enclosed in quotes.
00070   logical :: enclose_all_in_quotes = .false. ! if true, *all* cells will
00071                                           ! be enclosed in quotes.
00072   character(len=1) :: logical_true_string = 'T' ! when writing a logical 'true'
00073                                           ! value to a CSV file, this
00074                                           ! is the string to use
00075                                           ! (default is 'T')
00076   character(len=1) :: logical_false_string = 'F' ! when writing a logical 'false'
00077                                           ! value to a CSV file, this
00078                                           ! is the string to use
00079                                           ! (default is 'F')
00080
00081 contains
00082
00083   private
00084
00085   procedure,public :: initialize => initialize_csv_file
00086   procedure,public :: read => read_csv_file
00087   procedure,public :: destroy => destroy_csv_file
00088
00089   procedure,public :: variable_types
00090
00091   generic,public :: get_header => get_header_str,&
00092                                           get_header_csv_str
00093   procedure :: get_header_str
00094   procedure :: get_header_csv_str
00095

```

```

00097      ! For getting data from the class
00098      ! after the file has been read.
00099      generic,public :: get => get_csv_data_as_str,&
00100                          csv_get_value,&
00101                          get_real_column,&
00102                          get_integer_column,&
00103                          get_logical_column,&
00104                          get_character_column,&
00105                          get_csv_string_column
00106      procedure :: get_csv_data_as_str
00107      procedure :: csv_get_value
00108      procedure :: get_real_column
00109      procedure :: get_integer_column
00110      procedure :: get_logical_column
00111      procedure :: get_character_column
00112      procedure :: get_csv_string_column
00113
00114      procedure,public :: open => open_csv_file
00115
00116      generic,public :: add => add_cell,&
00117                          add_vector,&
00118                          add_matrix
00119      procedure :: add_cell
00120      procedure :: add_vector
00121      procedure :: add_matrix
00122
00123      procedure,public :: next_row
00124      procedure,public :: close => close_csv_file
00125
00126      procedure :: tokenize => tokenize_csv_line
00127      procedure :: read_line_from_file
00128      procedure :: get_column
00129
00130      end type csv_file
00131
00132      contains
00133      !*****
00134
00135      !-----
00136      ! SUBROUTINE EXPAND_VECTOR
00137      !-----
00138      !-----
00139
00140      subroutine initialize_csv_file(me,quote,delimiter,&
00141                                   enclose_strings_in_quotes,&
00142                                   enclose_all_in_quotes,&
00143                                   logical_true_string,&
00144                                   logical_false_string,&
00145                                   chunk_size)
00146
00147      implicit none
00148
00149      class(csv_file),intent(out) :: me
00150      character(len=1),intent(in),optional :: quote          ! note: can only be one character
00151                                                             ! (Default is '"')
00152      character(len=1),intent(in),optional :: delimiter      ! note: can only be one character
00153                                                             ! (Default is ',')
00154      logical,intent(in),optional :: enclose_strings_in_quotes ! if true, all string cells
00155                                                             ! will be enclosed in quotes.
00156                                                             ! (Default is True)
00157      logical,intent(in),optional :: enclose_all_in_quotes    ! if true, *all* cells will
00158                                                             ! be enclosed in quotes.
00159                                                             ! (Default is False)
00160      character(len=1),intent(in),optional :: logical_true_string ! when writing a logical 'true'
00161                                                             ! value to a CSV file, this
00162                                                             ! is the string to use
00163                                                             ! (default is 'T')
00164      character(len=1),intent(in),optional :: logical_false_string ! when writing a logical 'false'
00165                                                             ! value to a CSV file, this
00166                                                             ! is the string to use
00167                                                             ! (default is 'F')
00168      integer,intent(in),optional :: chunk_size ! factor for expanding vectors
00169                                               ! (default is 100)
00170
00171      if (present(quote)) me%quote = quote
00172      if (present(delimiter)) me%delimiter = delimiter
00173      if (present(enclose_strings_in_quotes)) &
00174          me%enclose_strings_in_quotes = enclose_strings_in_quotes
00175      if (present(enclose_all_in_quotes)) &
00176          me%enclose_all_in_quotes = enclose_all_in_quotes
00177      if (present(logical_true_string)) &
00178          me%logical_true_string = logical_true_string

```

```

00205     if (present(logical_false_string)) &
00206         me%logical_false_string = logical_false_string
00207     if (present(chunk_size)) me%chunk_size = chunk_size
00208
00209     ! override:
00210     if (me%enclose_all_in_quotes) me%enclose_strings_in_quotes = .true.
00211
00212     end subroutine initialize_csv_file
00213 !*****
00214 !*****
00215 !*****
00216 !
00217
00218     subroutine destroy_csv_file(me)
00219
00220     implicit none
00221
00222     class(csv_file), intent(out) :: me
00223
00224     end subroutine destroy_csv_file
00225 !*****
00226 !*****
00227 !*****
00228 !*****
00229 !
00230
00231     subroutine read_csv_file(me, filename, header_row, skip_rows, status_ok)
00232
00233     implicit none
00234
00235     class(csv_file), intent(inout) :: me
00236     character(len=*) , intent(in) :: filename
00237     logical, intent(out) :: status_ok
00238     integer, intent(in), optional :: header_row
00239     integer, dimension(:), intent(in), optional :: skip_rows
00240
00241     type(csv_string), dimension(:), allocatable :: row_data
00242     type(csv_string) :: empty_data
00243     integer, dimension(:), allocatable :: rows_to_skip
00244     character(len=:), allocatable :: line
00245     integer :: i
00246     integer :: j
00247     integer :: irow
00248     integer :: n_rows_in_file
00249     integer :: n_rows
00250     integer :: n_cols
00251     integer :: istat
00252     integer :: line_n_cols
00253     integer :: iunit
00254     logical :: arrays_allocated
00255     integer :: iheader
00256     character(len=1) :: tmp
00257
00258     empty_data%str = ' '
00259     iunit = lun_btrk
00260
00261     CALL setmessagesource("read_csv_file")
00262
00263     call me%destroy()
00264     arrays_allocated = .false.
00265
00266     CALL openfileforread(iunit, trim(adjustl(filename)), istat)
00267
00268     IF (istat /= 0) THEN
00269         WRITE(scratchmessage, '(a)') 'Error opening the file: ' // trim(adjustl(filename))
00270         CALL allmessage(error, scratchmessage)
00271
00272         CALL unsetmessagesource()
00273
00274         CALL terminate()
00275     ELSE
00276         WRITE(scratchmessage, '(a)') 'Processing the file: ' // trim(adjustl(filename))
00277         CALL logmessage(info, scratchmessage)
00278     END IF
00279
00280     ! if (istat==0) then
00281
00282         !get number of lines in the file
00283         n_rows_in_file = number_of_lines_in_file(iunit)
00284
00285         !get number of lines in the data array
00286         if (present(skip_rows)) then

```

```

00290         !get size of unique elements in skip_rows,
00291         !and subtract from n_rows_in_file
00292         rows_to_skip = unique(skip_rows,chunk_size=me%chunk_size)
00293         n_rows = n_rows_in_file - size(rows_to_skip)
00294     else
00295         n_rows = n_rows_in_file
00296     end if
00297     if (present(header_row)) then
00298         iheader = max(0,header_row)
00299         n_rows = n_rows - 1
00300     else
00301         iheader = 0
00302     end if
00303
00304     me%n_rows = n_rows
00305
00306     ! we don't know the number of columns
00307     ! until we parse the first row (or the header)
00308     ! Panagiotis Velissariou: some csv files do not have the same number
00309     ! of columns, so we need to determine the max number of columns
00310     ! for the allocation of the arrays
00311     !--- PV
00312     n_cols = 0
00313     do i=1,n_rows_in_file
00314         call me%read_line_from_file(iunit,line,status_ok)
00315         call me%tokenize(line,row_data)
00316         n_cols = max(n_cols,size(row_data))
00317     end do
00318     rewind(iunit)
00319
00320     me%n_cols = n_cols
00321     allocate(me%csv_data(n_rows,n_cols))
00322     if (iheader/=0) allocate(me%header(n_cols))
00323     arrays_allocated = .true.
00324     !--- PV
00325
00326     !read each line in the file, parse it, and populate data
00327     irow = 0
00328     do i=1,n_rows_in_file
00329
00330         ! skip row if necessary
00331         if (allocated(rows_to_skip)) then
00332             if (any(i==rows_to_skip)) then
00333                 read(iunit,fmt='(A1)',iostat=istat) tmp
00334                 if (istat/=0) then
00335                     scratchmessage = 'Error skipping row in file: '//trim(filename)
00336                     CALL allmessage(error, scratchmessage)
00337
00338                     close(unit=iunit,iostat=istat)
00339                     status_ok = .false.
00340
00341                     CALL unsetmessagesource()
00342                     return
00343                 end if
00344             cycle
00345         end if
00346     end if
00347
00348     call me%read_line_from_file(iunit,line,status_ok)
00349     if (.not. status_ok) then
00350         CALL unsetmessagesource()
00351         return ! file read error
00352     end if
00353     call me%tokenize(line,row_data)
00354     line_n_cols = size(row_data)
00355
00356     if (i==iheader) then
00357         do j=1,me%n_cols
00358             me%header(j)%str = row_data(j)%str
00359         end do
00360     else
00361         irow = irow + 1
00362         do j=1,n_cols
00363             if (j <= line_n_cols) then
00364                 me%csv_data(irow,j) = row_data(j) !%str
00365             else
00366                 me%csv_data(irow,j) = empty_data !%str
00367             end if
00368         end do
00369     end if
00370

```

```

00371         end do
00372
00373         ! close the file
00374         close(unit=iunit,iostat=istat)
00375
00376         status_ok = .true.
00377
00378 !     else
00379 !         scratchMessage = 'Error opening file: '//trim(filename)
00380 !         CALL AllMessage(ERROR, scratchMessage)
00381 !         status_ok = .false.
00382 !     end if
00383
00384     CALL unsetmessagesource()
00385
00386 end subroutine read_csv_file
00387 !*****
00388
00389 !*****
00390 !
00391
00392     subroutine open_csv_file(me,filename,n_cols,status_ok,append)
00393
00394     implicit none
00395
00396     class(csv_file),intent(inout)    :: me
00397     character(len=*) ,intent(in)      :: filename
00398     integer,intent(in)                :: n_cols
00399     logical,intent(out)               :: status_ok
00400     logical,intent(in),optional       :: append
00401
00402     integer :: istat
00403     logical :: append_flag
00404     logical :: file_exists
00405
00406     CALL setmessagesource("open_csv_file")
00407
00408     call me%destroy()
00409
00410     me%n_cols = n_cols
00411
00412     ! optional append argument:
00413     append_flag = .false.
00414     file_exists = .false.
00415     if (present(append)) then
00416         append_flag = append
00417         if (append) inquire(file=filename, exist=file_exists)
00418     end if
00419
00420     if (append_flag .and. file_exists) then
00421         open(unit=me%iunit,file=filename,status='OLD',position='APPEND',iostat=istat)
00422     else
00423         open(unit=me%iunit,file=filename,status='REPLACE',iostat=istat)
00424     end if
00425
00426     if (istat==0) then
00427         status_ok = .true.
00428     else
00429         scratchmessage = 'Error opening file: '//trim(filename)
00430         CALL allmessage(error, scratchmessage)
00431         status_ok = .false.
00432     end if
00433
00434     CALL unsetmessagesource()
00435
00436 end subroutine open_csv_file
00437 !*****
00438
00439 !*****
00440 !
00441
00442     subroutine close_csv_file(me,status_ok)
00443
00444     implicit none
00445
00446     class(csv_file),intent(inout) :: me
00447     logical,intent(out) :: status_ok
00448
00449     integer :: istat
00450
00451     close(me%iunit,iostat=istat)

```

```

00456     status_ok = istat==0
00457
00458     end subroutine close_csv_file
00459 !*****
00460
00461 !*****
00462 !
00466
00467     subroutine add_cell(me,val,int_fmt,real_fmt,trim_str)
00468
00469     implicit none
00470
00471     class(csv_file),intent(inout) :: me
00472     class(*),intent(in) :: val
00473     character(len=*),intent(in),optional :: int_fmt
00475     character(len=*),intent(in),optional :: real_fmt
00477     logical,intent(in),optional :: trim_str
00478
00479     integer :: istat
00480     character(len=:),allocatable :: ifmt
00481     character(len=:),allocatable :: rfmt
00482     logical :: trimstr
00483     character(len=max_real_str_len) :: real_val
00484     character(len=max_integer_str_len) :: int_val
00485
00486     CALL setmessagesource("add_cell")
00487
00488     ! make sure the row isn't already finished
00489     if (me%icol<me%n_cols) then
00490
00491         me%icol = me%icol + 1
00492
00493         if (me%enclose_all_in_quotes) then
00494             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00495         end if
00496
00497         select type (val)
00498         type is (integer(ip))
00499             if (present(int_fmt)) then
00500                 ifmt = trim(adjustl(int_fmt))
00501             else
00502                 ifmt = default_int_fmt
00503             end if
00504             write(int_val,fmt=ifmt,iostat=istat) val
00505             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(int_val))
00506         type is (real(wp))
00507             if (present(real_fmt)) then
00508                 rfmt = trim(adjustl(real_fmt))
00509             else
00510                 rfmt = default_real_fmt
00511             end if
00512             write(real_val,fmt=rfmt,iostat=istat) val
00513             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(real_val))
00514         type is (logical)
00515             if (val) then
00516                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_true_string
00517             else
00518                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_false_string
00519             end if
00520         type is (character(len=*))
00521             if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00522                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00523             if (present(trim_str)) then
00524                 trimstr = trim_str
00525             else
00526                 trimstr = .false.
00527             end if
00528             if (trimstr) then
00529                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val)
00530             else
00531                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val
00532             end if
00533             if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00534                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00535         type is (csv_string)
00536             if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00537                 write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00538             if (present(trim_str)) then
00539                 trimstr = trim_str
00540             else
00541                 trimstr = .false.

```

```

00542         end if
00543         if (trimstr) then
00544             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val%str)
00545         else
00546             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val%str
00547         end if
00548         if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00549             write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00550     class default
00551         scratchmessage = 'Error: cannot write unknown variable type to CSV file.'
00552         CALL allmessage(error, scratchmessage)
00553     end select
00554
00555     if (me%enclose_all_in_quotes) then
00556         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00557     end if
00558     if (me%icol<me%n_cols) write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%delimiter
00559
00560 else
00561     scratchmessage = 'Error: cannot write more cells to the current row.'
00562     CALL allmessage(error, scratchmessage)
00563 end if
00564
00565 CALL unsetmessagesource()
00566
00567 end subroutine add_cell
00568 !*****
00569
00570 !*****
00571 !
00572
00573     subroutine add_vector(me,val,int_fmt,real_fmt,trim_str)
00574
00575     implicit none
00576
00577     class(csv_file),intent(inout) :: me
00578     class(*),dimension(:),intent(in) :: val
00579     character(len=*),intent(in),optional :: int_fmt
00580     character(len=*),intent(in),optional :: real_fmt
00581     logical,intent(in),optional :: trim_str
00582
00583     integer :: i
00584
00585     do i=1,size(val)
00586
00587         #if defined __GFORTRAN__
00588             ! This is a stupid workaround for gfortran bugs (tested with 7.2.0)
00589             select type (val)
00590             type is (character(len=*))
00591                 call me%add(val(i),int_fmt,real_fmt,trim_str)
00592             class default
00593                 call me%add(val(i),int_fmt,real_fmt,trim_str)
00594             end select
00595         #else
00596             call me%add(val(i),int_fmt,real_fmt,trim_str)
00597         #endif
00598     end do
00599
00600 end subroutine add_vector
00601 !*****
00602 !
00603
00604     subroutine add_matrix(me,val,int_fmt,real_fmt,trim_str)
00605
00606     implicit none
00607
00608     class(csv_file),intent(inout) :: me
00609     class(*),dimension(:,,:),intent(in) :: val
00610     character(len=*),intent(in),optional :: int_fmt
00611     character(len=*),intent(in),optional :: real_fmt
00612     logical,intent(in),optional :: trim_str
00613
00614     integer :: i
00615
00616     ! add each row:
00617     do i=1,size(val,1)
00618         call me%add(val(i,:),int_fmt,real_fmt,trim_str)
00619     end do
00620
00621     call me%next_row()
00622
00623 end

```

```

00631     end do
00632
00633     end subroutine add_matrix
00634 !*****
00635
00636 !*****
00637 !
00640
00641     subroutine next_row(me)
00642
00643     implicit none
00644
00645     class(csv_file),intent(inout) :: me
00646
00647     integer :: i
00648     integer :: n
00649
00650     if (me%icol>0) then
00651         n = me%n_cols - me%icol
00652         do i=1,n
00653             if (i==n) then !no trailing delimiter
00654                 if (me%enclose_strings_in_quotes) then
00655                     write(me%iunit,'(A)',advance='NO') me%quote//me%quote
00656                 end if
00657             else
00658                 if (me%enclose_strings_in_quotes) then
00659                     write(me%iunit,'(A)',advance='NO') me%quote//me%quote//me%delimiter
00660                 else
00661                     write(me%iunit,'(A)',advance='NO') me%delimiter
00662                 end if
00663             end if
00664         end do
00665         write(me%iunit,'(A)') " ! new line
00666     end if
00667
00668     me%icol = 0 ! this row is finished
00669
00670     end subroutine next_row
00671 !*****
00672
00673 !*****
00674 !
00677
00678     subroutine get_header_csv_str(me,header,status_ok)
00679
00680     implicit none
00681
00682     class(csv_file),intent(inout) :: me
00683     type(csv_string),dimension(:),allocatable,intent(out) :: header
00684     logical,intent(out) :: status_ok
00685
00686     integer :: i
00687
00688     CALL setmessagesource("get_header_csv_str")
00689
00690     if (allocated(me%header)) then
00691
00692         allocate(header(me%n_cols))
00693         do i=1,me%n_cols
00694             header(i) = me%header(i)
00695         end do
00696         status_ok = .false.
00697
00698     else
00699         scratchmessage = 'Error: no header in class.'
00700         CALL allmessage(error, scratchmessage)
00701         status_ok = .false.
00702     end if
00703
00704     CALL unsetmessagesource()
00705
00706     end subroutine get_header_csv_str
00707 !*****
00708
00709 !*****
00710 !
00713
00714     subroutine get_header_str(me,header,status_ok)
00715
00716     implicit none
00717

```



```

00718     class(csv_file),intent(inout) :: me
00719     character(len=*),dimension(:),allocatable,intent(out) :: header
00720     logical,intent(out) :: status_ok
00721
00722     integer :: i
00723
00724     CALL setmessagesource("get_header_str")
00725
00726     if (allocated(me%header)) then
00727         allocate(header(me%n_cols))
00728         do i=1,me%n_cols
00729             header(i) = me%header(i)%str
00730         end do
00731         status_ok = .false.
00732     else
00733         scratchmessage = 'Error: no header in class.'
00734         CALL allmessage(error, scratchmessage)
00735         status_ok = .false.
00736     end if
00737
00738     CALL unsetmessagesource()
00739
00740     end subroutine get_header_str
00741 !*****
00742 !*****
00743 !*****
00744 !*****
00745 !*****
00746 !*****
00747 !*****
00748 !*****
00749
00750     subroutine get_csv_data_as_str(me, csv_data, status_ok)
00751
00752     implicit none
00753
00754     class(csv_file),intent(inout) :: me
00755     character(len=*),dimension(:,:),allocatable,intent(out) :: csv_data
00756     logical,intent(out) :: status_ok
00757
00758     integer :: i
00759     integer :: j
00760
00761     CALL setmessagesource("get_csv_data_as_str")
00762
00763     if (allocated(me%csv_data)) then
00764         ! size the output array:
00765         allocate(csv_data(me%n_rows,me%n_cols))
00766         ! convert each element to a string:
00767         do concurrent(i=1:me%n_rows)
00768             do concurrent(j=1:me%n_cols)
00769                 csv_data(i,j) = me%csv_data(i,j)%str
00770             end do
00771         end do
00772         status_ok = .true.
00773     else
00774         scratchmessage = 'Error: class has not been initialized'
00775         CALL allmessage(error, scratchmessage)
00776         status_ok = .false.
00777     end if
00778
00779     CALL unsetmessagesource()
00780
00781     end subroutine get_csv_data_as_str
00782 !*****
00783 !*****
00784 !*****
00785 !*****
00786 !*****
00787
00788     pure elemental subroutine to_real(str, val, status_ok)
00789
00790     implicit none
00791
00792     character(len=*),intent(in) :: str
00793     real(wp),intent(out) :: val
00794     logical,intent(out) :: status_ok
00795
00796     integer :: istat
00797
00798     read(str,fmt=*,iostat=istat) val
00799     if (istat==0) then
00800         status_ok = .true.
00801     else

```

```

00802         status_ok = .false.
00803         val = zero
00804     end if
00805
00806     end subroutine to_real
00807 !*****
00808 !*****
00809 !*****
00810 !
00812
00813     pure elemental subroutine to_integer(str,val,status_ok)
00814
00815     implicit none
00816
00817     character(len=*),intent(in) :: str
00818     integer(ip),intent(out) :: val
00819     logical,intent(out) :: status_ok
00820
00821     integer :: istat
00822
00823     read(str,fmt=default_int_fmt,iostat=istat) val
00824     if (istat==0) then
00825         status_ok = .true.
00826     else
00827         status_ok = .false.
00828         val = 0
00829     end if
00830
00831     end subroutine to_integer
00832 !*****
00833 !*****
00834 !*****
00835 !
00842
00843     pure elemental subroutine to_logical(str,val,status_ok)
00844
00845     implicit none
00846
00847     character(len=*),intent(in) :: str
00848     logical,intent(out) :: val
00849     logical,intent(out) :: status_ok
00850
00851     character(len=:),allocatable :: tmp
00852
00853     ! True and False options (all lowercase):
00854     character(len=*),dimension(4),parameter :: true_str = ['1', '&',
00855                                                             't', '&',
00856                                                             'true', '&',
00857                                                             '.true.'],
00858     character(len=*),dimension(4),parameter :: false_str = ['0', '&',
00859                                                             'f', '&',
00860                                                             'false', '&',
00861                                                             '.false.'],
00862
00863     tmp = tolowercase(str)
00864     if ( any(tmp==true_str) ) then
00865         val = .true.
00866         status_ok = .true.
00867     else if ( any(tmp==false_str) ) then
00868         val = .false.
00869         status_ok = .true.
00870     else
00871         val = .false.
00872         status_ok = .false.
00873     end if
00874
00875     end subroutine to_logical
00876 !*****
00877 !*****
00878 !*****
00879 !
00883
00884     subroutine variable_types(me,itypes,status_ok)
00885
00886     implicit none
00887
00888     class(csv_file),intent(inout) :: me
00889     integer,dimension(:),allocatable,intent(out) :: itypes
00890     logical,intent(out) :: status_ok
00891
00892     integer :: i

```

```

00893
00894     CALL setmessagesource("variable_types")
00895
00896     if (allocated(me%csv_data)) then
00897         allocate(itypes(me%n_cols))
00898         do i=1,me%n_cols
00899             call infer_variable_type(me%csv_data(1,i)%str, itypes(i))
00900         end do
00901     else
00902         scratchmessage = 'Error: class has not been initialized'
00903         CALL allmessage(error, scratchmessage)
00904         status_ok = .false.
00905     end if
00906
00907     CALL unsetmessagesource()
00908
00909     end subroutine variable_types
00910 !*****
00911 !*****
00912 !*****
00913 !
00920
00921     subroutine infer_variable_type(str, itype)
00922
00923     implicit none
00924
00925     character(len=*) ,intent(in) :: str
00926     integer,intent(out) :: itype
00927
00928     real(wp)      :: rval
00929     integer(ip)   :: ival
00930     logical       :: lval
00931     logical       :: status_ok
00932
00933     call to_integer(str, ival, status_ok)
00934     if (status_ok) then
00935         itype = csv_type_integer
00936         return
00937     end if
00938
00939     call to_real(str, rval, status_ok)
00940     if (status_ok) then
00941         itype = csv_type_double
00942         return
00943     end if
00944
00945     call to_logical(str, lval, status_ok)
00946     if (status_ok) then
00947         itype = csv_type_logical
00948         return
00949     end if
00950
00951     ! default is string:
00952     itype = csv_type_string
00953
00954     end subroutine infer_variable_type
00955 !*****
00956 !*****
00957 !*****
00958 !
00963
00964     subroutine csv_get_value(me, row, col, val, status_ok)
00965
00966     implicit none
00967
00968     class(csv_file), intent(inout) :: me
00969     integer, intent(in) :: row
00970     integer, intent(in) :: col
00971     class(*), intent(out) :: val
00972     logical, intent(out) :: status_ok
00973
00974     select type (val)
00975     type is (integer(ip))
00976         call to_integer(me%csv_data(row,col)%str, val, status_ok)
00977     type is (real(wp))
00978         call to_real(me%csv_data(row,col)%str, val, status_ok)
00979     type is (logical)
00980         call to_logical(me%csv_data(row,col)%str, val, status_ok)
00981     type is (character(len=*))
00982         status_ok = .true.
00983         if (allocated(me%csv_data(row,col)%str)) then

```

```

00984         val = me%csv_data(row,col)%str
00985     else
00986         val = "
00987     end if
00988     type is (csv_string)
00989         status_ok = .true.
00990         val = me%csv_data(row,col)
00991     class default
00992         status_ok = .false.
00993     end select
00994
00995     end subroutine csv_get_value
00996 !*****
00997
00998 !*****
00999 !
01006
01007     subroutine get_column(me,icol,r,status_ok)
01008
01009     implicit none
01010
01011     class(csv_file),intent(inout) :: me
01012     integer,intent(in) :: icol
01013     class(*),dimension(:),intent(out) :: r
01016     logical,intent(out) :: status_ok
01017
01018     integer :: i
01019 #if defined __GFORTRAN__
01020     character(len=:),allocatable :: tmp
01021 #endif
01022
01023     CALL setmessagesource("get_column")
01024
01025     ! we know the data is allocated, since that
01026     ! was checked by the calling routines.
01027
01028     if (me%n_cols>=icol .and. icol>0) then
01029
01030         do i=1,me%n_rows ! row loop
01031
01032 #if defined __GFORTRAN__
01033         ! the following is a workaround for gfortran bugs:
01034         select type (r)
01035         type is (character(len=*))
01036             tmp = repeat(' ',len(r)) ! size the string
01037             call me%csv_get_value(i,icol,tmp,status_ok)
01038             r(i) = tmp
01039         class default
01040             call me%csv_get_value(i,icol,r(i),status_ok)
01041         end select
01042     #else
01043         call me%csv_get_value(i,icol,r(i),status_ok)
01044     #endif
01045
01046         if (.not. status_ok) then
01047             select type (r)
01048             ! note: character conversion can never fail, so not
01049             ! checking for that here. also we know it is real,
01050             ! integer, or logical at this point.
01051             type is (integer(ip))
01052                 scratchmessage = 'Error converting string to integer: '//trim(me%csv_data(i,icol)%str)
01053                 CALL allmessage(error, scratchmessage)
01054                 r(i) = 0
01055             type is (real(wp))
01056                 scratchmessage = 'Error converting string to real: '//trim(me%csv_data(i,icol)%str)
01057                 CALL allmessage(error, scratchmessage)
01058                 r(i) = zero
01059             type is (logical)
01060                 scratchmessage = 'Error converting string to logical: '//trim(me%csv_data(i,icol)%str)
01061                 CALL allmessage(error, scratchmessage)
01062                 r(i) = .false.
01063             end select
01064         end if
01065
01066     end do
01067
01068     else
01069         WRITE(scratchmessage,'(A,1X,I5)') 'Error: invalid column number: ', icol
01070         CALL allmessage(error, scratchmessage)
01071         status_ok = .false.
01072     end if

```

```

01073     CALL unsetmessagesource()
01074
01075     end subroutine get_column
01076 !*****
01077
01078 !*****
01079 !
01081
01082     subroutine get_real_column(me,icol,r,status_ok)
01083
01084     implicit none
01085
01086     class(csv_file),intent(inout) :: me
01087     integer,intent(in) :: icol
01088     real(wp),dimension(:),allocatable,intent(out) :: r
01089     logical,intent(out) :: status_ok
01090
01091     CALL setmessagesource("get_real_column")
01092
01093     if (allocated(me%csv_data)) then
01094         allocate(r(me%n_rows)) ! size the output vector
01095         call me%get_column(icol,r,status_ok)
01096     else
01097         scratchmessage = 'Error: class has not been initialized'
01098         CALL allmessage(error, scratchmessage)
01099         status_ok = .false.
01100     end if
01101
01102     CALL unsetmessagesource()
01103
01104     end subroutine get_real_column
01105 !*****
01106
01107 !*****
01108 !
01110
01111     subroutine get_integer_column(me,icol,r,status_ok)
01112
01113     implicit none
01114
01115     class(csv_file),intent(inout) :: me
01116     integer,intent(in) :: icol
01117     integer(ip),dimension(:),allocatable,intent(out) :: r
01118     logical,intent(out) :: status_ok
01119
01120     CALL setmessagesource("get_integer_column")
01121
01122     if (allocated(me%csv_data)) then
01123         allocate(r(me%n_rows)) ! size the output vector
01124         call me%get_column(icol,r,status_ok)
01125     else
01126         scratchmessage = 'Error: class has not been initialized'
01127         CALL allmessage(error, scratchmessage)
01128         status_ok = .false.
01129     end if
01130
01131     CALL unsetmessagesource()
01132
01133     end subroutine get_integer_column
01134 !*****
01135
01136 !*****
01137 !
01139
01140     subroutine get_logical_column(me,icol,r,status_ok)
01141
01142     implicit none
01143
01144     class(csv_file),intent(inout) :: me
01145     integer,intent(in) :: icol
01146     logical,dimension(:),allocatable,intent(out) :: r
01147     logical,intent(out) :: status_ok
01148
01149     CALL setmessagesource("get_logical_column")
01150
01151     if (allocated(me%csv_data)) then
01152         allocate(r(me%n_rows)) ! size the output vector
01153         call me%get_column(icol,r,status_ok)
01154     else
01155         scratchmessage = 'Error: class has not been initialized'
01156         CALL allmessage(error, scratchmessage)

```

```

01157         status_ok = .false.
01158     end if
01159
01160     CALL unsetmessagesource()
01161
01162     end subroutine get_logical_column
01163 !*****
01164 !*****
01165 !*****
01166 !
01167
01168     subroutine get_character_column(me,icol,r,status_ok)
01169
01170     implicit none
01171
01172
01173     class(csv_file),intent(inout) :: me
01174     integer,intent(in) :: icol
01175     character(len=*),dimension(:),allocatable,intent(out) :: r
01176     logical,intent(out) :: status_ok
01177
01178     CALL setmessagesource("get_character_column")
01179
01180     if (allocated(me%csv_data)) then
01181         allocate(r(me%n_rows)) ! size the output vector
01182         call me%get_column(icol,r,status_ok)
01183     else
01184         scratchmessage = 'Error: class has not been initialized'
01185         CALL allmessage(error, scratchmessage)
01186         status_ok = .false.
01187     end if
01188
01189     CALL unsetmessagesource()
01190
01191     end subroutine get_character_column
01192 !*****
01193 !*****
01194 !*****
01195 !
01196
01197     subroutine get_csv_string_column(me,icol,r,status_ok)
01198
01199     implicit none
01200
01201
01202     class(csv_file),intent(inout) :: me
01203     integer,intent(in) :: icol
01204     type(csv_string),dimension(:),allocatable,intent(out) :: r
01205     logical,intent(out) :: status_ok
01206
01207     CALL setmessagesource("get_csv_string_column")
01208
01209     if (allocated(me%csv_data)) then
01210         allocate(r(me%n_rows)) ! size the output vector
01211         call me%get_column(icol,r,status_ok)
01212     else
01213         scratchmessage = 'Error: class has not been initialized'
01214         CALL allmessage(error, scratchmessage)
01215         status_ok = .false.
01216     end if
01217
01218     CALL unsetmessagesource()
01219
01220     end subroutine get_csv_string_column
01221 !*****
01222 !*****
01223 !*****
01224 !
01225
01226     subroutine tokenize_csv_line(me,line,cells)
01227
01228     implicit none
01229
01230
01231     class(csv_file),intent(inout) :: me
01232     character(len=*),intent(in) :: line
01233     type(csv_string),dimension(:),allocatable,intent(out) :: cells
01234
01235     integer :: i
01236     character(len=:),allocatable :: tmp
01237     integer :: n
01238
01239     call split(line,me%delimiter,me%chunk_size,cells)
01240

```

```

01247      ! remove quotes if present:
01248      do i = 1, size(cells)
01249
01250          ! remove whitespace from the string:
01251          tmp = trim(adjustl(cells(i)%str))
01252          n = len(tmp)
01253
01254          if (n>1) then
01255              ! if the first and last non-blank character is
01256              ! a quote, then remove them and replace with what
01257              ! is inside the quotes. Otherwise, leave it as is.
01258              if (tmp(1:1)==me%quote .and. tmp(n:n)==me%quote) then
01259                  if (n>2) then
01260                      cells(i)%str = tmp(2:n-1) ! remove the quotes
01261                  else
01262                      cells(i)%str = " ! empty string
01263                  end if
01264              end if
01265          end if
01266      end do
01267
01268      end subroutine tokenize_csv_line
01269
01270      !*****
01271      !*****
01272      !*****
01273      !
01277
01278      function number_of_lines_in_file(iunit) result(n_lines)
01279
01280      implicit none
01281
01282      integer,intent(in) :: iunit
01283      integer :: n_lines
01284
01285      character(len=1) :: tmp
01286      integer :: istat
01287
01288      rewind(iunit)
01289      n_lines = 0
01290      do
01291          read(iunit,fmt='(A1)',iostat=istat) tmp
01292          if (is_iostat_end(istat)) exit
01293          n_lines = n_lines + 1
01294      end do
01295      rewind(iunit)
01296
01297      end function number_of_lines_in_file
01298
01299      !*****
01300
01301      !*****
01302      !
01304
01305      subroutine read_line_from_file(me,iunit,line,status_ok)
01306
01307      implicit none
01308
01309      class(csv_file),intent(in) :: me
01310      integer,intent(in) :: iunit
01311      character(len=:),allocatable,intent(out) :: line
01312      logical,intent(out) :: status_ok
01313
01314      integer :: nread
01315      integer :: istat
01316      character(len=me%chunk_size) :: buffer
01317
01318      CALL setmessagesource("read_line_from_file")
01319
01320      nread = 0
01321      buffer = ""
01322      line = ""
01323      status_ok = .true.
01324
01325      do
01326          ! read in the next block of text from the line:
01327          read(iunit,fmt='(A)',advance='NO',size=nread,iostat=istat) buffer
01328          if (is_iostat_end(istat) .or. is_iostat_eor(istat)) then
01329              ! add the last block of text before the end of record
01330              if (nread>0) line = line//buffer(1:nread)
01331              exit
01332          else if (istat==0) then ! all the characters were read

```

```

01333         line = line//buffer ! add this block of text to the string
01334     else ! some kind of error
01335         WRITE(scratchmessage,'(A,1X,I5)') 'Read error for file unit: ', iunit
01336         CALL allmessage(error, scratchmessage)
01337         status_ok = .false.
01338         exit
01339     end if
01340 end do
01341
01342 CALL unsetmessagesource()
01343
01344 end subroutine read_line_from_file
01345 !*****
01346 !*****
01347 !*****
01348 !
01361
01362 pure subroutine split(str,token,chunk_size,vals)
01363
01364 implicit none
01365
01366 character(len=*),intent(in) :: str
01367 character(len=*),intent(in) :: token
01368 integer,intent(in) :: chunk_size
01369 type(csv_string),dimension(:),allocatable,intent(out) :: vals
01370
01371 integer :: i
01372 integer :: len_str
01373 integer :: len_token
01374 integer :: n_tokens
01375 integer :: i1
01376 integer :: i2
01377 integer :: j
01378 integer,dimension(:),allocatable :: itokens
01380
01381 len_token = len(token) ! length of the token
01382 n_tokens = 0 ! initialize the token counter
01383 j = 0 ! index to start looking for the next token
01384
01385 ! first, count the number of times the token
01386 ! appears in the string, and get the token indices.
01387 !
01388 ! Examples:
01389 ! ' ', ' --> 1
01390 ! '1234,67,90' --> 5,8
01391 ! '123, ' --> 4
01392
01393 ! length of the string
01394 if (token == ' ') then
01395     ! in this case, we can't ignore trailing space
01396     len_str = len(str)
01397 else
01398     ! safe to ignore trailing space when looking for tokens
01399     len_str = len_trim(str)
01400 end if
01401
01402 j = 1
01403 n_tokens = 0
01404 do
01405     if (j>len_str) exit ! end of string, finished
01406     i = index(str(j:),token) ! index of next token in remaining string
01407     if (i<=0) exit ! no more tokens found
01408     call expand_vector(itokens,n_tokens,chunk_size,i+j-1) ! save the token location
01409     j = j + i + (len_token - 1)
01410 end do
01411 call expand_vector(itokens,n_tokens,chunk_size,finished=.true.) ! resize the vector
01412
01413 allocate(vals(n_tokens+1))
01414
01415 if (n_tokens>0) then
01416
01417     len_str = len(str)
01418
01419     i1 = 1
01420     i2 = itokens(1)-1
01421     if (i2>=i1) then
01422         vals(1)%str = str(i1:i2)
01423     else
01424         vals(1)%str = " !the first character is a token
01425     end if
01426

```



```

01427      !      1 2 3
01428      !      'a,b,c,d'
01429
01430      do i=2,n_tokens
01431          i1 = itokens(i-1)+len_token
01432          i2 = itokens(i)-1
01433          if (i2>=i1) then
01434              vals(i)%str = str(i1:i2)
01435          else
01436              vals(i)%str = " !empty element (e.g., 'abc„def')
01437          end if
01438      end do
01439
01440      i1 = itokens(n_tokens) + len_token
01441      i2 = len_str
01442      if (itokens(n_tokens)+len_token<=len_str) then
01443          vals(n_tokens+1)%str = str(i1:i2)
01444      else
01445          vals(n_tokens+1)%str = " !the last character was a token
01446      end if
01447
01448      else
01449          !no tokens present, so just return the original string:
01450          vals(1)%str = str
01451      end if
01452
01453      end subroutine split
01454 !*****
01455
01456 !*****
01457      end module csv_module
01458 !*****

```

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.](#)

```

00001 !-----
00002 !           M O D U L E   C S V _ P A R A M E T E R S
00003 !-----
00014 !-----
00015
00016     module csv_parameters
00017
00018         USE pahm_sizes, ONLY : wp, ip
00019
00020         private
00021
00022         integer(ip),parameter,public :: max_real_str_len = 27
00023         character(len=*),parameter,public :: default_real_fmt = '(E27.17E4)'
00025
00026         integer(ip),parameter,public :: max_integer_str_len = 256
00027         character(len=*),parameter,public :: default_int_fmt = '(I256)'
00029
00030     end module csv_parameters
00031 !*****

```

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 ≤ 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.](#)

```

00001 !-----
00002 !               M O D U L E   C S V _ U T I L I T I E S
00003 !-----
00014 !-----
00015
00016     module csv_utilities
00017
00018         USE pahm_sizes, ONLY : wp, ip
00019         use csv_parameters
00020
00021     private
00022
00023     integer,parameter :: max_size_for_insertion_sort = 20
00024
00025     public :: unique
00026     public :: expand_vector
00027     public :: sort_ascending
00028
00029     contains
00030 !*****
00031
00032 !-----
00033 ! S U B R O U T I N E   E X P A N D _ V E C T O R
00034 !-----
00053 !-----
00054     pure subroutine expand_vector(vec, n, chunk_size, val, finished)
00055
00056     implicit none
00057
00058     integer,dimension(:),allocatable,intent(inout) :: vec
00059     integer,intent(inout) :: n ! counter for last element added to 'vec'.
00060                                ! must be initialized to 'size(vec)'
00061                                ! (or 0 if not allocated) before first call
00062     integer,intent(in) :: chunk_size ! allocate 'vec' in blocks of this size (>0)
00063     integer,intent(in),optional :: val ! the value to add to 'vec'
00064     logical,intent(in),optional :: finished ! set to true to return 'vec'
00065                                           ! as its correct size ('n')
00066
00067     integer,dimension(:),allocatable :: tmp ! temporary array
00068
00069     if (present(val)) then
00070         if (allocated(vec)) then
00071             if (n==size(vec)) then
00072                 ! have to add another chunk:
00073                 allocate(tmp(size(vec)+chunk_size))
00074                 tmp(1:size(vec)) = vec
00075                 call move_alloc(tmp,vec)
00076             end if
00077             n = n + 1
00078         else

```

```

00079         ! the first element:
00080         allocate(vec(chunk_size))
00081         n = 1
00082     end if
00083     vec(n) = val
00084 end if
00085
00086 if (present(finished)) then
00087     if (finished) then
00088         ! set vec to actual size (n):
00089         if (allocated(tmp)) deallocate(tmp)
00090         allocate(tmp(n))
00091         tmp = vec(1:n)
00092         call move_alloc(tmp,vec)
00093     end if
00094 end if
00095
00096 end subroutine expand_vector
00097 !*****
00098
00099 !*****
00100 !
00102
00103 function unique(vec,chunk_size) result(ivec_unique)
00104
00105 implicit none
00106
00107 integer,dimension(:),intent(in) :: vec
00108 integer,intent(in) :: chunk_size
00109 integer,dimension(:),allocatable :: ivec_unique
00110
00111 integer,dimension(size(vec)) :: ivec
00112 integer :: i
00113 integer :: n
00114
00115 ! first we sort it:
00116 ivec = vec ! make a copy
00117 call sort_ascending(ivec)
00118
00119 ! add the first element:
00120 n = 1
00121 ivec_unique = [ivec(1)]
00122
00123 ! walk through array and get the unique ones:
00124 if (size(ivec)>1) then
00125     do i = 2, size(ivec)
00126         if (ivec(i)/=ivec(i-1)) then
00127             call expand_vector(ivec_unique,n,chunk_size,val=ivec(i))
00128         end if
00129     end do
00130     call expand_vector(ivec_unique,n,chunk_size,finished=.true.)
00131 end if
00132
00133 end function unique
00134 !*****
00135
00136 !*****
00141
00142 subroutine sort_ascending(ivec)
00143
00144 implicit none
00145
00146 integer,dimension(:),intent(inout) :: ivec
00147
00148 call quicksort(1,size(ivec))
00149
00150 contains
00151
00152     recursive subroutine quicksort(ilow,ihigh)
00153
00154     implicit none
00155
00156     integer,intent(in) :: ilow
00157     integer,intent(in) :: ihigh
00158
00159     integer :: ipivot
00160     integer :: i
00161     integer :: j
00162
00163     if ( ihigh-ilow<=max_size_for_insertion_sort .and. ihigh>ilow ) then

```

```

00166
00167         ! do insertion sort:
00168         do i = ilow + 1, ihigh
00169             do j = i, ilow + 1, -1
00170                 if ( ivec(j) < ivec(j-1) ) then
00171                     call swap(ivec(j), ivec(j-1))
00172                 else
00173                     exit
00174                 end if
00175             end do
00176         end do
00177
00178         else if ( ihigh-ilow > max_size_for_insertion_sort ) then
00179
00180             ! do the normal quicksort:
00181             call partition(ilow, ihigh, ipivot)
00182             call quicksort(ilow, ipivot - 1)
00183             call quicksort(ipivot + 1, ihigh)
00184
00185         end if
00186
00187     end subroutine quicksort
00188
00189     subroutine partition(ilow, ihigh, ipivot)
00190
00191     implicit none
00192
00193     integer, intent(in) :: ilow
00194     integer, intent(in) :: ihigh
00195     integer, intent(out) :: ipivot
00196
00197     integer :: i, ip
00198
00199     call swap(ivec(ilow), ivec((ilow+ihigh)/2))
00200     ip = ilow
00201     do i = ilow + 1, ihigh
00202         if ( ivec(i) < ivec(ilow) ) then
00203             ip = ip + 1
00204             call swap(ivec(ip), ivec(i))
00205         end if
00206     end do
00207     call swap(ivec(ilow), ivec(ip))
00208     ipivot = ip
00209
00210     end subroutine partition
00211
00212 end subroutine sort_ascending
00213
00214 !*****
00215 !*****
00216 !*****
00217 !*****
00218 !*****
00219 !*****
00220
00221     pure elemental subroutine swap(i1, i2)
00222
00223     implicit none
00224
00225     integer, intent(inout) :: i1
00226     integer, intent(inout) :: i2
00227
00228     integer :: tmp
00229
00230     tmp = i1
00231     i1 = i2
00232     i2 = tmp
00233
00234     end subroutine swap
00235
00236 !*****
00237 !*****
00238 !*****
00239 end module csv_utilities
00240 !*****

```

9.10 doxy_sample.F90 File Reference

9.11 doxy_sample.F90

[Go to the documentation of this file.](#)

```
00001 finished: netcdfio.f90
00002 !-----
00003 !           M O D U L E   S I Z E S
00004 !-----
00014 !-----
00015
00016
00017
00018 !-----
00019 ! F U N C T I O N   CompareDoubleReals
00020 !-----
00056 !-----
```

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.](#)

```

00001 !-----
00002 !           M O D U L E   P A H M   D R I V E R   M O D
00003 !-----
00013 !-----
00014
00015 MODULE pahm_drivermod
00016
00017   USE pahm_messages
00018   USE utilities
00019   !USE TimeDateUtils
00020
00021   IMPLICIT NONE
00022
00023   INTEGER, SAVE :: cnttimebegin, cnttimeend
00024
00025
00026 CONTAINS
00027
00028
00029 !-----
00030 !           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
00031 !-----
00039 !-----
00040 SUBROUTINE getprogramcmdlargs()
00041
00042   USE pahm_global, ONLY : controldatafilename
00043
00044   IMPLICIT NONE
00045
00046   INTEGER           :: argNumb, argCnt      ! number of command line arguments and argument counter
00047   CHARACTER(1024) :: argCmdLine
00048
00049   CALL initlogging()
00050
00051   argnumb = iargc()
00052   IF (argnumb > 0) THEN
00053     argcnt = 0
00054     DO WHILE (argcnt < argnumb)
00055       argcnt = argcnt + 1
00056       CALL getarg(argcnt, argcmdline)
00057
00058       SELECT CASE(trim(argcmdline))
00059         CASE("-v", "-v", "--v", "--v", "--version")
00060           CALL programversion
00061           stop
00062
00063         CASE("-h", "-h", "--h", "--h", "--help")
00064           CALL programhelp
00065           stop
00066
00067         CASE DEFAULT
00068           ! Do nothing
00069       END SELECT
00070     END DO
00071     ! This is the first argument if not "-v/-h" were supplied.
00072     ! It is assumed that this argument is the filename of the user control file.
00073     CALL getarg(1, argcmdline)
00074     controldatafilename = trim(adjustl(argcmdline))
00075   ENDIF
00076
00077   CALL readcontrolfile(trim(controldatafilename))
00078
00079 END SUBROUTINE getprogramcmdlargs
00080
00081 !-----
00082
00083 !-----
00084 !           S U B R O U T I N E   P A H M   M O D E L   I N I T
00085 !-----
00093 !-----
00094 SUBROUTINE pahm_init()
00095
00096   USE pahm_global, ONLY : noutdt
00097   USE pahm_mesh, ONLY : readmesh
00098   USE parwind, ONLY : readcsvbesttrackfile
00099

```

```

00100      ! Initialize the logging system, needs to be called first
00101      CALL initlogging()
00102
00103      CALL setmessagesource("PaHM_Init")
00104
00105      ! Get possible command line arguments
00106      CALL getprogramcmdlargs()
00107
00108      ! Read the mesh/grid of the domain or the generic mesh/grid input file
00109      CALL readmesh()
00110
00111      ! Read all track files and save the data into the array of the best track structures
00112      ! for subsequent access by the P-W models in the program
00113      !CALL ReadBestTrackFile()
00114      CALL readcsvbesttrackfile()
00115
00116      cnttimebegin = 1
00117      cnttimeend   = noutdt
00118
00119      CALL unsetmessagesource()
00120
00121      END SUBROUTINE pahm_init
00122
00123      !=====
00124
00125      !-----
00126      !  SUBROUTINE  PAHM MODEL RUN
00127      !-----
00128      !-----
00136      SUBROUTINE pahm_run(nTimeSTP)
00137
00138          USE pahm_global, ONLY : modeltype, wvelx, wvely, wpress, times
00139          USE parwind, ONLY : gethollandfields
00140          USE pahm_netcdfio
00141
00142          IMPLICIT NONE
00143
00144          INTEGER, INTENT(IN), OPTIONAL :: nTimeSTP
00145
00146          INTEGER :: iCnt
00147
00148          CALL setmessagesource("PaHM_Run")
00149
00150          IF (PRESENT(ntimestp)) THEN
00151              cnttimeend = cnttimebegin + ntimestp - 1
00152          ENDIF
00153
00154          SELECT CASE (modeltype)
00155              CASE (1)
00156                  DO icnt = cnttimebegin, cnttimeend
00157                      CALL gethollandfields(icnt)
00158                      IF (outfilenamespecified) THEN
00159                          ! Create the output NetCDF file and fill it with the static data only
00160                          ! Initialize all variables. This subroutine is called just once
00161                          CALL initadcircnetcdfoutfile(outfilename)
00162
00163                          CALL writenetcdfrecord(outfilename, icnt)
00164                      END IF
00165                  END DO
00166              CASE DEFAULT
00167                  WRITE(scratchmessage, '(a, i0)') &
00168                      'This model type is not supported: modelType = ', modeltype
00169                  CALL logmessage(error, scratchmessage)
00170              END SELECT
00171
00172          IF (PRESENT(ntimestp)) THEN
00173              cnttimebegin = cnttimeend + 1
00174          ENDIF
00175
00176          CALL unsetmessagesource()
00177
00178      END SUBROUTINE pahm_run
00179
00180      !=====
00181
00182      !-----
00183      !  SUBROUTINE  PAHM MODEL FINALIZE
00184      !-----
00185      !-----
00194      SUBROUTINE pahm_finalize()

```



```

00195
00196     CALL setmessagesource("PaHM_Finalize")
00197
00198     CALL unsetmessagesource()
00199
00200     ! Close the logging facilities
00201     CALL closelogfile()
00202
00203 END SUBROUTINE pahm_finalize
00204
00205 !=====
00206
00207 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::logfile` = 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_PRE
- 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.](#)

```

00001 !-----
00002 !               M O D U L E   G L O B A L
00003 !-----
00013 !-----
00014
00015 MODULE pahm_global
00016
00017   USE version
00018   USE pahm_sizes
00019
00020   IMPLICIT NONE
00021
00022 !#####
00023 !###   BEG:: LUN NUMBERS FOR I/O OPERATIONS
00024 !#####
00025   INTEGER, PARAMETER :: lun_screen = 6   ! I/O unit where screen output is sent
00026   INTEGER, PARAMETER :: lun_ctrl   = 10  ! I/O unit for the model's control file
00027   INTEGER, PARAMETER :: lun_inp    = 14  ! I/O unit for the input files (mesh)

```

```

00028 INTEGER, PARAMETER :: lun_inpl = 15 ! I/O unit for the input files (mesh)
00029 INTEGER, PARAMETER :: lun_log = 35 ! I/O unit where log output is sent
00030 INTEGER, PARAMETER :: lun_btrk = 22 ! I/O unit for the best track files
00031 INTEGER, PARAMETER :: lun_btrk1 = 23 ! I/O unit for the best track files
00032 INTEGER, PARAMETER :: lun_out = 25 ! I/O unit for the output files
00033 INTEGER, PARAMETER :: lun_out1 = 26 ! I/O unit for the output files
00034 !#####
00035 !### END:: LUN NUMBERS FOR I/O OPERATIONS
00036 !#####
00037
00038
00039 !#####
00040 !### BEG:: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
00041 !#####
00042 REAL(sz), PARAMETER :: defv_gravity = 9.80665_sz ! Default (standard) gravitational acceleration
(m/s^2)
00043 REAL(sz), PARAMETER :: defv_atmpress = 1013.25_sz ! Default (standard) atmospheric pressure (mb)
00044
00045 REAL(sz), PARAMETER :: defv_rhoair = 1.1478_sz ! Default (standard) density of air at STP
(kg/m^3)
00046
00047 ! 1.1478 (1013.25 mb, Rel. Hum 90%, 30 deg C)
00048
00049 ! Water density is used in the code to convert the pressure to units of mH2O
00049 REAL(sz), PARAMETER :: defv_rhowater = 1000.0000 ! Default density of fresh water (kg/m^3)
00050 !--- FRESH WATER
00051 ! 999.8900 ( 0 deg C)
00052 ! 1000.0000 ( 4 deg C)
00053 ! 999.7025 (10 deg C)
00054 ! 999.1026 (15 deg C)
00055 ! 998.2072 (20 deg C)
00056 ! 997.0476 (25 deg C)
00057 ! 995.6495 (30 deg C)
00058 ! 994.0333 (35 deg C)
00059 ! 992.2164 (40 deg C)
00060 !--- SEA WATER
00061 ! 1028.0941 (35% S, 1 deg C)
00062 ! 1027.8336 (35% S, 4 deg C)
00063 ! 1027.0000 (35% S, 10 deg C)
00064 ! 1026.0210 (35% S, 15 deg C)
00065 ! 1024.8103 (35% S, 20 deg C)
00066 ! 1023.3873 (35% S, 25 deg C)
00067 ! 1021.7694 (35% S, 30 deg C)
00068 ! 1019.9000 (35% S, 35 deg C)
00069 ! 1018.0000 (35% S, 40 deg C)
00070
00071 ! 1-min to 10-min wind conversion factors
00072 REAL(sz), PARAMETER :: one2ten = 0.8928_sz
00073 REAL(sz), PARAMETER :: ten2one = 1.0_sz / 0.8928_sz
00074
00075 REAL(sz), PARAMETER :: pi = 3.141592653589793_sz
00076 REAL(sz), PARAMETER :: deg2rad = pi / 180.0_sz ! degrees to radians
00077 REAL(sz), PARAMETER :: rad2deg = 180.0_sz / pi ! radians to degrees
00078 REAL(sz), PARAMETER :: basee = 2.718281828459045_sz ! mathematical constant e (natural logarithm base)
00079
00080 REAL(sz), PARAMETER :: rearth = 6378206.4_sz ! radius of earth (m) (Clarke 1866 major spheroid
radius)
00081 REAL(sz), PARAMETER :: nm2m = 1852.0_sz ! nautical miles to meters
00082 REAL(sz), PARAMETER :: m2nm = 1.0_sz / nm2m ! meters to nautical miles
00083 REAL(sz), PARAMETER :: kt2ms = nm2m / 3600.0_sz ! knots to m/s
00084 REAL(sz), PARAMETER :: ms2kt = 1.0_sz / kt2ms ! m/s to knots
00085 REAL(sz), PARAMETER :: omega = 2.0_sz * pi / 86164.2_sz
00086 REAL(sz), PARAMETER :: mb2pa = 100.0_sz
00087 REAL(sz), PARAMETER :: mb2kpa = 0.1_sz
00088 !#####
00089 !### END:: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
00090 !#####
00091
00092
00093 !#####
00094 !### BEG :: VARIABLES RELATED TO THE CONTROL FILE
00095 !#####
00096 CHARACTER(LEN=FNAMLEN) :: logfile = trim(adjustl(prog_name_low)) // '_model.log'
00097
00098 !----- Input files
00099 CHARACTER(FNAMLEN) :: ctrlfilename = trim(adjustl(prog_name_low)) // '_control.in' ! default
value
00100
00101 LOGICAL :: meshfilenamespecified = .false. ! .TRUE. if the user supplied a valid
filename
00102 CHARACTER(LEN=FNAMLEN) :: meshfilename = blank ! there is no default value here
00103 CHARACTER(LEN=64) :: meshfiletype = blank ! ADCIRC, SCHISM, FVCOM, ROMS, GENERIC

```

```

00104 (no default)
00105 CHARACTER (LEN=64)      :: meshfileform = blank           ! ASCII, NETCDF (no default)
00106 LOGICAL                  :: besttrackfilenamespecified = .false.
00107 INTEGER                  :: nbtrfiles = imissv
00108 CHARACTER (LEN=FNAMELEN), ALLOCATABLE :: besttrackfilename(:)
00109 !-----
00110
00111 !----- Other parameters in the control file
00112 CHARACTER (LEN=512)      :: title = blank
00113
00114 REAL (sz)                :: gravity          = defv_gravity    ! m/s^2    Gravitational acceleration
00115 REAL (sz)                :: rhowater         = defv_rhowater    ! kg/m^3    Mean water density
00116 REAL (sz)                :: rhoair          = defv_rhoair      ! kg/m^3    Mean air density
00117 REAL (sz)                :: backgroundatmpress = defv_atmpress  ! mb        Background atmospheric
pressure
00118
00119 ! This is for the BL reduction factor used in the Holland model
00120 REAL (sz), PARAMETER     :: defv_bladjustfac = 0.9_sz
00121 REAL (sz)                :: windreduction    = defv_bladjustfac ! PV BL reduction factor used in the
Holland model
00122 REAL (sz)                :: bladjustfac      = defv_bladjustfac !PV same as windReduction?
00123
00124 !=====
00125 !=== This block is for the : time/date and time stepping variables
00126 !=====
00127 !---
00128 ! the reference date/time for the model run YYYYMMDDhhmmss
00129 CHARACTER (LEN=64)      :: refdatetime      = blank
00130 INTEGER                  :: refdate         = imissv
00131 INTEGER                  :: reftime         = imissv
00132 INTEGER                  :: refyear         = imissv
00133 INTEGER                  :: refmonth        = 0
00134 INTEGER                  :: refday         = 0
00135 INTEGER                  :: refhour        = 0
00136 INTEGER                  :: refmin         = 0
00137 INTEGER                  :: refsec         = 0
00138 LOGICAL                  :: refdatespecified = .false.
00139 !---
00140 ! the start date/time for the model run YYYYMMDDhhmmss
00141 CHARACTER (LEN=64)      :: begdatetime      = blank
00142 INTEGER                  :: begdate         = imissv
00143 INTEGER                  :: begtime         = imissv
00144 INTEGER                  :: begyear         = imissv
00145 INTEGER                  :: begmonth        = 0
00146 INTEGER                  :: begday         = 0
00147 INTEGER                  :: beghour        = 0
00148 INTEGER                  :: begmin         = 0
00149 INTEGER                  :: begsec         = 0
00150 LOGICAL                  :: begdatespecified = .false.
00151 !---
00152 ! the stop date/time for the model run YYYYMMDDhhmmss
00153 CHARACTER (LEN=64)      :: enddatetime      = blank
00154 INTEGER                  :: enddate         = imissv
00155 INTEGER                  :: endtime         = imissv
00156 INTEGER                  :: endyear         = imissv
00157 INTEGER                  :: endmonth        = 0
00158 INTEGER                  :: endday         = 0
00159 INTEGER                  :: endhour        = 0
00160 INTEGER                  :: endmin         = 0
00161 INTEGER                  :: endsec         = 0
00162 LOGICAL                  :: enddatespecified = .false.
00163 !---
00164 ! alternative definitions for the stop date/time for the model run
00165 REAL (sz)                :: begsimtime      = rmissv
00166 REAL (sz)                :: endsimtime      = rmissv
00167 LOGICAL                  :: begsimspecified = .false.
00168 LOGICAL                  :: endsimspecified = .false.
00169
00170 CHARACTER (LEN=1)        :: unittime = 'S'
00171 !=====
00172
00173 !---
00174 ! time stepping variables for the model run
00175 REAL (sz)                :: outdt          = rmissv
00176 INTEGER                  :: noutdt         = imissv
00177 REAL (sz)                :: mdoutdt       = rmissv
00178 REAL (sz)                :: mdbegsimtime  = rmissv
00179 REAL (sz)                :: mdendsimtime  = rmissv
00180
00181 LOGICAL                  :: outfilenamespecified = .false.

```

```

00182 CHARACTER(LEN=FNAMLEN) :: outfile = blank ! Name of the output NetCDF file
00183 INTEGER :: ncshuff = 0 ! Turn on the shuffle filter (>0)
00184 INTEGER :: ncdeflate = 0 ! Turn on the deflate filter (>0)
00185 INTEGER :: ncdlevel = 0 ! Deflate level [0-9]
00186
00187 ! Create a list of NetCDF variable names in the form ncYyyyVarNam = value
00188 ! The user can specify his/her own values in the control file (will be hidden variables)
00189 ! Default values
00190 CHARACTER(LEN=20), PARAMETER :: def_ncnam_pres = 'P', &
00191                                def_ncnam_wndx = 'uwnd', &
00192                                def_ncnam_wndy = 'vwnd'
00193
00194 CHARACTER(LEN=20) :: ncvarnam_pres = def_ncnam_pres, &
00195                                ncvarnam_wndx = def_ncnam_wndx, &
00196                                ncvarnam_wndy = def_ncnam_wndy
00197
00198 INTEGER :: modeltype = imissv ! The parametric model to use
00199                                ! 0: Rankin Vortex
00200                                ! 1: Holland B (1998)
00201                                ! 2: Holland B (2010)
00202                                ! 3: Willoughby model - NOT IMPLEMENTED YET
00203                                ! 9: Assymetric vortex model (Mattocks)
00204                                ! 10: Generalized assymetric vortex Holland model
00205 LOGICAL :: writeparams = .false.
00206 #####
00207 !### END :: VARIABLES RELATED TO THE CONTROL FILE
00208 #####
00209
00210
00211 #####
00212 !### BEG :: GLOBAL DATA ARRAYS
00213 #####
00214 ! Arrays to hold the P-W fields
00215 ! REAL(SZ), DIMENSION(:, :), ALLOCATABLE :: wVelX, wVelY, wPress
00216 REAL(sz), DIMENSION(:, :), ALLOCATABLE :: wvelx, wvely, wpress
00217 REAL(sz), DIMENSION(:, :), ALLOCATABLE :: times
00218 #####
00219 !### END :: GLOBAL DATA ARRAYS
00220 #####
00221
00222
00223 CONTAINS
00224
00225
00226 !-----
00227 ! FUNCTION AIR DENSITY
00228 !-----
00237 ! >@see http://www.emd.dk/files/windpro/WindPRO\_AirDensity.pdf
00238 !-----
00250 REAL(sz) function airdensity(atmt, atmp, relhum) result(myvalout)
00251
00252 IMPLICIT NONE
00253
00254 REAL(sz), INTENT(IN) :: atmt ! Surface temperature in degrees C (-50.0 <= T <= 100.0)
00255 REAL(sz), INTENT(IN) :: atmp ! Atmospheric pressure (mb)
00256 REAL(sz), INTENT(IN) :: relhum ! Relative humidity (0 - 100)
00257
00258 ! Local variables
00259 REAL(hp) :: es, p, pv, pd, rh
00260 REAL(hp) :: rd, rv, temp, tempk, dens
00261
00262
00263 rh = relhum
00264 IF (rh < 0.01) rh = 0.01_hp
00265 IF (rh > 100.0) rh = 100.0_hp
00266
00267 temp = atmt
00268 IF (temp < -50.0) temp = -50.0_hp
00269 IF (temp > 100.0) temp = 100.0_hp
00270
00271 rd = 287.058_hp ! specific gas constant for dry air (J/kg*K)
00272 rv = 461.495_hp ! specific gas constant for water vapor (J/kg*K)
00273
00274 ! Convert relative humidity to %
00275 rh = 0.01_sz * rh
00276
00277 ! Convert atmT (C) to K
00278 tempk = temp + 273.15_hp
00279
00280 ! Calculate the saturated vapor pressure (mb)
00281 ! Temperature is in degrees Celcius

```

```

00282      p = 0.99999683e+00_hp + temp * (-0.90826951e-02_hp + temp * (0.78736169e-04_hp + temp * &
00283      (-0.61117958e-06_hp + temp * (0.43884187e-08_hp + temp * &
00284      (-0.29883885e-10_hp + temp * (0.21874425e-12_hp + temp * &
00285      (-0.17892321e-14_hp + temp * (0.11112018e-16_hp + temp * &
00286      (-0.30994571e-19_hp)))))))))
00287      es = 6.1078_hp / p**8      ! saturated vapour pressure (mb)
00288
00289      ! Calculate the actual vapor pressure (mb)
00290      pv = es * rh
00291
00292      ! Calculate the actual vapor pressure
00293      pd = atmp - pv
00294
00295      ! Convert the pressures from mb to Pa
00296      pd = pd * 100.0_hp
00297      pv = pv * 100.0_hp
00298
00299      ! Calculate the air density
00300      dens = pd / (rd * tempk) + pv / (rv * tempk)
00301
00302      myvalout = dens
00303
00304      RETURN
00305
00306  END FUNCTION airdensity
00307
00308  !=====
00309
00310  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.](#)

```

00001 !-----
00002 !           M O D U L E   M E S H
00003 !-----
00023 !-----
00024
00025 MODULE pahm_mesh
00026
00027   USE pahm_sizes
00028   USE pahm_messages
00029
00030   IMPLICIT NONE
00031
00032   CHARACTER (LEN=80)      :: agrid
00033   INTEGER                 :: np = imissv      ! number of nodes in the mesh
00034   INTEGER                 :: ne = imissv      ! number of elements in the mesh
00035   INTEGER                 :: ics              ! mesh coordinate system (1=cartesian, 2=geographic)
00036   REAL (sz), ALLOCATABLE :: dp(:)           ! bathymetric depth
00037   INTEGER, ALLOCATABLE    :: nfn(:)          ! element number of face nodes (ne)
00038   INTEGER, ALLOCATABLE    :: nm(:, :)        ! element table size(ne, nfn)
00039   REAL (sz), ALLOCATABLE  :: slam(:)         ! longitude node locations in CPP slam(np)
00040   REAL (sz), ALLOCATABLE  :: sfea(:)         ! latitude node locations in CPP sfea(np)
00041   REAL (sz), ALLOCATABLE  :: xcslam(:)       ! x cartesian node locations xcSlam(np)
00042   REAL (sz), ALLOCATABLE  :: ycsfea(:)       ! y cartesian node locations ycSfea(np)
00043

```



```

00044 REAL(sz)                :: slam0 = rmissv ! center point of CPP spherical projection
00045 REAL(sz)                :: sfea0 = rmissv ! center point of CPP spherical projection
00046
00047 ! The maximum number of faces of an element
00048 INTEGER, PARAMETER      :: maxfacenodes = 5
00049
00050 ! This varibale is set to .TRUE. if the mesh file read successfully
00051 LOGICAL                  :: ismeshok = .false.
00052
00053 CONTAINS
00054
00055 !-----
00056 ! SUBROUTINE READ MESH
00057 !-----
00058 SUBROUTINE readmesh()
00059
00060     USE pahm_global, ONLY : meshfilenamespecified, meshfilename, meshfiletype, meshfileform
00061     USE utilities, ONLY : touppercase
00062
00063     IMPLICIT NONE
00064
00065     CALL setmessagesource("ReadMesh")
00066
00067     IF (meshfilenamespecified .EQV. .false.) THEN
00068         WRITE(scratchmessage, '(a)') 'ReadMesh: First specify a valid grid filename to proceed: ' // &
00069             '[' // trim(meshfilename) // ']'
00070         CALL allmessage(error, scratchmessage)
00071         CALL terminate()
00072     END IF
00073
00074     SELECT CASE(touppercase(meshfiletype))
00075     !----- ADCIRC case
00076     CASE('ADCIRC')
00077         SELECT CASE(touppercase(meshfileform))
00078         CASE('ASCII')
00079             CALL readmeshasciifort14()
00080         CASE('NETCDF')
00081             WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file
00082 type: ' // &
00083                 '[' // trim(meshfiletype) // ']'
00084             CALL allmessage(error, scratchmessage)
00085             CALL terminate()
00086         CASE DEFAULT
00087             WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the
00088 mesh file type: ' // &
00089                 '[' // trim(meshfiletype) // ']'
00090             CALL allmessage(error, scratchmessage)
00091             CALL terminate()
00092         END SELECT
00093     !----- SCHISM case
00094     CASE('SCHISM')
00095         SELECT CASE(touppercase(meshfileform))
00096         CASE('ASCII')
00097             CALL readmeshasciifort14()
00098         CASE('NETCDF')
00099             WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file
00100 type: ' // &
00101                 '[' // trim(meshfiletype) // ']'
00102             CALL allmessage(error, scratchmessage)
00103             CALL terminate()
00104         CASE DEFAULT
00105             WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the
00106 mesh file type: ' // &
00107                 '[' // trim(meshfiletype) // ']'
00108             CALL allmessage(error, scratchmessage)
00109             CALL terminate()
00110         END SELECT
00111     !----- FVCOM case
00112     CASE('FVCOM')
00113         WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet implemented: ' // &
00114             '[' // trim(meshfiletype) // ']'
00115     END SELECT

```

```

00129     CALL allmessage(error, scratchmessage)
00130     CALL terminate()
00131
00132     !----- ROMS case
00133     CASE('ROMS')
00134         WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet implemented: ' // &
00135             '[' // trim(meshfiletype) // ']'
00136         CALL allmessage(error, scratchmessage)
00137         CALL terminate()
00138
00139     !----- GENERIC case
00140     CASE('GENERIC')
00141         WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet implemented: ' // &
00142             '[' // trim(meshfiletype) // ']'
00143         CALL allmessage(error, scratchmessage)
00144         CALL terminate()
00145
00146     CASE DEFAULT
00147         WRITE(scratchmessage, '(a)') 'ReadMesh: Invalid mesh file type specified: ' // &
00148             '[' // trim(meshfiletype) // ']'
00149         CALL allmessage(error, scratchmessage)
00150         CALL terminate()
00151     END SELECT
00152
00153     CALL unsetmessagesource()
00154
00155 END SUBROUTINE readmesh
00156
00157 !=====
00158
00159 !-----
00160 ! SUBROUTINE READ MESH ASCII FORT 14
00161 !-----
00162 !-----
00163 !-----
00170 SUBROUTINE readmeshasciifort14()
00171
00172     USE pahm_global, ONLY : lun_inp, meshfilename
00173     USE utilities !PV specify what are we using here from utilities
00174
00175     IMPLICIT NONE
00176
00177     INTEGER, PARAMETER :: iUnit = lun_inp      ! LUN for read operations
00178     INTEGER              :: ios                 ! I/O status
00179     CHARACTER(LEN=512) :: fmtStr               ! String to hold formats for I/O
00180     INTEGER              :: lineNum            ! Line number currently being read
00181
00182     INTEGER              :: labNodes, numFNodes ! Label and number of nodal faces for that label
00183     INTEGER              :: iCnt               ! Counters
00184
00185
00186     CALL setmessagesource("ReadMeshASCII Fort14")
00187
00188     CALL openfileforread(iunit, trim(meshfilename), ios)
00189
00190     linenum = 1
00191
00192     READ(unit=iunit, fmt='(a80)', err=10, END=20, IOSTAT=ios) agrid
00193     linenum = linenum + 1
00194
00195     CALL logmessage(info, "Reading the mesh file: " // trim(meshfilename))
00196     CALL logmessage(info, "Mesh file comment line: " // trim(agrid))
00197     CALL logmessage(info, "Reading mesh file dimensions and coordinates.")
00198
00199     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) ne, np
00200     linenum = linenum + 1
00201
00202     CALL allocatenodalandelementalarrays()
00203
00204     ! N O D E   T A B L E
00205     DO icnt = 1, np
00206         READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, slam(icnt), sfea(icnt), dp(icnt)
00207
00208         ! Check for (invalid longitude, latitude) values.
00209         ! Currently only geographical coordinates are supported.
00210         IF (.NOT. ((slam(icnt) >= -180.0_sz) .AND. (slam(icnt) <= 180.0_sz)) .OR. &
00211             .NOT. ((sfea(icnt) >= -90.0_sz) .AND. (sfea(icnt) <= 90.0_sz))) THEN
00212
00213             fmtstr = ' ("Input file: ' // trim(meshfilename) // '", line ', i0, '
00214             fmtstr = trim(fmtstr) // ' " contains invalid (lon, lat) values: ', " [", f14.4, ", ", f14.4,
00215             "]", '
             fmtstr = trim(fmtstr) // ' " (should be degrees east and degrees north)" )'

```

```

00216         WRITE(scratchmessage, trim(fmtstr)) linenum, slam(icnt), sfea(icnt)
00217
00218         CALL allmessage(error, scratchmessage)
00219         CLOSE(iunit)
00220         CALL terminate()
00221     END IF
00222
00223     linenum = linenum + 1
00224 END DO
00225
00226 ! E L E M E N T   T A B L E
00227 DO icnt = 1, ne
00228     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, numfnodes
00229
00230     ! Check if numfnodes in the line is beyond the value of parameter MAXFACENODES,
00231     ! to avoid out of bounds errors for the array "nm".
00232     IF (numfnodes > maxfacenodes) THEN
00233         fmtstr = '("Input file: ' // trim(meshfilename) // ", reading line ", i0, '
00234             fmtstr = trim(fmtstr) // ' " gave a number of face nodes equal to: ", i0, '
00235             fmtstr = trim(fmtstr) // ' ", which is greater than MAXFACENODES")'
00236         WRITE(scratchmessage, trim(fmtstr)) linenum, numfnodes
00237
00238         CALL allmessage(error, scratchmessage)
00239         CLOSE(iunit)
00240         CALL terminate()
00241     ELSE
00242         backspace(unit=iunit)
00243     END IF
00244
00245     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, nfn(iCnt), nm(iCnt, 1:nfn(iCnt))
00246
00247     linenum = linenum + 1
00248 END DO
00249
00250 CLOSE(iunit)
00251
00252 !PV Need to also check if arrays contain any missing values
00253 IF ((comparereals(slam0, rmissv) == 0) .OR. &
00254     (comparereals(sfea0, rmissv) == 0)) THEN
00255     slam0 = sum(slam, 1) / np
00256     sfea0 = sum(sfea, 1) / np
00257 END IF
00258
00259 CALL geotocpp(sfea, slam, sfea0, slam0, xcslam, ycsfea)
00260
00261 CALL logmessage(info, 'Finished reading mesh file dimensions and coordinates.')
00262
00263 CALL unsetmessagesource()
00264
00265 ismeshok = .true.
00266
00267 RETURN
00268
00269 ! Jump to here on error condition during read
00270 10 fmtstr = '("Reading line ", i0, " gave the following error code: ", i0, ".")'
00271 WRITE(scratchmessage, fmtstr) linenum, ios
00272
00273 CALL allmessage(error, scratchmessage)
00274 CLOSE(iunit)
00275 CALL terminate()
00276
00277 ! Jump to here on end condition during read
00278 20 fmtstr = '("Reached premature end of file on line ", i0, ".")'
00279 WRITE(scratchmessage, trim(fmtstr)) linenum
00280
00281 CALL allmessage(error, scratchmessage)
00282 CLOSE(iunit)
00283 CALL terminate()
00284
00285 END SUBROUTINE readmeshasciifort14
00286
00287 !=====
00288
00289 !-----
00290 ! SUBROUTINE ALLOCATE NODAL AND ELEMENTAL ARRAYS
00291 !-----
00300
00301 SUBROUTINE allocatenodalandelementalarrays()
00302
00303     IMPLICIT NONE
00304

```

```

00305     CALL setmessagesource("AllocateNodalAndElementalArrays")
00306
00307     ALLOCATE(slam(np), sfea(np), xcslam(np), ycsfea(np), dp(np))
00308
00309     ALLOCATE(nfn(ne), nm(ne, maxfacenodes))
00310
00311     ! Initialize to something troublesome to make it easy to spot issues
00312     slam = rmissv
00313     sfea = rmissv
00314     xcslam = rmissv
00315     ycsfea = rmissv
00316     dp = rmissv
00317     nm = imissv
00318     nfn = imissv
00319
00320     CALL unsetmessagesource()
00321
00322     END SUBROUTINE allocatenodalandelementalarrays
00323
00324     !=====
00325
00326
00327 END MODULE pahm_mesh
00328

```

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 writing.
- 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::sourcenum](#)
- 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.](#)

```

00001 !-----
00002 !           M O D U L E   M E S S A G E S
00003 !-----
00014 !-----
00015
00016 MODULE pahm_messages
00017
00018   USE pahm_sizes, ONLY : fnamelen
00019   USE pahm_global, ONLY : lun_screen, lun_log, logfilename
00020
00021 #ifdef __INTEL_COMPILER
00022   USE ifport
00023 #endif
00024
00025   IMPLICIT NONE
00026
00027   INTEGER                                :: nscreen = 1      ! >= 1: write to screen, <=0 do not write to
                                screen
00028
00029   ! Logging levels
00030   INTEGER, PARAMETER               :: debug   = -1      ! write all messages and echo input
00031   INTEGER, PARAMETER               :: echo    =  0      ! echo input, plus write all non-debug
00032   INTEGER, PARAMETER               :: info    =  1      ! don't echo input; write all non-debug
00033   INTEGER, PARAMETER               :: warning =  2      ! don't echo input; write only warn/err
00034   INTEGER, PARAMETER               :: error   =  3      ! don't echo input; only fatal msgs
00035
00036   CHARACTER(LEN=10), DIMENSION(5) :: loglevelnames
00037   CHARACTER(LEN=50), DIMENSION(100) :: messagesources ! subroutine names
00038   CHARACTER(LEN=1024)               :: scratchmessage ! used for formatted messages
00039   CHARACTER(LEN=1024)               :: scratchformat  ! used for Fortran format strings
00040   INTEGER                           :: sourcenumber   ! index into messageSources for current sub
00041
00042   ! Logging flags
00043   LOGICAL                           :: logfileopened = .false.
00044   LOGICAL                           :: loginitcalled = .false.
00045
00046 !-----
00047 ! I N T E R F A C E S
00048 !-----
00049 INTERFACE logmessage
00050   MODULE PROCEDURE logmessage_1
00051   MODULE PROCEDURE logmessage_2
00052 END INTERFACE logmessage
00053
00054 INTERFACE screenmessage
00055   MODULE PROCEDURE screenmessage_1
00056   MODULE PROCEDURE screenmessage_2
00057 END INTERFACE screenmessage
00058
00059 INTERFACE allmessage
00060   MODULE PROCEDURE allmessage_1
00061   MODULE PROCEDURE allmessage_2
00062 END INTERFACE allmessage
00063 !-----
00064
00065
00066 CONTAINS
00067
00068
00069 !-----
00070 !           S U B R O U T I N E   I N I T   L O G G I N G
00071 !-----
00080
00081 SUBROUTINE initlogging()
00082
00083   IMPLICIT NONE
00084
00085   IF (loginitcalled .EQV. .false.) THEN
00086     sourcenumber = 0
00087     loglevelnames(1) = "DEBUG"
00088     loglevelnames(2) = "ECHO"
00089     loglevelnames(3) = "INFO"
00090     loglevelnames(4) = "WARNING"
00091     loglevelnames(5) = "ERROR"
00092
00093     loginitcalled = .true.

```

```

00094
00095     CALL openlogfile
00096     END IF
00097
00098 END SUBROUTINE initlogging
00099
00100 !=====
00101
00102 !-----
00103 !   S U B R O U T I N E   O P E N   L O G   F I L E
00104 !-----
00112 !-----
00113 SUBROUTINE openlogfile()
00114
00115     IMPLICIT NONE
00116
00117     INTEGER :: errorIO    ! zero if the file opened successfully
00118
00119     logfileopened = .false.
00120
00121     OPEN(unit=lun_log, file=trim(adjustl(logfilename)), action='WRITE', status='REPLACE', iostat=errorio)
00122
00123     IF (errorio == 0) THEN
00124         logfileopened = .true.
00125     ELSE
00126         WRITE(scratchmessage, '(a, i0, a, i0)')
00127         'Could not open the log file = ' // trim(adjustl(logfilename)) //
00128         ' on logical unit LUN_LOG = ', lun_log,
00129         ' . Error code was: errorIO = ', errorio
00130         CALL screenmessage(error, scratchmessage)
00131     END IF
00132
00133 END SUBROUTINE openlogfile
00134
00135 !=====
00136
00137 !-----
00138 !   S U B R O U T I N E   C L O S E   L O G   F I L E
00139 !-----
00147 !-----
00148 SUBROUTINE closelogfile()
00149
00150     IMPLICIT NONE
00151
00152     IF (logfileopened) CLOSE(unit=lun_log)
00153
00154 END SUBROUTINE closelogfile
00155
00156 !=====
00157
00158 !-----
00159 !   S U B R O U T I N E   S C R E E N   M E S S A G E
00160 !-----
00176 !-----
00177 SUBROUTINE screenmessage_1(message)
00178
00179     IMPLICIT NONE
00180
00181     ! Global variables
00182     CHARACTER(LEN=*) , INTENT(IN) :: message
00183
00184     IF (nscreen > 0) THEN
00185         IF (loginitcalled) THEN
00186             WRITE(lun_screen, '(a)') ' --- ' // trim(adjustl(message))
00187         ELSE
00188             WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called',
00189             trim(adjustl(messagesources(sourcenum))),
00190             trim(adjustl(message))
00191         END IF
00192     !#ifdef FLUSH_MESSAGES
00193     ! In Fortran >=2003 the call is:
00194     ! FLUSH(LUN_LOG)
00195     CALL flush(lun_screen)
00196     !#endif
00197     END IF
00198
00199 END SUBROUTINE screenmessage_1
00200
00201 SUBROUTINE screenmessage_2(level, message)
00202
00203     IMPLICIT NONE

```

```

00204
00205      ! Global variables
00206      INTEGER, INTENT(IN)      :: level
00207      CHARACTER(LEN=*) , INTENT(IN) :: message
00208
00209      IF (nscreen > 0) THEN
00210        IF (loginitcalled) THEN
00211          WRITE(lun_screen, '(a, " :: ", a, ": ", a)') trim(adjustl(loglevelnames(level + 2))), &
00212                                                    trim(adjustl(messagesources(sourcenum))), &
00213                                                    trim(adjustl(message))
00214        ELSE
00215          WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called', &
00216                                                    trim(adjustl(messagesources(sourcenum))), &
00217                                                    trim(adjustl(message))
00218        END IF
00219      !#ifdef FLUSH_MESSAGES
00220        ! In Fortran >=2003 the call is:
00221        ! FLUSH(LUN_LOG)
00222        CALL flush(lun_screen)
00223      !#endif
00224      END IF
00225
00226      END SUBROUTINE screenmessage_2
00227
00228      !=====
00229
00230      !-----
00231      !      S U B R O U T I N E      L O G   M E S S A G E
00232      !-----
00233      !-----
00244      !-----
00245      SUBROUTINE logmessage_1(message)
00246
00247        IMPLICIT NONE
00248
00249        ! Global variables
00250        CHARACTER(LEN=*) , INTENT(IN) :: message
00251
00252        IF (logfileopened) THEN
00253          IF (loginitcalled) THEN
00254            WRITE(lun_log, '(a)') ' --- ' // trim(adjustl(message))
00255          ELSE
00256            WRITE(lun_log, '(a, " :: ", a, ": ", a)') 'InitLogging not called', &
00257                                                    trim(adjustl(messagesources(sourcenum))), &
00258                                                    trim(adjustl(message))
00259          END IF
00260        !#ifdef FLUSH_MESSAGES
00261          ! In Fortran >=2003 the call is:
00262          ! FLUSH(LUN_LOG)
00263          CALL flush(lun_log)
00264        !#endif
00265        END IF
00266
00267        END SUBROUTINE logmessage_1
00268
00269      SUBROUTINE logmessage_2(level, message)
00270
00271        IMPLICIT NONE
00272
00273        ! Global variables
00274        INTEGER, INTENT(IN)      :: level
00275        CHARACTER(LEN=*) , INTENT(IN) :: message
00276
00277        IF (logfileopened) THEN
00278          IF (loginitcalled) THEN
00279            WRITE(lun_log, '(a, " :: ", a, ": ", a)') trim(adjustl(loglevelnames(level + 2))), &
00280                                                    trim(adjustl(messagesources(sourcenum))), &
00281                                                    trim(adjustl(message))
00282          ELSE
00283            WRITE(lun_log, '(a, " :: ", a, ": ", a)') 'InitLogging not called', &
00284                                                    trim(adjustl(messagesources(sourcenum))), &
00285                                                    trim(adjustl(message))
00286          END IF
00287        !#ifdef FLUSH_MESSAGES
00288          ! In Fortran >=2003 the call is:
00289          ! FLUSH(LUN_LOG)
00290          CALL flush(lun_log)
00291        !#endif
00292        END IF
00293
00294        END SUBROUTINE logmessage_2
00295

```



```

00296 !=====
00297
00298 !-----
00299 !   S U B R O U T I N E   A L L   M E S S A G E
00300 !-----
00308 !-----
00309 SUBROUTINE allmessage_1(message)
00310
00311     IMPLICIT NONE
00312
00313     ! Global variables
00314     CHARACTER(LEN=*) , INTENT(IN) :: message
00315
00316     CALL screenmessage(message)
00317     CALL logmessage(message)
00318
00319 END SUBROUTINE allmessage_1
00320
00321 SUBROUTINE allmessage_2(level, message)
00322
00323     IMPLICIT NONE
00324
00325     ! Global variables
00326     INTEGER, INTENT(IN) :: level
00327     CHARACTER(LEN=*) , INTENT(IN) :: message
00328
00329
00330     CALL screenmessage(level, message)
00331     CALL logmessage(level, message)
00332
00333 END SUBROUTINE allmessage_2
00334
00335 !=====
00336
00337 !-----
00338 !   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
00339 !-----
00348 !-----
00349 SUBROUTINE setmessagesource(source)
00350
00351     IMPLICIT NONE
00352
00353     ! Global variables
00354     CHARACTER(LEN=*) , INTENT(IN) :: source
00355
00356     sourcenumber = sourcenumber + 1
00357     messagesources(sourcenumber) = source
00358
00359 END SUBROUTINE setmessagesource
00360
00361 !=====
00362
00363 !-----
00364 !   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
00365 !-----
00375 !-----
00376 SUBROUTINE unsetmessagesource()
00377
00378     IMPLICIT NONE
00379
00380     sourcenumber = sourcenumber - 1
00381
00382 END SUBROUTINE unsetmessagesource
00383
00384 !=====
00385
00386 !-----
00387 !   S U B R O U T I N E   P R O G R A M   V E R S I O N
00388 !-----
00396 !-----
00397 SUBROUTINE programversion()
00398
00399     USE version
00400
00401     IMPLICIT NONE
00402
00403     WRITE(lun_screen, '(a)') trim(prog_fullname) // ' ' // trim(prog_version) // ' ' // trim(prog_date)
00404 !   WRITE(LUN_SCREEN, '(a)') 'NOAA/NOS/CSDL, Coastal Marine Modeling Branch.'
00405     WRITE(lun_screen, '(a)') ' Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/). '
00406     WRITE(lun_screen, '(a)') ' NOAA/NOS/CSDL (https://nauticalcharts.noaa.gov/). '
00407     WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - This is free software; see the source for copying

```

```

conditions.'
00408     WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - There is NO warranty.'
00409
00410     WRITE(lun_screen, '(a)') "
00411
00412 END SUBROUTINE programversion
00413
00414 !=====
00415
00416 !-----
00417 !   S U B R O U T I N E   P R O G R A M   H E L P
00418 !-----
00426 !-----
00427 SUBROUTINE programhelp()
00428
00429     IMPLICIT NONE
00430
00431     CALL programversion
00432
00433     WRITE(lun_screen, '(a)') 'Help Screen not yet implemented'
00434
00435     WRITE(lun_screen, '(a)') "
00436
00437 END SUBROUTINE programhelp
00438
00439 !=====
00440
00441 !-----
00442 !   S U B R O U T I N E   T E R M I N A T E
00443 !-----
00451 !-----
00452 SUBROUTINE terminate()
00453
00454     USE version
00455
00456     IMPLICIT NONE
00457
00458     CALL setmessagesource("Terminate")
00459
00460     CALL allmessage(error, trim(adjustl(prog_name)) // " Terminating.")
00461
00462     CALL unsetmessagesource()
00463
00464     stop
00465
00466 END SUBROUTINE terminate
00467
00468 !=====
00469
00470 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.](#)

```

00001 !-----
00002 !           M O D U L E   N E T C D F   I O
00003 !-----
00006 !-----
00007
00008 MODULE pahm_netcdfio
00009
00010     USE pahm_sizes
00011     USE pahm_messages
00012     USE pahm_global
00013     USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
00014     USE netcdf
00015
00016 #ifdef __INTEL_COMPILER
00017     USE ifport
00018 #endif
00019
00020     IMPLICIT NONE
00021
00022 #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
00023
00024     INTEGER, PRIVATE                :: ncFormat

```

```

00025 INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
00026 INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)
00027
00028 INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
00029 INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID
00030
00031 TYPE :: filedata_t
00032     LOGICAL :: initialized = .false.
00033     INTEGER :: filereccounter = 0
00034     CHARACTER(LEN=FNAMELEN) :: filename
00035     LOGICAL :: filefound = .false. ! .true. if the netCDF file is present
00036 END TYPE filedata_t
00037
00038 TYPE :: timedata_t
00039     LOGICAL :: initialized = .false.
00040     INTEGER :: timelen = 1 ! number of time slices to write
00041     INTEGER :: timedimid
00042     INTEGER :: timeid
00043     INTEGER :: timedims(1)
00044     REAL(sz), ALLOCATABLE :: time(:)
00045 END TYPE timedata_t
00046
00047 TYPE, PRIVATE :: adcircoorddata_t
00048     LOGICAL :: initialized = .false.
00049     REAL(sz) :: initval
00050     INTEGER :: dimid
00051     INTEGER :: varid
00052     INTEGER :: vardimids
00053     INTEGER :: vardims
00054     CHARACTER(50) :: varname
00055     REAL(sz), ALLOCATABLE :: var(:)
00056     INTEGER :: start(1), count(1)
00057 END TYPE adcircoorddata_t
00058
00059 TYPE, PRIVATE :: adcircvardata_t
00060     LOGICAL :: initialized = .false.
00061     REAL(sz) :: initval
00062     INTEGER :: varid
00063     INTEGER :: vardimids(2)
00064     INTEGER :: vardims(2)
00065     CHARACTER(50) :: varname
00066     REAL(sz), ALLOCATABLE :: var(:, :)
00067     INTEGER :: start(2), count(2)
00068 END TYPE adcircvardata_t
00069
00070 TYPE, PRIVATE :: adcircvardata3d_t
00071     LOGICAL :: initialized = .false.
00072     REAL(sz) :: initval
00073     INTEGER :: varid
00074     INTEGER :: vardimids(3)
00075     INTEGER :: vardims(3)
00076     CHARACTER(50) :: varname
00077     REAL(sz), ALLOCATABLE :: var(:, :, :)
00078     INTEGER :: start(3), count(3)
00079 END TYPE adcircvardata3d_t
00080
00081 TYPE(filedata_t), SAVE :: myfile
00082 TYPE(timedata_t), SAVE :: mytime
00083
00084 TYPE(adcircoorddata_t), PRIVATE, SAVE :: crdtime
00085 TYPE(adcircoorddata_t), PRIVATE, SAVE :: crdlons
00086 TYPE(adcircoorddata_t), PRIVATE, SAVE :: crdlats
00087 TYPE(adcircoorddata_t), PRIVATE, SAVE :: crdxcs
00088 TYPE(adcircoorddata_t), PRIVATE, SAVE :: crdyys
00089
00090 TYPE(adcircvardata_t), PRIVATE, SAVE :: datelements
00091 TYPE(adcircvardata_t), PRIVATE, SAVE :: datatmpres
00092 TYPE(adcircvardata_t), PRIVATE, SAVE :: datwindx
00093 TYPE(adcircvardata_t), PRIVATE, SAVE :: datwindy
00094
00095 CONTAINS
00096
00097
00098
00099 !-----
00100 ! SUBROUTINE INIT ADCIRC NETCDF OUT FILE
00101 !-----
00102 ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00103 !-----
00114 SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)
00115

```

```

00116     USE version
00117     USE timedateutils, ONLY : gettimeconvsec, datetime2string
00118
00119     IMPLICIT NONE
00120
00121     CHARACTER(LEN=*) , INTENT(INOUT) :: adcircOutFile
00122
00123     INTEGER :: ncID
00124     CHARACTER(LEN=64) :: refDateTimeStr, modDateTimeStr, tmpVarName
00125     CHARACTER(LEN=128) :: institution, source, history, comments, host, &
00126                          conventions, contact, references
00127     INTEGER :: tvals(8)
00128     INTEGER :: ierr ! success or failure of a netcdf call
00129
00130
00131     CALL setmessagesource("InitAdcircNetCDFOutFile")
00132
00133     refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units = unittime)
00134
00135     institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
00136     source      = "
00137     history     = "
00138     comments    = "
00139     host        = "
00140     conventions = 'UGRID-0.9.0'
00141     contact     = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
00142     references  = "
00143
00144
00145     ! Create the NetCDF output file.
00146     CALL newadcircnetcdfoutfile(ncid, adcircoutfile)
00147
00148     !=====
00149     !===== (1) Define all the dimensions
00150     !=====
00151     tmpvarname = 'time'
00152     ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
00153     CALL netcdfcheckerr(ierr)
00154
00155     tmpvarname = 'longitude'
00156     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
00157     CALL netcdfcheckerr(ierr)
00158
00159     tmpvarname = 'latitude'
00160     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
00161     CALL netcdfcheckerr(ierr)
00162
00163     tmpvarname = 'node'
00164     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
00165     CALL netcdfcheckerr(ierr)
00166
00167     tmpvarname = 'element'
00168     ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemddimid)
00169     CALL netcdfcheckerr(ierr)
00170
00171     tmpvarname = 'noel'
00172     ierr = nf90_def_dim(ncid, trim(tmpvarname), 3, vertddimid)
00173     CALL netcdfcheckerr(ierr)
00174
00175     tmpvarname = 'mesh'
00176     ierr = nf90_def_dim(ncid, trim(tmpvarname), 1, meshddimid)
00177     CALL netcdfcheckerr(ierr)
00178
00179     !=====
00180     !===== (2) Define all the variables
00181     !=====
00182     !----- Time variable
00183     tmpvarname = 'time'
00184     crdtime%varname = trim(tmpvarname)
00185     crdtime%varDimIDs = crdtime%dimID
00186     crdtime%varDims = SIZE(times, 1)
00187     crdtime%start(1) = 1
00188     crdtime%count(1) = crdtime%varDims
00189
00190     ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
00191     CALL netcdfcheckerr(ierr)
00192     ierr = nf90_put_att(ncid, crdtime%varID, 'long_name', 'model ' // trim(tmpvarname))
00193     CALL netcdfcheckerr(ierr)
00194     ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
00195     CALL netcdfcheckerr(ierr)
00196     ierr = nf90_put_att(ncid, crdtime%varID, 'units', trim(refdatetimestr))

```

```

00197         CALL netcdfcheckerr(ierr)
00198
00199         ALLOCATE(crdtime%var(crdtime%varDims))
00200         crdtime%var = times * gettimeconvsec(unittime, 1)
00201
00202         !----- Longitude variable
00203         tmpvarname = 'longitude'
00204         crdlons%varname = trim(tmpvarname)
00205         crdlons%varDimIDs = nodedimid
00206         ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlons%varDims)
00207         CALL netcdfcheckerr(ierr)
00208         crdlons%start(1) = 1
00209         crdlons%count(1) = crdlons%varDims
00210
00211         ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
00212         CALL netcdfcheckerr(ierr)
00213         ierr = nf90_put_att(ncid, crdlons%varID, 'long_name', trim(tmpvarname))
00214         CALL netcdfcheckerr(ierr)
00215         ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
00216         CALL netcdfcheckerr(ierr)
00217         ierr = nf90_put_att(ncid, crdlons%varID, 'units', 'degrees_east')
00218         CALL netcdfcheckerr(ierr)
00219         ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue', rmissv)
00220         CALL netcdfcheckerr(ierr)
00221         ierr = nf90_put_att(ncid, crdlons%varID, 'positive', 'east')
00222         CALL netcdfcheckerr(ierr)
00223
00224         ALLOCATE(crdlons%var(crdlons%varDims))
00225         crdlons%var = slam
00226
00227         !----- Latitude variable
00228         tmpvarname = 'latitude'
00229         crdlats%varname = trim(tmpvarname)
00230         crdlats%varDimIDs = nodedimid
00231         ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlats%varDims)
00232         CALL netcdfcheckerr(ierr)
00233         crdlats%start(1) = 1
00234         crdlats%count(1) = crdlats%varDims
00235
00236         ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
00237         CALL netcdfcheckerr(ierr)
00238         ierr = nf90_put_att(ncid, crdlats%varID, 'long_name', trim(tmpvarname))
00239         CALL netcdfcheckerr(ierr)
00240         ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
00241         CALL netcdfcheckerr(ierr)
00242         ierr = nf90_put_att(ncid, crdlats%varID, 'units', 'degrees_north')
00243         CALL netcdfcheckerr(ierr)
00244         ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue', rmissv)
00245         CALL netcdfcheckerr(ierr)
00246         ierr = nf90_put_att(ncid, crdlats%varID, 'positive', 'north')
00247         CALL netcdfcheckerr(ierr)
00248
00249         ALLOCATE(crdlats%var(crdlats%varDims))
00250         crdlats%var = sfea
00251
00252         !----- Element variable
00253         tmpvarname = 'tri'
00254         datelements%varname = trim(tmpvarname)
00255         datelements%varDimIDs(1) = vertdimid
00256         datelements%varDimIDs(2) = elemddimid
00257         ierr = nf90_inquire_dimension(ncid, vertdimid, len = datelements%varDims(1))
00258         CALL netcdfcheckerr(ierr)
00259         ierr = nf90_inquire_dimension(ncid, elemddimid, len = datelements%varDims(2))
00260         CALL netcdfcheckerr(ierr)
00261         datelements%start(1) = 1
00262         datelements%count(1) = datelements%varDims(1)
00263         datelements%start(2) = 1
00264         datelements%count(2) = datelements%varDims(2)
00265
00266         ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
00267         CALL netcdfcheckerr(ierr)
00268         ierr = nf90_put_att(ncid, datelements%varID, 'long_name', trim(tmpvarname))
00269         CALL netcdfcheckerr(ierr)
00270         ierr = nf90_put_att(ncid, datelements%varID, 'standard_name', trim(tmpvarname))
00271         CALL netcdfcheckerr(ierr)
00272         ierr = nf90_put_att(ncid, datelements%varID, 'cf_role', 'face_node_connectivity')
00273         CALL netcdfcheckerr(ierr)
00274         ierr = nf90_put_att(ncid, datelements%varID, 'start_index', 1)
00275         CALL netcdfcheckerr(ierr)
00276         ierr = nf90_put_att(ncid, datelements%varID, 'units', 'nondimensional')
00277         CALL netcdfcheckerr(ierr)

```

```

00278     ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',      imissv)
00279     CALL netcdfcheckerr(ierr)
00280
00281     ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
00282     datelements%var = nm
00283
00284     !----- Mesh variable
00285     tmpvarname = 'adcirc_mesh'
00286     ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
00287     CALL netcdfcheckerr(ierr)
00288
00289     ierr = nf90_put_att(ncid, meshvarid, 'long_name',                'mesh_topology')
00290     CALL netcdfcheckerr(ierr)
00291     ierr = nf90_put_att(ncid, meshvarid, 'standard_name',            'mesh_topology')
00292     CALL netcdfcheckerr(ierr)
00293     ierr = nf90_put_att(ncid, meshvarid, 'cf_role',                  'mesh_topology')
00294     CALL netcdfcheckerr(ierr)
00295     ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates',         'lon lat')
00296     CALL netcdfcheckerr(ierr)
00297     ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity',  'element')
00298     CALL netcdfcheckerr(ierr)
00299
00300     !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
00301     tmpvarname = 'projection'
00302     ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
00303     CALL netcdfcheckerr(ierr)
00304
00305     ierr = nf90_put_att(ncid, projvarid, 'long_name',                'equidistant cylindrical projection')
00306     CALL netcdfcheckerr(ierr)
00307     ierr = nf90_put_att(ncid, projvarid, 'standard_name',            'CPP')
00308     CALL netcdfcheckerr(ierr)
00309     ierr = nf90_put_att(ncid, projvarid, 'node_coordinates',         'x y')
00310     CALL netcdfcheckerr(ierr)
00311     ierr = nf90_put_att(ncid, projvarid, 'lon0',                     slam0)
00312     CALL netcdfcheckerr(ierr)
00313     ierr = nf90_put_att(ncid, projvarid, 'lat0',                     sfea0)
00314     CALL netcdfcheckerr(ierr)
00315     ierr = nf90_put_att(ncid, projvarid, 'earth_radius',            rearth)
00316     CALL netcdfcheckerr(ierr)
00317
00318     !----- CPP CPP x-coordinates
00319     tmpvarname = 'x'
00320     crdxcs%varname = trim(tmpvarname)
00321     crdxcs%dimID = nodedimid
00322     crdxcs%varDimIDs = nodedimid
00323     ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
00324     CALL netcdfcheckerr(ierr)
00325     crdxcs%start(1) = 1
00326     crdxcs%count(1) = crdxcs%varDims
00327
00328     ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
00329     CALL netcdfcheckerr(ierr)
00330     ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',              'CPP x coordinate')
00331     CALL netcdfcheckerr(ierr)
00332     ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name',          'cpp_x')
00333     CALL netcdfcheckerr(ierr)
00334     ierr = nf90_put_att(ncid, crdxcs%varID, 'units',                  'm')
00335     CALL netcdfcheckerr(ierr)
00336     ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',            rmissv)
00337     CALL netcdfcheckerr(ierr)
00338
00339     ALLOCATE(crdxcs%var(crdxcs%varDims))
00340     crdxcs%var = xcslam
00341
00342     !----- CPP y-coordinates
00343     tmpvarname = 'y'
00344     crdyics%varname = trim(tmpvarname)
00345     crdyics%dimID = nodedimid
00346     crdyics%varDimIDs = nodedimid
00347     ierr = nf90_inquire_dimension(ncid, crdyics%dimID, len = crdyics%varDims)
00348     CALL netcdfcheckerr(ierr)
00349     crdyics%start(1) = 1
00350     crdyics%count(1) = crdyics%varDims
00351
00352     ierr = nf90_def_var(ncid, trim(crdyics%varname), nf90_double, crdyics%varDimIDs, crdyics%varID)
00353     CALL netcdfcheckerr(ierr)
00354     ierr = nf90_put_att(ncid, crdyics%varID, 'long_name',              'CPP y coordinate')
00355     CALL netcdfcheckerr(ierr)
00356     ierr = nf90_put_att(ncid, crdyics%varID, 'standard_name',          'cpp_y')
00357     CALL netcdfcheckerr(ierr)
00358     ierr = nf90_put_att(ncid, crdyics%varID, 'units',                  'm')

```

```

00359         CALL netcdfcheckerr(ierr)
00360         ierr = nf90_put_att(ncid, crdyics%varID, '_FillValue',      rmissv)
00361         CALL netcdfcheckerr(ierr)
00362
00363         ALLOCATE(crdyics%var(crdyics%varDims))
00364         crdyics%var = ycsfea
00365
00366         !----- Atmospheric Pressure variable
00367         tmpvarname = trim(ncvarnam_pres)
00368         datatmpres%varname = trim(tmpvarname)
00369         datatmpres%varDimIDs(1) = nodedimid
00370         datatmpres%varDimIDs(2) = crdtime%dimID
00371         datatmpres%varDims(1) = SIZE(wpress, 1)
00372         datatmpres%varDims(2) = SIZE(wpress, 2)
00373         datatmpres%start(1) = 1
00374         datatmpres%count(1) = datatmpres%varDims(1)
00375         datatmpres%start(2) = 1
00376         datatmpres%count(2) = datatmpres%varDims(2)
00377
00378         ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
00379                             datatmpres%varDimIDs, datatmpres%varID)
00380         CALL netcdfcheckerr(ierr)
00381         ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',      'air pressure at sea level')
00382         CALL netcdfcheckerr(ierr)
00383         ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
00384         CALL netcdfcheckerr(ierr)
00385         ierr = nf90_put_att(ncid, datatmpres%varID, 'units',          'Pa')
00386         CALL netcdfcheckerr(ierr)
00387         ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
00388         CALL netcdfcheckerr(ierr)
00389         ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',    'time lat lon')
00390         CALL netcdfcheckerr(ierr)
00391         ierr = nf90_put_att(ncid, datatmpres%varID, 'location',       'node')
00392         CALL netcdfcheckerr(ierr)
00393         ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',          'adcirc_mesh')
00394         CALL netcdfcheckerr(ierr)
00395
00396         ALLOCATE(datatmpres%var(datatmpres%varDims(1), datatmpres%varDims(2)))
00397         datatmpres%var = wpress
00398
00399         !----- Wind velocity variables
00400         ! Eastward
00401         tmpvarname = trim(ncvarnam_wndx)
00402         datwindx%varname = trim(tmpvarname)
00403         datwindx%varDimIDs(1) = nodedimid
00404         datwindx%varDimIDs(2) = crdtime%dimID
00405         datwindx%varDims(1) = SIZE(wvelx, 1)
00406         datwindx%varDims(2) = SIZE(wvelx, 2)
00407         datwindx%start(1) = 1
00408         datwindx%count(1) = datwindx%varDims(1)
00409         datwindx%start(2) = 1
00410         datwindx%count(2) = datwindx%varDims(2)
00411
00412         ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
00413                             datwindx%varDimIDs, datwindx%varID)
00414         CALL netcdfcheckerr(ierr)
00415         ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',      '10-m eastward wind component')
00416         CALL netcdfcheckerr(ierr)
00417         ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
00418         CALL netcdfcheckerr(ierr)
00419         ierr = nf90_put_att(ncid, datwindx%varID, 'units',          'm s-1')
00420         CALL netcdfcheckerr(ierr)
00421         ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',    rmissv)
00422         CALL netcdfcheckerr(ierr)
00423         ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates',    'time lat lon')
00424         CALL netcdfcheckerr(ierr)
00425         ierr = nf90_put_att(ncid, datwindx%varID, 'location',       'node')
00426         CALL netcdfcheckerr(ierr)
00427         ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',          'adcirc_mesh')
00428         CALL netcdfcheckerr(ierr)
00429
00430         ALLOCATE(datwindx%var(datwindx%varDims(1), datwindx%varDims(2)))
00431         datwindx%var = wvelx
00432
00433         ! Northward
00434         tmpvarname = trim(ncvarnam_wndy)
00435         datwindy%varname = trim(tmpvarname)
00436         datwindy%varDimIDs(1) = nodedimid
00437         datwindy%varDimIDs(2) = crdtime%dimID
00438         datwindy%varDims(1) = SIZE(wvely, 1)
00439         datwindy%varDims(2) = SIZE(wvely, 2)

```



```

00440      datwindy%start(1)      = 1
00441      datwindy%count(1)      = datwindy%varDims(1)
00442      datwindy%start(2)      = 1
00443      datwindy%count(2)      = datwindy%varDims(2)
00444
00445      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
00446                        datwindy%varDimIDs, datwindy%varID)
00447      CALL netcdfcheckerr(ierr)
00448      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',      '10-m northward wind component')
00449      CALL netcdfcheckerr(ierr)
00450      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
00451      CALL netcdfcheckerr(ierr)
00452      ierr = nf90_put_att(ncid, datwindy%varID, 'units',          'm s-1')
00453      CALL netcdfcheckerr(ierr)
00454      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',     rmissv)
00455      CALL netcdfcheckerr(ierr)
00456      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates',    'time lat lon')
00457      CALL netcdfcheckerr(ierr)
00458      ierr = nf90_put_att(ncid, datwindy%varID, 'location',       'node')
00459      CALL netcdfcheckerr(ierr)
00460      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',           'adcirc_mesh')
00461      CALL netcdfcheckerr(ierr)
00462
00463      ALLOCATE(datwindy%var(datwindy%varDims(1), datwindy%varDims(2)))
00464      datwindy%var = wvely
00465
00466      !=====
00467      !==== (3) Set Deflate parameters if requested by the user
00468      !=====
00469      #ifdef NETCDF_CAN_DEFLATE
00470      IF (ncformat == nc4form) THEN
00471          ierr = nf90_def_var_deflate(ncid, crdlons%varID,      ncshuffle, ncdeflate, ncdlevel)
00472          CALL netcdfcheckerr(ierr)
00473          ierr = nf90_def_var_deflate(ncid, crdlats%varID,      ncshuffle, ncdeflate, ncdlevel)
00474          CALL netcdfcheckerr(ierr)
00475          ierr = nf90_def_var_deflate(ncid, crdxcs%varID,      ncshuffle, ncdeflate, ncdlevel)
00476          CALL netcdfcheckerr(ierr)
00477          ierr = nf90_def_var_deflate(ncid, crdycs%varID,      ncshuffle, ncdeflate, ncdlevel)
00478          CALL netcdfcheckerr(ierr)
00479          ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
00480          CALL netcdfcheckerr(ierr)
00481          ierr = nf90_def_var_deflate(ncid, datatmpres%varID,  ncshuffle, ncdeflate, ncdlevel)
00482          CALL netcdfcheckerr(ierr)
00483          ierr = nf90_def_var_deflate(ncid, datwindx%varID,    ncshuffle, ncdeflate, ncdlevel)
00484          CALL netcdfcheckerr(ierr)
00485          ierr = nf90_def_var_deflate(ncid, datwindy%varID,    ncshuffle, ncdeflate, ncdlevel)
00486          CALL netcdfcheckerr(ierr)
00487      END IF
00488      #endif
00489
00490      !=====
00491      !==== (4) Global metadata definitions and variables
00492      !=====
00493      ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
00494      CALL netcdfcheckerr(ierr)
00495      ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
00496      ')')
00497      CALL netcdfcheckerr(ierr)
00498      ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
00499      CALL netcdfcheckerr(ierr)
00500      ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
00501      CALL netcdfcheckerr(ierr)
00502      ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
00503      CALL netcdfcheckerr(ierr)
00504      ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
00505      CALL netcdfcheckerr(ierr)
00506      ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
00507      CALL netcdfcheckerr(ierr)
00508      ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
00509      CALL netcdfcheckerr(ierr)
00510      ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
00511      CALL netcdfcheckerr(ierr)
00512      ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
00513      CALL netcdfcheckerr(ierr)
00514      ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
00515      CALL netcdfcheckerr(ierr)
00516      ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
00517      CALL netcdfcheckerr(ierr)
00518      ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
00519      CALL netcdfcheckerr(ierr)

```

```

00520     CALL date_and_time(values = tvals)
00521     WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
00522     moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
moddatetimestr)
00523
00524     ierr = nf90_put_att(ncid, nf90_global, 'creation_date', trim(moddatetimestr))
00525     CALL netcdfcheckerr(ierr)
00526     ierr = nf90_put_att(ncid, nf90_global, 'modification_date', trim(moddatetimestr))
00527     CALL netcdfcheckerr(ierr)
00528
00529     !----- Finalize the definitions in the NetCDF file
00530     ierr = nf90_enddef(ncid)
00531     CALL netcdfcheckerr(ierr)
00532
00533     !=====
00534     !===== (5) Put the static data into the NetCDF file and then close it
00535     !=====
00536     ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
00537     CALL netcdfcheckerr(ierr)
00538
00539     ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
00540     CALL netcdfcheckerr(ierr)
00541
00542     ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
00543     CALL netcdfcheckerr(ierr)
00544
00545     ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
00546     CALL netcdfcheckerr(ierr)
00547
00548     ierr = nf90_put_var(ncid, crdyics%varID, crdyics%var, crdyics%start, crdyics%count)
00549     CALL netcdfcheckerr(ierr)
00550
00551     ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00552     CALL netcdfcheckerr(ierr)
00553
00554     ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00555     CALL netcdfcheckerr(ierr)
00556
00557     ierr = nf90_put_var(ncid, datatmpres%varID, datatmpres%var, datatmpres%start, datatmpres%count)
00558     CALL netcdfcheckerr(ierr)
00559
00560     ierr = nf90_put_var(ncid, datwindx%varID, datwindx%var, datwindx%start, datwindx%count)
00561     CALL netcdfcheckerr(ierr)
00562
00563     ierr = nf90_put_var(ncid, datwindy%varID, datwindy%var, datwindy%start, datwindy%count)
00564     CALL netcdfcheckerr(ierr)
00565
00566
00567     !----- (16) Set all the "initialized" flags to .TRUE.
00568     crdlons%initialized = .true.
00569     crdlats%initialized = .true.
00570     crdxcs%initialized = .true.
00571     crdyics%initialized = .true.
00572     datelements%initialized = .true.
00573     datatmpres%initialized = .true.
00574     datwindx%initialized = .true.
00575     datwindy%initialized = .true.
00576
00577     myfile%fileName = adcircoutfile
00578     myfile%initialized = .true.
00579
00580     !----- Close the NetCDF file
00581     ierr = nf90_close(ncid)
00582     CALL netcdfcheckerr(ierr)
00583
00584     CALL unsetmessagesource()
00585
00586     END SUBROUTINE initadcircnetcdfoutfile
00587
00588     !=====
00589
00590     !-----
00591     ! SUBROUTINE NEW ADCIRC NETCDF OUT FILE
00592     !-----
00593     ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00594     !-----
00606
00607     SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)
00608
00609     IMPLICIT NONE
00610
00611     INTEGER, INTENT(OUT) :: ncID

```

```

00612     CHARACTER(LEN=*) , INTENT(INOUT)    :: adcircOutFile
00613
00614     LOGICAL                                :: fileFound = .false.
00615     CHARACTER(LEN=FNAMELEN)                :: outFile, sys_cmd
00616     CHARACTER(LEN=14)                     :: fext, date_time
00617     INTEGER                                :: pos, ierr, tvals(8)
00618
00619
00620     CALL setmessagesource("NewAdcircNetCDFOutFile")
00621
00622     !-----
00623     ! Set some variables that depend upon the type of NetCDF supported.
00624 #if defined(HAVE_NETCDF4)
00625     fext = ".nc4"
00626     ncformat = nc4form
00627 #else
00628     fext = ".nc"
00629     ncformat = nc3form
00630 #endif
00631
00632     !-----
00633     ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
00634     ! extension in the filename; re-define the adcircOutFile variable.
00635     pos = scan(trim(adcircoutfile), ".", back=.true.)
00636     IF (pos > 0) THEN
00637         adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
00638     ELSE
00639         adcircoutfile = trim(adcircoutfile) // trim(fext)
00640     END IF
00641
00642     !-----
00643     ! If the adcircOutFile exists then rename it to:
00644     !   adcircOutFile-YYYYMMDDhhmmss.
00645     ! The user can remove these files afterwards.
00646     INQUIRE(file=adcircoutfile, exist=filefound)
00647     IF (filefound) THEN
00648         CALL date_and_time(values = tvals)
00649         WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
00650         outfile = trim(adcircoutfile) // "-" // trim(date_time)
00651         sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
00652         ierr = system(trim(sys_cmd))
00653         IF (ierr == 0) THEN
00654             WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
00655             CALL logmessage(info, scratchmessage)
00656             filefound = .false.
00657         ELSE
00658             WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' //
trim(outfile)
00659             CALL logmessage(error, scratchmessage)
00660         END IF
00661     END IF
00662
00663     IF (filefound) THEN
00664         WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adcircoutfile) // ' exists. Remove the
file to proceed.'
00665         CALL allmessage(error, scratchmessage)
00666
00667         CALL unsetmessagesource()
00668
00669         CALL netcdfterminate
00670     END IF
00671
00672     WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adcircoutfile) // ' and putting it in define
mode.'
00673     CALL logmessage(info, scratchmessage)
00674
00675     ! Create the NetCDF file
00676     ierr = nf90_create(adcircoutfile, ncformat, ncid)
00677     CALL netcdfcheckerr(ierr)
00678
00679     CALL unsetmessagesource()
00680
00681 END SUBROUTINE newadcircnetcdfoutfile
00682
00683 !=====
00684
00685 !-----
00686 ! SUBROUTINE NETCDF CHECK ERR
00687 !-----
00688 !
00689 !-----

```

```

00697 SUBROUTINE base_netcdfcheckerr(ierr, file, line)
00698
00699     IMPLICIT NONE
00700
00701     INTEGER, INTENT(IN)          :: ierr
00702     CHARACTER(LEN=*), INTENT(IN) :: file
00703     INTEGER, INTENT(IN)          :: line
00704
00705     CHARACTER(LEN=1024)          :: tmpSTR
00706
00707     CALL setmessagesource("NetCDFCheckErr")
00708
00709     IF (ierr /= nf90_noerr) THEN
00710         CALL allmessage(error, nf90_strerror(ierr))
00711         WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
00712         CALL allmessage(info, tmpstr)
00713         CALL netcdfterminate()
00714     END IF
00715
00716     CALL unsetmessagesource()
00717
00718 END SUBROUTINE base_netcdfcheckerr
00719
00720 !=====
00721 !-----
00722 ! SUBROUTINE NETCDF TERMINATE
00723 !-----
00724 !
00725 !-----
00726 !
00727 SUBROUTINE netcdfterminate()
00728
00729     USE version
00730
00731     IMPLICIT NONE
00732
00733     CALL setmessagesource("NetCDFTerminate")
00734
00735     CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")
00736
00737     CALL exit(1)
00738
00739     CALL unsetmessagesource()
00740
00741 END SUBROUTINE netcdfterminate
00742
00743 !=====
00744 !-----
00745 !
00746 ! SUBROUTINE WRITE NETCDF RECORD
00747 !-----
00748 !
00749 ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00750 !-----
00751 !
00752 SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)
00753
00754     IMPLICIT NONE
00755
00756     CHARACTER(LEN=*), INTENT(IN) :: adcircOutFile
00757
00758     INTEGER :: timeLoc
00759     INTEGER :: ncID, ierr, nodes
00760     INTEGER :: start(2), kount(2)
00761
00762     CALL setmessagesource("WriteNetCDFRecord")
00763
00764     ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
00765     CALL netcdfcheckerr(ierr)
00766
00767     ! Set up the 2D netcdf data extents
00768     ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
00769     start(1) = 1
00770     start(2) = myfile%fileRecCounter
00771     kount(1) = nodes
00772     kount(2) = mytime%timeLen
00773
00774     ierr = nf90_put_var(ncid, datatmpres%varID, wpress(:, timeloc), start, kount)
00775     CALL netcdfcheckerr(ierr)
00776
00777     ierr = nf90_put_var(ncid, datwindx%varID, wvelx(:, timeloc), start, kount)
00778     CALL netcdfcheckerr(ierr)
00779

```

```

00790
00791     ierr = nf90_put_var(ncid, datwindy%varID, wvely(:, timeloc), start, kount)
00792     CALL netcdfcheckerr(ierr)
00793
00794     ! Close netCDF file
00795     ierr = nf90_close(ncid)
00796     CALL netcdfcheckerr(ierr)
00797
00798     DEALLOCATE(mytime%time)
00799
00800     CALL unsetmessagesource()
00801
00802 END SUBROUTINE writenetcdfrecord
00803
00804 !=====
00805
00806 !-----
00807 ! SUBROUTINE SET RECORD COUNTER AND STORE TIME
00808 !-----
00809 ! author
00810 !-----
00811
00812 SUBROUTINE setrecordcounterandstoretime(ncID, f, t)
00813
00814     IMPLICIT NONE
00815
00816     INTEGER, INTENT(IN)          :: ncID
00817     TYPE(filedata_t), INTENT(INOUT) :: f
00818     TYPE(timedata_t), INTENT(INOUT) :: t
00819
00820     REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
00821     LOGICAL                :: timeFound      ! true if current time is in array of stored times
00822
00823     INTEGER :: ndim      ! number of dimensions in the netcdf file
00824     INTEGER :: nvar      ! number of variables in the netcdf file
00825     INTEGER :: natt      ! number of attributes in the netcdf file
00826
00827     INTEGER :: counti(1), starti(1)
00828     INTEGER :: ierr ! success or failure of netcdf call
00829     INTEGER :: i    ! loop counter
00830
00831     CALL setmessagesource("SetRecordCounterAndStoreTime")
00832
00833     ! Inquire the time variable
00834     ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
00835     CALL netcdfcheckerr(ierr)
00836
00837     ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
00838     CALL netcdfcheckerr(ierr)
00839
00840     ierr = nf90_inq_varid(ncid, 'time', t%timeID)
00841     CALL netcdfcheckerr(ierr)
00842
00843     ! Determine the relationship between the current simulation time
00844     ! and the time array stored in the netcdf file. Set the record
00845     ! counter based on this relationship.
00846     IF (f%fileRecCounter /= 0) THEN
00847         ALLOCATE(storedtimes(f%fileRecCounter))
00848         ierr = nf90_get_var(ncid, t%timeID, storedtimes)
00849         CALL netcdfcheckerr(ierr)
00850         timefound = .false.
00851
00852         DO i = 1, f%fileRecCounter
00853             IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
00854                 timefound = .true.
00855                 EXIT
00856             ENDIF
00857         END DO
00858
00859         IF (timefound .EQV. .false.) THEN
00860             ! Increment the record counter so that we can store data at the
00861             ! next location in the netcdf file (i.e., all of the times
00862             ! in the netcdf file were found to be earlier than the current
00863             ! adcirc simulation time).
00864             f%fileRecCounter = f%fileRecCounter + 1
00865         ELSE
00866             ! set the counter at the index that reflects the
00867             ! current time within the netcdf file (or is between two times
00868             ! found in the netcdf file).
00869             ! WARNING: all subsequent data will remain in the file, we
00870             ! are just overwriting it ... if we don't overwrite all of it,

```

```

00886      ! the pre-existing data will still be there, which is probably
00887      ! not what the user intended ... but apparently there is no
00888      ! way to delete data from netcdf files:
00889      ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
00890      scratchformat = ' ("Overwriting pre-existing data in netcdf file ",a,' // &
00891      " " for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.')"
00892      WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
00893      CALL allmessage(info, scratchmessage)
00894      f%fileRecCounter = i
00895  ENDF
00896
00897      DEALLOCATE(storedtimes)
00898  ELSE
00899      ! set the counter at 1 so we can record our first time value
00900      f%fileRecCounter = 1
00901  ENDF
00902
00903      ! Store simulation time in netcdf file
00904      starti(1) = f%fileRecCounter
00905      counti(1) = t%timeLen
00906      ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
00907      CALL netcdfcheckerr(ierr)
00908
00909      CALL unsetmessagesource()
00910
00911  END SUBROUTINE setrecordcounterandstoretime
00912
00913  !=====
00914
00915  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.](#)

```

00001 !-----
00002 !           M O D U L E   N E T C D F   I O
00003 !-----
00006 !-----
00007
00008 MODULE pahm_netcdfio
00009
00010     USE pahm_sizes
00011     USE pahm_messages
00012     USE pahm_global
00013     USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
00014     USE netcdf
00015
00016 #ifdef __INTEL_COMPILER
00017     USE ifport
00018 #endif
00019
00020     IMPLICIT NONE
00021
00022 #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
00023
00024     INTEGER, PRIVATE :: ncFormat
00025     INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
00026     INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)
00027
00028     INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
00029     INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID
00030
00031     TYPE :: filedata_t
00032         LOGICAL :: initialized = .false.
00033         INTEGER :: fileRecCounter = 0
00034         CHARACTER(LEN=FNAMELEN) :: fileName
00035         LOGICAL :: fileFound = .false. ! .true. if the netCDF file is present
00036     END TYPE filedata_t
00037
00038     TYPE :: timedata_t
00039         LOGICAL :: initialized = .false.
00040         INTEGER :: timeLen = 1 ! number of time slices to write
00041         INTEGER :: timeDimID
00042         INTEGER :: timeID
00043         INTEGER :: timeDims(1)
00044         REAL(SZ), ALLOCATABLE :: time(:)
00045     END TYPE timedata_t
00046
00047     TYPE, PRIVATE :: adircCOORDdata_t
00048         LOGICAL :: initialized = .false.
00049         REAL(SZ) :: initVal
00050         INTEGER :: dimID
00051         INTEGER :: varID
00052         INTEGER :: varDimIDs
00053         INTEGER :: varDims
00054         CHARACTER(50) :: varname
00055         REAL(SZ), ALLOCATABLE :: var(:)
00056         INTEGER :: start(1), count(1)
00057     END TYPE adircCOORDdata_t
00058
00059     TYPE, PRIVATE :: adircVARdata_t
00060         LOGICAL :: initialized = .false.
00061         REAL(SZ) :: initVal
00062         INTEGER :: varID
00063         INTEGER :: varDimIDs(2)

```

```

00064     INTEGER                :: varDims(2)
00065     CHARACTER(50)           :: varname
00066     REAL(SZ), ALLOCATABLE :: var(:, :)
00067     INTEGER                  :: start(2), count(2)
00068 END TYPE adcircvardata_t
00069
00070 TYPE, PRIVATE :: adcircvardata3d_t
00071     LOGICAL                :: initialized = .false.
00072     REAL(SZ)                :: initVal
00073     INTEGER                 :: varID
00074     INTEGER                 :: varDimIDs(3)
00075     INTEGER                 :: varDims(3)
00076     CHARACTER(50)           :: varname
00077     REAL(SZ), ALLOCATABLE :: var(:, :, :)
00078     INTEGER                  :: start(3), count(3)
00079 END TYPE adcircvardata3d_t
00080
00081 TYPE(FileData_T), SAVE      :: myFile
00082 TYPE(TimeData_T), SAVE      :: myTime
00083
00084 TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdTime
00085 TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLons
00086 TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLats
00087 TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdXCs
00088 TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdYCs
00089
00090 TYPE(AdcircVarData_T), PRIVATE, SAVE :: datElements
00091 TYPE(AdcircVarData_T), PRIVATE, SAVE :: datAtmPres
00092 TYPE(AdcircVarData_T), PRIVATE, SAVE :: datWindX
00093 TYPE(AdcircVarData_T), PRIVATE, SAVE :: datWindY
00094
00095 CONTAINS
00096
00097 !-----
00098 ! SUBROUTINE INIT ADCIRC NETCDF OUTPUT FILE
00099 !-----
00100 ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00101 !-----
00102 SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)
00103
00104     USE version
00105     USE timedateutils, ONLY : gettimeconvsec, datetime2string
00106
00107     IMPLICIT NONE
00108
00109     CHARACTER(LEN=*) , INTENT(INOUT) :: adcircOutFile
00110
00111     INTEGER                :: ncID
00112     CHARACTER(LEN=64)       :: refDateTimeStr, modDateTimeStr, tmpVarName
00113     CHARACTER(LEN=128)      :: institution, source, history, comments, host, &
00114                               conventions, contact, references
00115     INTEGER                 :: tvals(8)
00116     INTEGER                 :: ierr ! success or failure of a netcdf call
00117
00118     CALL setmessagesource("InitAdcircNetCDFOutFile")
00119
00120     refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units = unittime)
00121
00122     institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
00123     source      = "
00124     history     = "
00125     comments    = "
00126     host       = "
00127     conventions = 'UGRID-0.9.0'
00128     contact    = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
00129     references  = "
00130
00131     ! Create the NetCDF output file.
00132     CALL newadcircnetcdfoutfile(ncid, adcircoutfile)
00133
00134     !=====
00135     !===== (1) Define all the dimensions
00136     !=====
00137     tmpvarname = 'time'
00138     ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
00139     CALL netcdfcheckerr(ierr)
00140

```



```

00155     tmpvarname = 'longitude'
00156     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
00157     CALL netcdfcheckerr(ierr)
00158
00159     tmpvarname = 'latitude'
00160     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
00161     CALL netcdfcheckerr(ierr)
00162
00163     tmpvarname = 'node'
00164     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
00165     CALL netcdfcheckerr(ierr)
00166
00167     tmpvarname = 'nele'
00168     ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemddimid)
00169     CALL netcdfcheckerr(ierr)
00170
00171     tmpvarname = 'nvertex'
00172     ierr = nf90_def_dim(ncid, trim(tmpvarname), 3, vertddimid)
00173     CALL netcdfcheckerr(ierr)
00174
00175     tmpvarname = 'mesh'
00176     ierr = nf90_def_dim(ncid, trim(tmpvarname), 1, meshddimid)
00177     CALL netcdfcheckerr(ierr)
00178
00179     !=====
00180     !==== (2) Define all the variables
00181     !=====
00182     !----- Time variable
00183     tmpvarname = 'time'
00184     crdtime%varname = trim(tmpvarname)
00185     crdtime%varDimIDs = crdtime%dimID
00186     crdtime%varDims = SIZE(times, 1)
00187     crdtime%start(1) = 1
00188     crdtime%count(1) = crdtime%varDims
00189
00190     ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
00191     CALL netcdfcheckerr(ierr)
00192     ierr = nf90_put_att(ncid, crdtime%varID, 'long_name', 'model ' // trim(tmpvarname))
00193     CALL netcdfcheckerr(ierr)
00194     ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
00195     CALL netcdfcheckerr(ierr)
00196     ierr = nf90_put_att(ncid, crdtime%varID, 'units', trim(refdatetimestr))
00197     CALL netcdfcheckerr(ierr)
00198
00199     ALLOCATE(crdtime%var(crdtime%varDims))
00200     crdtime%var = times * gettimeconvsec(unittime, 1)
00201
00202     !----- Longitude variable
00203     tmpvarname = 'longitude'
00204     crdlons%varname = trim(tmpvarname)
00205     crdlons%varDimIDs = crdlons%dimID
00206     ierr = nf90_inquire_dimension(ncid, crdlons%dimID, len = crdlons%varDims)
00207     CALL netcdfcheckerr(ierr)
00208     crdlons%start(1) = 1
00209     crdlons%count(1) = crdlons%varDims
00210
00211     ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
00212     CALL netcdfcheckerr(ierr)
00213     ierr = nf90_put_att(ncid, crdlons%varID, 'long_name', trim(tmpvarname))
00214     CALL netcdfcheckerr(ierr)
00215     ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
00216     CALL netcdfcheckerr(ierr)
00217     ierr = nf90_put_att(ncid, crdlons%varID, 'units', 'degrees_east')
00218     CALL netcdfcheckerr(ierr)
00219     ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue', rmissv)
00220     CALL netcdfcheckerr(ierr)
00221     ierr = nf90_put_att(ncid, crdlons%varID, 'positive', 'east')
00222     CALL netcdfcheckerr(ierr)
00223
00224     ALLOCATE(crdlons%var(crdlons%varDims))
00225     crdlons%var = slam
00226
00227     !----- Latitude variable
00228     tmpvarname = 'latitude'
00229     crdlats%varname = trim(tmpvarname)
00230     crdlats%varDimIDs = crdlats%dimID
00231     ierr = nf90_inquire_dimension(ncid, crdlats%dimID, len = crdlats%varDims)
00232     CALL netcdfcheckerr(ierr)
00233     crdlats%start(1) = 1
00234     crdlats%count(1) = crdlats%varDims
00235

```

```

00236     ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
00237     CALL netcdfcheckerr(ierr)
00238     ierr = nf90_put_att(ncid, crdlats%varID, 'long_name',      trim(tmpvarname))
00239     CALL netcdfcheckerr(ierr)
00240     ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
00241     CALL netcdfcheckerr(ierr)
00242     ierr = nf90_put_att(ncid, crdlats%varID, 'units',          'degrees_north')
00243     CALL netcdfcheckerr(ierr)
00244     ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',     rmissv)
00245     CALL netcdfcheckerr(ierr)
00246     ierr = nf90_put_att(ncid, crdlats%varID, 'positive',       'north')
00247     CALL netcdfcheckerr(ierr)
00248
00249     ALLOCATE(crdlats%var(crdlats%varDims))
00250     crdlats%var = sfea
00251
00252     !----- Element variable
00253     tmpvarname = 'element'
00254     datelements%varname = trim(tmpvarname)
00255     datelements%varDimIDs(1) = vertdimid
00256     datelements%varDimIDs(2) = elemdimid
00257     ierr = nf90_inquire_dimension(ncid, vertdimid, len = datelements%varDims(1))
00258     CALL netcdfcheckerr(ierr)
00259     ierr = nf90_inquire_dimension(ncid, elemdimid, len = datelements%varDims(2))
00260     CALL netcdfcheckerr(ierr)
00261     datelements%start(1) = 1
00262     datelements%count(1) = datelements%varDims(1)
00263     datelements%start(2) = 1
00264     datelements%count(2) = datelements%varDims(2)
00265
00266     ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
00267     CALL netcdfcheckerr(ierr)
00268     ierr = nf90_put_att(ncid, datelements%varID, 'long_name',      trim(tmpvarname))
00269     CALL netcdfcheckerr(ierr)
00270     ierr = nf90_put_att(ncid, datelements%varID, 'standard_name', trim(tmpvarname))
00271     CALL netcdfcheckerr(ierr)
00272     ierr = nf90_put_att(ncid, datelements%varID, 'cf_role',        'face_node_connectivity')
00273     CALL netcdfcheckerr(ierr)
00274     ierr = nf90_put_att(ncid, datelements%varID, 'start_index',    1)
00275     CALL netcdfcheckerr(ierr)
00276     ierr = nf90_put_att(ncid, datelements%varID, 'units',          'nondimensional')
00277     CALL netcdfcheckerr(ierr)
00278     ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',     imissv)
00279     CALL netcdfcheckerr(ierr)
00280
00281     ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
00282     datelements%var = nm
00283
00284     !----- Mesh variable
00285     tmpvarname = 'addirc_mesh'
00286     ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
00287     CALL netcdfcheckerr(ierr)
00288
00289     ierr = nf90_put_att(ncid, meshvarid, 'long_name',              'mesh_topology')
00290     CALL netcdfcheckerr(ierr)
00291     ierr = nf90_put_att(ncid, meshvarid, 'standard_name',          'mesh_topology')
00292     CALL netcdfcheckerr(ierr)
00293     ierr = nf90_put_att(ncid, meshvarid, 'cf_role',                'mesh_topology')
00294     CALL netcdfcheckerr(ierr)
00295     ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates',       'lon lat')
00296     CALL netcdfcheckerr(ierr)
00297     ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
00298     CALL netcdfcheckerr(ierr)
00299
00300     !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
00301     tmpvarname = 'projection'
00302     ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
00303     CALL netcdfcheckerr(ierr)
00304
00305     ierr = nf90_put_att(ncid, projvarid, 'long_name',              'equidistant cylindrical projection')
00306     CALL netcdfcheckerr(ierr)
00307     ierr = nf90_put_att(ncid, projvarid, 'standard_name',          'CPP')
00308     CALL netcdfcheckerr(ierr)
00309     ierr = nf90_put_att(ncid, projvarid, 'node_coordinates',       'x y')
00310     CALL netcdfcheckerr(ierr)
00311     ierr = nf90_put_att(ncid, projvarid, 'lon0',                   slam0)
00312     CALL netcdfcheckerr(ierr)
00313     ierr = nf90_put_att(ncid, projvarid, 'lat0',                   sfea0)
00314     CALL netcdfcheckerr(ierr)
00315     ierr = nf90_put_att(ncid, projvarid, 'earth_radius',          rearth)
00316     CALL netcdfcheckerr(ierr)

```

```

00317
00318 !----- CPP CPP x-coordinates
00319 tmpvarname = 'x'
00320 crdxcs%varname = trim(tmpvarname)
00321 crdxcs%dimID = nodedimid
00322 crdxcs%varDimIDs = nodedimid
00323 ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
00324 CALL netcdfcheckerr(ierr)
00325 crdxcs%start(1) = 1
00326 crdxcs%count(1) = crdxcs%varDims
00327
00328 ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
00329 CALL netcdfcheckerr(ierr)
00330 ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name', 'CPP x coordinate')
00331 CALL netcdfcheckerr(ierr)
00332 ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
00333 CALL netcdfcheckerr(ierr)
00334 ierr = nf90_put_att(ncid, crdxcs%varID, 'units', 'm')
00335 CALL netcdfcheckerr(ierr)
00336 ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue', rmissv)
00337 CALL netcdfcheckerr(ierr)
00338
00339 ALLOCATE(crdxcs%var(crdxcs%varDims))
00340 crdxcs%var = xcslam
00341
00342 !----- CPP y-coordinates
00343 tmpvarname = 'y'
00344 crdyics%varname = trim(tmpvarname)
00345 crdyics%dimID = nodedimid
00346 crdyics%varDimIDs = nodedimid
00347 ierr = nf90_inquire_dimension(ncid, crdyics%dimID, len = crdyics%varDims)
00348 CALL netcdfcheckerr(ierr)
00349 crdyics%start(1) = 1
00350 crdyics%count(1) = crdyics%varDims
00351
00352 ierr = nf90_def_var(ncid, trim(crdyics%varname), nf90_double, crdyics%varDimIDs, crdyics%varID)
00353 CALL netcdfcheckerr(ierr)
00354 ierr = nf90_put_att(ncid, crdyics%varID, 'long_name', 'CPP y coordinate')
00355 CALL netcdfcheckerr(ierr)
00356 ierr = nf90_put_att(ncid, crdyics%varID, 'standard_name', 'cpp_y')
00357 CALL netcdfcheckerr(ierr)
00358 ierr = nf90_put_att(ncid, crdyics%varID, 'units', 'm')
00359 CALL netcdfcheckerr(ierr)
00360 ierr = nf90_put_att(ncid, crdyics%varID, '_FillValue', rmissv)
00361 CALL netcdfcheckerr(ierr)
00362
00363 ALLOCATE(crdyics%var(crdyics%varDims))
00364 crdyics%var = ycsfea
00365
00366 !----- Atmospheric Pressure variable
00367 tmpvarname = trim(ncvarnam_pres)
00368 datatmpres%varname = trim(tmpvarname)
00369 datatmpres%varDimIDs(1) = nodedimid
00370 datatmpres%varDimIDs(2) = crdtime%dimID
00371 datatmpres%varDims(1) = SIZE(wpress, 1)
00372 datatmpres%varDims(2) = SIZE(wpress, 2)
00373 datatmpres%start(1) = 1
00374 datatmpres%count(1) = datatmpres%varDims(1)
00375 datatmpres%start(2) = 1
00376 datatmpres%count(2) = datatmpres%varDims(2)
00377
00378 ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
00379 datatmpres%varDimIDs, datatmpres%varID)
00380 CALL netcdfcheckerr(ierr)
00381 ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name', 'air pressure at sea level')
00382 CALL netcdfcheckerr(ierr)
00383 ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
00384 CALL netcdfcheckerr(ierr)
00385 ierr = nf90_put_att(ncid, datatmpres%varID, 'units', 'Pa')
00386 CALL netcdfcheckerr(ierr)
00387 ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue', rmissv)
00388 CALL netcdfcheckerr(ierr)
00389 ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates', 'time lat lon')
00390 CALL netcdfcheckerr(ierr)
00391 ierr = nf90_put_att(ncid, datatmpres%varID, 'location', 'node')
00392 CALL netcdfcheckerr(ierr)
00393 ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh', 'adcirc_mesh')
00394 CALL netcdfcheckerr(ierr)
00395
00396 ALLOCATE(datatmpres%var(datatmpres%varDims(1), datatmpres%varDims(2)))
00397 datatmpres%var = wpress

```

```

00398
00399      !----- Wind velocity variables
00400      ! Eastward
00401      tmpvarname = trim(ncvarnam_wndx)
00402      datwindx%varname = trim(tmpvarname)
00403      datwindx%varDimIDs(1) = nodedimid
00404      datwindx%varDimIDs(2) = crdtime%dimID
00405      datwindx%varDims(1) = SIZE(wvelx, 1)
00406      datwindx%varDims(2) = SIZE(wvelx, 2)
00407      datwindx%start(1) = 1
00408      datwindx%count(1) = datwindx%varDims(1)
00409      datwindx%start(2) = 1
00410      datwindx%count(2) = datwindx%varDims(2)
00411
00412      ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
00413                          datwindx%varDimIDs, datwindx%varID)
00414      CALL netcdfcheckerr(ierr)
00415      ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',      '10-m eastward wind component')
00416      CALL netcdfcheckerr(ierr)
00417      ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
00418      CALL netcdfcheckerr(ierr)
00419      ierr = nf90_put_att(ncid, datwindx%varID, 'units',          'm s-1')
00420      CALL netcdfcheckerr(ierr)
00421      ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',    rmissv)
00422      CALL netcdfcheckerr(ierr)
00423      ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates',   'time lat lon')
00424      CALL netcdfcheckerr(ierr)
00425      ierr = nf90_put_att(ncid, datwindx%varID, 'location',      'node')
00426      CALL netcdfcheckerr(ierr)
00427      ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',          'adcirc_mesh')
00428      CALL netcdfcheckerr(ierr)
00429
00430      ALLOCATE(datwindx%var(datwindx%varDims(1), datwindx%varDims(2)))
00431      datwindx%var = wvelx
00432
00433      ! Northward
00434      tmpvarname = trim(ncvarnam_wndy)
00435      datwindy%varname = trim(tmpvarname)
00436      datwindy%varDimIDs(1) = nodedimid
00437      datwindy%varDimIDs(2) = crdtime%dimID
00438      datwindy%varDims(1) = SIZE(wvely, 1)
00439      datwindy%varDims(2) = SIZE(wvely, 2)
00440      datwindy%start(1) = 1
00441      datwindy%count(1) = datwindy%varDims(1)
00442      datwindy%start(2) = 1
00443      datwindy%count(2) = datwindy%varDims(2)
00444
00445      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
00446                          datwindy%varDimIDs, datwindy%varID)
00447      CALL netcdfcheckerr(ierr)
00448      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',      '10-m northward wind component')
00449      CALL netcdfcheckerr(ierr)
00450      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
00451      CALL netcdfcheckerr(ierr)
00452      ierr = nf90_put_att(ncid, datwindy%varID, 'units',          'm s-1')
00453      CALL netcdfcheckerr(ierr)
00454      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',    rmissv)
00455      CALL netcdfcheckerr(ierr)
00456      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates',   'time lat lon')
00457      CALL netcdfcheckerr(ierr)
00458      ierr = nf90_put_att(ncid, datwindy%varID, 'location',      'node')
00459      CALL netcdfcheckerr(ierr)
00460      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',          'adcirc_mesh')
00461      CALL netcdfcheckerr(ierr)
00462
00463      ALLOCATE(datwindy%var(datwindy%varDims(1), datwindy%varDims(2)))
00464      datwindy%var = wvely
00465
00466      !=====
00467      !===== (3) Set Deflate parameters if requested by the user
00468      !=====
00469      #ifdef NETCDF_CAN_DEFLATE
00470      IF (ncformat == nc4form) THEN
00471          ierr = nf90_def_var_deflate(ncid, crdlons%varID,      ncshuffle, ncdeflate, ncdlevel)
00472          CALL netcdfcheckerr(ierr)
00473          ierr = nf90_def_var_deflate(ncid, crdlats%varID,      ncshuffle, ncdeflate, ncdlevel)
00474          CALL netcdfcheckerr(ierr)
00475          ierr = nf90_def_var_deflate(ncid, crdxcs%varID,      ncshuffle, ncdeflate, ncdlevel)
00476          CALL netcdfcheckerr(ierr)
00477          ierr = nf90_def_var_deflate(ncid, crdyys%varID,      ncshuffle, ncdeflate, ncdlevel)
00478          CALL netcdfcheckerr(ierr)

```

```

00479      ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
00480      CALL netcdfcheckerr(ierr)
00481      ierr = nf90_def_var_deflate(ncid, datatmpres%varID, ncshuffle, ncdeflate, ncdlevel)
00482      CALL netcdfcheckerr(ierr)
00483      ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, ncdlevel)
00484      CALL netcdfcheckerr(ierr)
00485      ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, ncdlevel)
00486      CALL netcdfcheckerr(ierr)
00487  END IF
00488 #endif
00489
00490      !=====
00491      !==== (4) Global metadata definitions and variables
00492      !=====
00493      ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
00494      CALL netcdfcheckerr(ierr)
00495      ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
00496      ')')
00497      CALL netcdfcheckerr(ierr)
00498      ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
00499      CALL netcdfcheckerr(ierr)
00500      ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
00501      CALL netcdfcheckerr(ierr)
00502      ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
00503      CALL netcdfcheckerr(ierr)
00504      ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
00505      CALL netcdfcheckerr(ierr)
00506      ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
00507      CALL netcdfcheckerr(ierr)
00508      ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
00509      CALL netcdfcheckerr(ierr)
00510      ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
00511      CALL netcdfcheckerr(ierr)
00512      ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
00513      CALL netcdfcheckerr(ierr)
00514      ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
00515      CALL netcdfcheckerr(ierr)
00516      ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
00517      CALL netcdfcheckerr(ierr)
00518      ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
00519      CALL netcdfcheckerr(ierr)
00520      CALL date_and_time(values = tvals)
00521      WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
00522      moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
moddatetimestr)
00523
00524      ierr = nf90_put_att(ncid, nf90_global, 'creation_date', trim(moddatetimestr))
00525      CALL netcdfcheckerr(ierr)
00526      ierr = nf90_put_att(ncid, nf90_global, 'modification_date', trim(moddatetimestr))
00527      CALL netcdfcheckerr(ierr)
00528
00529      !----- Finalize the definitions in the NetCDF file
00530      ierr = nf90_enddef(ncid)
00531      CALL netcdfcheckerr(ierr)
00532
00533      !=====
00534      !==== (5) Put the static data into the NetCDF file and then close it
00535      !=====
00536      ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
00537      CALL netcdfcheckerr(ierr)
00538
00539      ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
00540      CALL netcdfcheckerr(ierr)
00541
00542      ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
00543      CALL netcdfcheckerr(ierr)
00544
00545      ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
00546      CALL netcdfcheckerr(ierr)
00547
00548      ierr = nf90_put_var(ncid, crdyys%varID, crdyys%var, crdyys%start, crdyys%count)
00549      CALL netcdfcheckerr(ierr)
00550
00551      ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00552      CALL netcdfcheckerr(ierr)
00553
00554      ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00555      CALL netcdfcheckerr(ierr)
00556
00557      ierr = nf90_put_var(ncid, datatmpres%varID, datatmpres%var, datatmpres%start, datatmpres%count)

```

```

00558      CALL netcdfcheckerr(ierr)
00559
00560      ierr = nf90_put_var(ncid, datwindx%varID, datwindx%var, datwindx%start, datwindx%count)
00561      CALL netcdfcheckerr(ierr)
00562
00563      ierr = nf90_put_var(ncid, datwindy%varID, datwindy%var, datwindy%start, datwindy%count)
00564      CALL netcdfcheckerr(ierr)
00565
00566
00567      !----- (16) Set all the "initialized" flags to .TRUE.
00568      crdlons%initialized = .true.
00569      crdlats%initialized = .true.
00570      crdxcs%initialized = .true.
00571      crdycs%initialized = .true.
00572      datelements%initialized = .true.
00573      datatmpres%initialized = .true.
00574      datwindx%initialized = .true.
00575      datwindy%initialized = .true.
00576
00577      myfile%fileName = adcircoutfile
00578      myfile%initialized = .true.
00579
00580      !----- Close the NetCDF file
00581      ierr = nf90_close(ncid)
00582      CALL netcdfcheckerr(ierr)
00583
00584      CALL unsetmessagesource()
00585
00586      END SUBROUTINE initadcircnetcdfoutfile
00587
00588      !=====
00589
00590      !-----
00591      ! SUBROUTINE NEW ADCIRC NETCDF OUTFILE
00592      !-----
00593      ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00594      !-----
00595      SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)
00596
00597      IMPLICIT NONE
00598
00599      INTEGER, INTENT(OUT) :: ncID
00600      CHARACTER(LEN=*), INTENT(INOUT) :: adcircOutFile
00601
00602      LOGICAL :: fileFound = .false.
00603      CHARACTER(LEN=FNAMELEN) :: outFile, sys_cmd
00604      CHARACTER(LEN=14) :: fext, date_time
00605      INTEGER :: pos, ierr, tvals(8)
00606
00607      CALL setmessagesource("NewAdcircNetCDFOutFile")
00608
00609      !-----
00610      ! Set some variables that depend upon the type of NetCDF supported.
00611      #if defined(HAVE_NETCDF4)
00612      fext = ".nc4"
00613      ncformat = nc4form
00614      #else
00615      fext = ".nc"
00616      ncformat = nc3form
00617      #endif
00618
00619      !-----
00620      ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
00621      ! extension in the filename; re-define the adcircOutFile variable.
00622      pos = scan(trim(adcircoutfile), ".", back=.true.)
00623      IF (pos > 0) THEN
00624      adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
00625      ELSE
00626      adcircoutfile = trim(adcircoutfile) // trim(fext)
00627      END IF
00628
00629      !-----
00630      ! If the adcircOutFile exists then rename it to:
00631      !   adcircOutFile-YYYYMMDDhhmmss.
00632      ! The user can remove these files afterwards.
00633      INQUIRE(file=adcircoutfile, exist=filefound)
00634      IF (filefound) THEN
00635      CALL date_and_time(values = tvals)
00636      WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
00637      outfile = trim(adcircoutfile) // "-" // trim(date_time)

```

```

00651     sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
00652     ierr = system(trim(sys_cmd))
00653     IF (ierr == 0) THEN
00654         WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
00655         CALL logmessage(info, scratchmessage)
00656         filefound = .false.
00657     ELSE
00658         WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' //
trim(outfile)
00659         CALL logmessage(error, scratchmessage)
00660     END IF
00661 END IF
00662
00663     IF (filefound) THEN
00664         WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adcircoutfile) // ' exists. Remove the
file to proceed.'
00665         CALL allmessage(error, scratchmessage)
00666
00667         CALL unsetmessagesource()
00668
00669         CALL netcdfterminate
00670     END IF
00671
00672     WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adcircoutfile) // ' and putting it in define
mode.'
00673     CALL logmessage(info, scratchmessage)
00674
00675     ! Create the NetCDF file
00676     ierr = nf90_create(adcircoutfile, ncformat, ncid)
00677     CALL netcdfcheckerr(ierr)
00678
00679     CALL unsetmessagesource()
00680
00681 END SUBROUTINE newadrcircnetcdfoutfile
00682
00683 !=====
00684
00685 !-----
00686 ! SUBROUTINE NETCDF CHECKERR
00687 !-----
00688 !
00689 !-----
00690
00691 SUBROUTINE base_netcdfcheckerr(ierr, file, line)
00692
00693     IMPLICIT NONE
00694
00695     INTEGER, INTENT(IN)          :: ierr
00696     CHARACTER(LEN=*), INTENT(IN) :: file
00697     INTEGER, INTENT(IN)          :: line
00698
00699     CHARACTER(LEN=1024)          :: tmpSTR
00700
00701     CALL setmessagesource("NetCDFCheckErr")
00702
00703     IF (ierr /= nf90_noerr) THEN
00704         CALL allmessage(error, nf90_strerror(ierr))
00705         WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
00706         CALL allmessage(info, tmpstr)
00707         CALL netcdfterminate()
00708     END IF
00709
00710     CALL unsetmessagesource()
00711
00712 END SUBROUTINE base_netcdfcheckerr
00713
00714 !=====
00715
00716 !-----
00717 ! SUBROUTINE NETCDF TERMINATE
00718 !-----
00719 !
00720 !-----
00721
00722 SUBROUTINE netcdfterminate()
00723
00724     USE version
00725
00726     IMPLICIT NONE
00727
00728     CALL setmessagesource("NetCDFTerminate")
00729
00730     CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")

```

```

00736
00737     CALL exit(1)
00738
00739     CALL unsetmessagesource()
00740
00741 END SUBROUTINE netcdfterminate
00742
00743 !=====
00744
00745
00746 !-----
00747 !  SUBROUTINE WRITE NETCDF RECORD
00748 !-----
00749 !  author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00762 !-----
00763 SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)
00764
00765     IMPLICIT NONE
00766
00767     CHARACTER(LEN=*) , INTENT(IN) :: adcircOutFile
00768
00769     INTEGER :: timeLoc
00770     INTEGER :: ncID, ierr, nodes
00771     INTEGER :: start(2), kount(2)
00772
00773     CALL setmessagesource("WriteNetCDFRecord")
00774
00775     ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
00776     CALL netcdfcheckerr(ierr)
00777
00778     ! Set up the 2D netcdf data extents
00779     ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
00780     start(1) = 1
00781     start(2) = myfile%fileRecCounter
00782     kount(1) = nodes
00783     kount(2) = mytime%timeLen
00784
00785     ierr = nf90_put_var(ncid, datatmpres%varID, wpress(:, timeloc), start, kount)
00786     CALL netcdfcheckerr(ierr)
00787
00788     ierr = nf90_put_var(ncid, datwindx%varID, wvelx(:, timeloc), start, kount)
00789     CALL netcdfcheckerr(ierr)
00790
00791     ierr = nf90_put_var(ncid, datwindy%varID, wvely(:, timeloc), start, kount)
00792     CALL netcdfcheckerr(ierr)
00793
00794     ! Close netCDF file
00795     ierr = nf90_close(ncid)
00796     CALL netcdfcheckerr(ierr)
00797
00798     DEALLOCATE(mytime%time)
00799
00800     CALL unsetmessagesource()
00801
00802 END SUBROUTINE writenetcdfrecord
00803
00804 !=====
00805
00806 !-----
00807 !  SUBROUTINE SET RECORD COUNTER AND STORE TIME
00808 !-----
00809 !  author
00825 !-----
00826 SUBROUTINE setrecordcounterandstoretime(ncID, f, t)
00827
00828     IMPLICIT NONE
00829
00830     INTEGER, INTENT(IN) :: ncID
00831     TYPE(FileData_T), INTENT(INOUT) :: f
00832     TYPE(TimeData_T), INTENT(INOUT) :: t
00833
00834     REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
00835     LOGICAL :: timeFound ! true if current time is in array of stored times
00836
00837     INTEGER :: ndim ! number of dimensions in the netcdf file
00838     INTEGER :: nvar ! number of variables in the netcdf file
00839     INTEGER :: natt ! number of attributes in the netcdf file
00840
00841     INTEGER :: counti(1), starti(1)
00842     INTEGER :: ierr ! success or failure of netcdf call
00843     INTEGER :: i ! loop counter

```



```

00844
00845
00846     CALL setmessagesource("SetRecordCounterAndStoreTime")
00847
00848     ! Inquire the time variable
00849     ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
00850     CALL netcdfcheckerr(ierr)
00851
00852     ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
00853     CALL netcdfcheckerr(ierr)
00854
00855     ierr = nf90_inq_varid(ncid, 'time', t%timeID)
00856     CALL netcdfcheckerr(ierr)
00857
00858     ! Determine the relationship between the current simulation time
00859     ! and the time array stored in the netcdf file. Set the record
00860     ! counter based on this relationship.
00861     IF (f%fileRecCounter /= 0) THEN
00862         ALLOCATE(storedtimes(f%fileRecCounter))
00863         ierr = nf90_get_var(ncid, t%timeID, storedtimes)
00864         CALL netcdfcheckerr(ierr)
00865         timefound = .false.
00866
00867         DO i = 1, f%fileRecCounter
00868             IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
00869                 timefound = .true.
00870                 EXIT
00871             ENDIF
00872         END DO
00873
00874         IF (timefound .EQV. .false.) THEN
00875             ! Increment the record counter so that we can store data at the
00876             ! next location in the netcdf file (i.e., all of the times
00877             ! in the netcdf file were found to be earlier than the current
00878             ! adcirc simulation time).
00879             f%fileRecCounter = f%fileRecCounter + 1
00880         ELSE
00881             ! set the counter at the index that reflects the
00882             ! current time within the netcdf file (or is between two times
00883             ! found in the netcdf file).
00884             ! WARNING: all subsequent data will remain in the file, we
00885             ! are just overwriting it ... if we don't overwrite all of it,
00886             ! the pre-existing data will still be there, which is probably
00887             ! not what the user intended ... but apparently there is no
00888             ! way to delete data from netcdf files:
00889             ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
00890             scratchformat = '("Overwriting pre-existing data in netcdf file ",a," // &
00891             " for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.')"
00892             WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
00893             CALL allmessage(info, scratchmessage)
00894             f%fileRecCounter = i
00895         ENDIF
00896
00897         DEALLOCATE(storedtimes)
00898     ELSE
00899         ! set the counter at 1 so we can record our first time value
00900         f%fileRecCounter = 1
00901     ENDIF
00902
00903     ! Store simulation time in netcdf file
00904     starti(1) = f%fileRecCounter
00905     counti(1) = t%timeLen
00906     ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
00907     CALL netcdfcheckerr(ierr)
00908
00909     CALL unsetmessagesource()
00910
00911 END SUBROUTINE setrecordcounterandstoretime
00912
00913 !=====
00914
00915 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.](#)

```

00001 !-----
00002 !           M O D U L E   N E T C D F   I O
00003 !-----
00014 !-----
00015
00016 MODULE pahm_netcdfio
00017
00018   USE pahm_sizes
00019   USE pahm_messages
00020   USE pahm_global
00021   USE pahm_mesh, ONLY : agrid, np, ne, nfn, nm, slam, sfea, xcslam, ycsfea, slam0, sfea0
00022   USE netcdf
00023
00024 #ifdef __INTEL_COMPILER
00025   USE ifport
00026 #endif
00027
00028   IMPLICIT NONE
00029
00030 #define NetCDFCheckErr(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
00031
00032   INTEGER, PRIVATE :: ncFormat
00033   INTEGER, PARAMETER, PRIVATE :: nc4Form = ior(nf90_netcdf4, nf90_classic_model)
00034   INTEGER, PARAMETER, PRIVATE :: nc3Form = ior(nf90_clobber, 0)
00035
00036   INTEGER, PRIVATE :: nodeDimID, vertDimID, elemDimID, meshDimID
00037   INTEGER, PRIVATE :: elemVarID, meshVarID, projVarID
00038
00039   TYPE :: filedata_t
00040     LOGICAL :: initialized = .false.
00041     INTEGER :: fileRecCounter = 0
00042     CHARACTER(LEN=FNAMELEN) :: fileName
00043     LOGICAL :: fileFound = .false. ! .true. if the netCDF file is present
00044   END TYPE filedata_t
00045
00046   TYPE :: timedata_t
00047     LOGICAL :: initialized = .false.
00048     INTEGER :: timeLen = 1 ! number of time slices to write
00049     INTEGER :: timeDimID
00050     INTEGER :: timeID
00051     INTEGER :: timeDims(1)
00052     REAL(SZ), ALLOCATABLE :: time(:)
00053   END TYPE timedata_t
00054
00055   TYPE, PRIVATE :: adcircoorddata_t
00056     LOGICAL :: initialized = .false.
00057     REAL(SZ) :: initVal
00058     INTEGER :: dimID
00059     INTEGER :: varID
00060     INTEGER :: varDimIDs
00061     INTEGER :: varDims
00062     CHARACTER(50) :: varname
00063     REAL(SZ), ALLOCATABLE :: var(:)
00064     INTEGER :: start(1), count(1)
00065   END TYPE adcircoorddata_t
00066
00067   TYPE, PRIVATE :: adcircvardata_t
00068     LOGICAL :: initialized = .false.
00069     REAL(SZ) :: initVal
00070     INTEGER :: varID
00071     INTEGER :: varDimIDs(2)
00072     INTEGER :: varDims(2)
00073     CHARACTER(50) :: varname
00074     REAL(SZ), ALLOCATABLE :: var(:, :)
00075     INTEGER :: start(2), count(2)
00076   END TYPE adcircvardata_t
00077
00078   TYPE, PRIVATE :: adcircvardata3d_t
00079     LOGICAL :: initialized = .false.
00080     REAL(SZ) :: initVal
00081     INTEGER :: varID
00082     INTEGER :: varDimIDs(3)
00083     INTEGER :: varDims(3)
00084     CHARACTER(50) :: varname
00085     REAL(SZ), ALLOCATABLE :: var(:, :, :)
00086     INTEGER :: start(3), count(3)

```

```

00087  END TYPE adcircvardata3d_t
00088
00089  TYPE(FileData_T), SAVE    :: myFile
00090  TYPE(TimeData_T), SAVE    :: myTime
00091
00092  TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdTime
00093  TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLons
00094  TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdLats
00095  TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdXCs
00096  TYPE(AdcircCoordData_T), PRIVATE, SAVE :: crdYCs
00097
00098  TYPE(AdcircVarData_T), PRIVATE, SAVE :: datElements
00099  TYPE(AdcircVarData_T), PRIVATE, SAVE :: datAtmPres
00100  TYPE(AdcircVarData_T), PRIVATE, SAVE :: datWindX
00101  TYPE(AdcircVarData_T), PRIVATE, SAVE :: datWindY
00102
00103
00104  CONTAINS
00105
00106
00107  !-----
00108  !  SUBROUTINE  INIT  ADCIRC  NETCDF  OUT  FILE
00109  !-----
00125  !-----
00126  SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)
00127
00128      USE version
00129      USE timedateutils, ONLY : gettimeconvsec, datetime2string
00130
00131      IMPLICIT NONE
00132
00133      CHARACTER(LEN=*) , INTENT(INOUT) :: adcircOutFile
00134
00135      INTEGER                :: ncID
00136      CHARACTER(LEN=64)      :: refDateTimeStr, modDateTimeStr, tmpVarName
00137      CHARACTER(LEN=128)     :: institution, source, history, comments, host, &
00138                             conventions, contact, references
00139      INTEGER                :: tvals(8)
00140      INTEGER                :: ierr ! success or failure of a netcdf call
00141      INTEGER                :: iCnt, jCnt
00142
00143      LOGICAL, SAVE          :: firstCall = .true.
00144
00145
00146      IF (firstcall) THEN
00147          firstcall = .false.
00148
00149          CALL setmessagesource("InitAdcircNetCDFOutFile")
00150
00151
00152          refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units =
unittime)
00153
00154          institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
00155          source      = "
00156          history     = "
00157          comments    = "
00158          host        = "
00159          conventions = 'UGRID-0.9.0'
00160          contact     = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
00161          references  = "
00162
00163
00164          ! Create the NetCDF output file.
00165          CALL newadcircnetcdfoutfile(ncid, adcircoutfile)
00166
00167          !=====
00168          !===== (1) Define all the dimensions
00169          !=====
00170          tmpvarname = 'time'
00171          ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtime%dimID)
00172          CALL netcdfcheckerr(ierr)
00173
00174          tmpvarname = 'longitude'
00175          ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
00176          CALL netcdfcheckerr(ierr)
00177
00178          tmpvarname = 'latitude'
00179          ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
00180          CALL netcdfcheckerr(ierr)
00181

```

```

00182     tmpvarname = 'node'
00183     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
00184     CALL netcdfcheckerr(ierr)
00185
00186     tmpvarname = 'element'
00187     ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemdimid)
00188     CALL netcdfcheckerr(ierr)
00189
00190     tmpvarname = 'noel'
00191     ierr = nf90_def_dim(ncid, trim(tmpvarname), 3, vertdimid)
00192     CALL netcdfcheckerr(ierr)
00193
00194     tmpvarname = 'mesh'
00195     ierr = nf90_def_dim(ncid, trim(tmpvarname), 1, meshdimid)
00196     CALL netcdfcheckerr(ierr)
00197
00198     !=====
00199     !===== (2) Define all the variables
00200     !=====
00201     !----- Time variable
00202     tmpvarname = 'time'
00203     crdtime%varname = trim(tmpvarname)
00204     crdtime%varDimIDs = crdtime%dimID
00205     crdtime%varDims = SIZE(times, 1)
00206     crdtime%start(1) = 1
00207     crdtime%count(1) = crdtime%varDims
00208
00209     ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
00210     CALL netcdfcheckerr(ierr)
00211     ierr = nf90_put_att(ncid, crdtime%varID, 'long_name', 'model ' // trim(tmpvarname))
00212     CALL netcdfcheckerr(ierr)
00213     ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
00214     CALL netcdfcheckerr(ierr)
00215     ierr = nf90_put_att(ncid, crdtime%varID, 'units', trim(refdatetimestr))
00216     CALL netcdfcheckerr(ierr)
00217
00218     ALLOCATE(crdtime%var(crdtime%varDims))
00219     crdtime%var = times * gettimeconvsec(unittime, 1)
00220
00221     !----- Longitude variable
00222     tmpvarname = 'longitude'
00223     crdlons%varname = trim(tmpvarname)
00224     crdlons%varDimIDs = nodedimid
00225     ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlons%varDims)
00226     CALL netcdfcheckerr(ierr)
00227     crdlons%start(1) = 1
00228     crdlons%count(1) = crdlons%varDims
00229
00230     ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
00231     CALL netcdfcheckerr(ierr)
00232     ierr = nf90_put_att(ncid, crdlons%varID, 'long_name', trim(tmpvarname))
00233     CALL netcdfcheckerr(ierr)
00234     ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
00235     CALL netcdfcheckerr(ierr)
00236     ierr = nf90_put_att(ncid, crdlons%varID, 'units', 'degrees_east')
00237     CALL netcdfcheckerr(ierr)
00238     ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue', rmissv)
00239     CALL netcdfcheckerr(ierr)
00240     ierr = nf90_put_att(ncid, crdlons%varID, 'positive', 'east')
00241     CALL netcdfcheckerr(ierr)
00242
00243     ALLOCATE(crdlons%var(crdlons%varDims))
00244     crdlons%var = slam
00245
00246     !----- Latitude variable
00247     tmpvarname = 'latitude'
00248     crdlats%varname = trim(tmpvarname)
00249     crdlats%varDimIDs = nodedimid
00250     ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlats%varDims)
00251     CALL netcdfcheckerr(ierr)
00252     crdlats%start(1) = 1
00253     crdlats%count(1) = crdlats%varDims
00254
00255     ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
00256     CALL netcdfcheckerr(ierr)
00257     ierr = nf90_put_att(ncid, crdlats%varID, 'long_name', trim(tmpvarname))
00258     CALL netcdfcheckerr(ierr)
00259     ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
00260     CALL netcdfcheckerr(ierr)
00261     ierr = nf90_put_att(ncid, crdlats%varID, 'units', 'degrees_north')
00262     CALL netcdfcheckerr(ierr)

```

```

00263      ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',      rmissv)
00264      CALL netcdfcheckerr(ierr)
00265      ierr = nf90_put_att(ncid, crdlats%varID, 'positive',      'north')
00266      CALL netcdfcheckerr(ierr)
00267
00268      ALLOCATE(crdlats%var(crdlats%varDims))
00269      crdlats%var = sfea
00270
00271      !----- Element variable
00272      !----- We need to switch the order in array for NetCDF
00273      !----- It should be: elements(nf, icnt) and NOT elements(icnt, nf)
00274      tmpvarname = 'tri'
00275      datelements%varname = trim(tmpvarname)
00276      datelements%varDimIDs(1) = vertdimid
00277      datelements%varDimIDs(2) = elemidmid
00278      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(1), len = datelements%varDims(1))
00279      CALL netcdfcheckerr(ierr)
00280      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(2), len = datelements%varDims(2))
00281      CALL netcdfcheckerr(ierr)
00282      datelements%start(1) = 1
00283      datelements%count(1) = datelements%varDims(1)
00284      datelements%start(2) = 1
00285      datelements%count(2) = datelements%varDims(2)
00286
00287      ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
00288      CALL netcdfcheckerr(ierr)
00289      ierr = nf90_put_att(ncid, datelements%varID, 'long_name',      trim(tmpvarname))
00290      CALL netcdfcheckerr(ierr)
00291      ierr = nf90_put_att(ncid, datelements%varID, 'standard_name', trim(tmpvarname))
00292      CALL netcdfcheckerr(ierr)
00293      ierr = nf90_put_att(ncid, datelements%varID, 'cf_role',      'face_node_connectivity')
00294      CALL netcdfcheckerr(ierr)
00295      ierr = nf90_put_att(ncid, datelements%varID, 'start_index',  1)
00296      CALL netcdfcheckerr(ierr)
00297      ierr = nf90_put_att(ncid, datelements%varID, 'units',      'nondimensional')
00298      CALL netcdfcheckerr(ierr)
00299      ierr = nf90_put_att(ncid, datelements%varID, '_FillValue',    imissv)
00300      CALL netcdfcheckerr(ierr)
00301
00302      ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
00303      DO icnt = 1, datelements%varDims(2)
00304      DO jcnt = 1, datelements%varDims(1)
00305      datelements%var(jcnt, icnt) = nm(icnt, jcnt)
00306      END DO
00307      END DO
00308
00309      !----- Mesh variable
00310      tmpvarname = 'adcirc_mesh'
00311      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
00312      CALL netcdfcheckerr(ierr)
00313
00314      ierr = nf90_put_att(ncid, meshvarid, 'long_name',      'mesh_topology')
00315      CALL netcdfcheckerr(ierr)
00316      ierr = nf90_put_att(ncid, meshvarid, 'standard_name',  'mesh_topology')
00317      CALL netcdfcheckerr(ierr)
00318      ierr = nf90_put_att(ncid, meshvarid, 'cf_role',      'mesh_topology')
00319      CALL netcdfcheckerr(ierr)
00320      ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates', 'lon lat')
00321      CALL netcdfcheckerr(ierr)
00322      ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
00323      CALL netcdfcheckerr(ierr)
00324
00325      !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
00326      tmpvarname = 'projection'
00327      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
00328      CALL netcdfcheckerr(ierr)
00329
00330      ierr = nf90_put_att(ncid, projvarid, 'long_name',      'equidistant cylindrical projection')
00331      CALL netcdfcheckerr(ierr)
00332      ierr = nf90_put_att(ncid, projvarid, 'standard_name',  'CPP')
00333      CALL netcdfcheckerr(ierr)
00334      ierr = nf90_put_att(ncid, projvarid, 'node_coordinates', 'x y')
00335      CALL netcdfcheckerr(ierr)
00336      ierr = nf90_put_att(ncid, projvarid, 'lon0',      slam0)
00337      CALL netcdfcheckerr(ierr)
00338      ierr = nf90_put_att(ncid, projvarid, 'lat0',      sfea0)
00339      CALL netcdfcheckerr(ierr)
00340      ierr = nf90_put_att(ncid, projvarid, 'earth_radius', rearth)
00341      CALL netcdfcheckerr(ierr)
00342
00343      !----- CPP CPP x-coordinates

```

```

00344     tmpvarname = 'x'
00345     crdxcs%varname = trim(tmpvarname)
00346     crdxcs%dimID = nodedimid
00347     crdxcs%varDimIDs = nodedimid
00348     ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
00349     CALL netcdfcheckerr(ierr)
00350     crdxcs%start(1) = 1
00351     crdxcs%count(1) = crdxcs%varDims
00352
00353     ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
00354     CALL netcdfcheckerr(ierr)
00355     ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',      'CPP x coordinate')
00356     CALL netcdfcheckerr(ierr)
00357     ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
00358     CALL netcdfcheckerr(ierr)
00359     ierr = nf90_put_att(ncid, crdxcs%varID, 'units',          'm')
00360     CALL netcdfcheckerr(ierr)
00361     ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',    rmissv)
00362     CALL netcdfcheckerr(ierr)
00363
00364     ALLOCATE(crdxcs%var(crdxcs%varDims))
00365     crdxcs%var = xcslam
00366
00367 !----- CPP y-coordinates
00368     tmpvarname = 'y'
00369     crdyics%varname = trim(tmpvarname)
00370     crdyics%dimID = nodedimid
00371     crdyics%varDimIDs = nodedimid
00372     ierr = nf90_inquire_dimension(ncid, crdyics%dimID, len = crdyics%varDims)
00373     CALL netcdfcheckerr(ierr)
00374     crdyics%start(1) = 1
00375     crdyics%count(1) = crdyics%varDims
00376
00377     ierr = nf90_def_var(ncid, trim(crdyics%varname), nf90_double, crdyics%varDimIDs, crdyics%varID)
00378     CALL netcdfcheckerr(ierr)
00379     ierr = nf90_put_att(ncid, crdyics%varID, 'long_name',      'CPP y coordinate')
00380     CALL netcdfcheckerr(ierr)
00381     ierr = nf90_put_att(ncid, crdyics%varID, 'standard_name', 'cpp_y')
00382     CALL netcdfcheckerr(ierr)
00383     ierr = nf90_put_att(ncid, crdyics%varID, 'units',          'm')
00384     CALL netcdfcheckerr(ierr)
00385     ierr = nf90_put_att(ncid, crdyics%varID, '_FillValue',    rmissv)
00386     CALL netcdfcheckerr(ierr)
00387
00388     ALLOCATE(crdyics%var(crdyics%varDims))
00389     crdyics%var = ycsfea
00390
00391 !----- Atmospheric Pressure variable
00392     tmpvarname = trim(ncvarnam_pres)
00393     datatmpres%varname = trim(tmpvarname)
00394     datatmpres%varDimIDs(1) = nodedimid
00395     datatmpres%varDimIDs(2) = crdtime%dimID
00396     datatmpres%varDims(1) = SIZE(wpress, 1)
00397     datatmpres%varDims(2) = crdtime%varDims
00398     datatmpres%start(1) = 1
00399     datatmpres%count(1) = datatmpres%varDims(1)
00400     datatmpres%start(2) = 1
00401     datatmpres%count(2) = datatmpres%varDims(2)
00402
00403     ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
00404         datatmpres%varDimIDs, datatmpres%varID)
00405     CALL netcdfcheckerr(ierr)
00406     ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',      'air pressure at sea level')
00407     CALL netcdfcheckerr(ierr)
00408     ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
00409     CALL netcdfcheckerr(ierr)
00410     ierr = nf90_put_att(ncid, datatmpres%varID, 'units',          'Pa')
00411     CALL netcdfcheckerr(ierr)
00412     ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
00413     CALL netcdfcheckerr(ierr)
00414     ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',    'time lat lon')
00415     CALL netcdfcheckerr(ierr)
00416     ierr = nf90_put_att(ncid, datatmpres%varID, 'location',        'node')
00417     CALL netcdfcheckerr(ierr)
00418     ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',            'adcirc_mesh')
00419     CALL netcdfcheckerr(ierr)
00420
00421 !PV     ALLOCATE(datAtmPres%var(datAtmPres%varDims(1), datAtmPres%varDims(2)))
00422 !PV     datAtmPres%var = wpress
00423
00424 !----- Wind velocity variables

```

```

00425      ! Eastward
00426      tmpvarname = trim(ncvarnam_wndx)
00427      datwindx%varname = trim(tmpvarname)
00428      datwindx%varDimIDs(1) = nodedimid
00429      datwindx%varDimIDs(2) = crdtime%dimID
00430      datwindx%varDims(1) = SIZE(wvelx, 1)
00431      datwindx%varDims(2) = crdtime%varDims
00432      datwindx%start(1) = 1
00433      datwindx%count(1) = datwindx%varDims(1)
00434      datwindx%start(2) = 1
00435      datwindx%count(2) = datwindx%varDims(2)
00436
00437      ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
00438                        datwindx%varDimIDs, datwindx%varID)
00439      CALL netcdfcheckerr(ierr)
00440      ierr = nf90_put_att(ncid, datwindx%varID, 'long_name', '10-m eastward wind component')
00441      CALL netcdfcheckerr(ierr)
00442      ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
00443      CALL netcdfcheckerr(ierr)
00444      ierr = nf90_put_att(ncid, datwindx%varID, 'units', 'm s-1')
00445      CALL netcdfcheckerr(ierr)
00446      ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue', rmissv)
00447      CALL netcdfcheckerr(ierr)
00448      ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates', 'time lat lon')
00449      CALL netcdfcheckerr(ierr)
00450      ierr = nf90_put_att(ncid, datwindx%varID, 'location', 'node')
00451      CALL netcdfcheckerr(ierr)
00452      ierr = nf90_put_att(ncid, datwindx%varID, 'mesh', 'adcirc_mesh')
00453      CALL netcdfcheckerr(ierr)
00454
00455      !PV      ALLOCATE(datWindX%var(datWindX%varDims(1), datWindX%varDims(2)))
00456      !PV      datWindX%var = wvelX
00457
00458      ! Northward
00459      tmpvarname = trim(ncvarnam_wndy)
00460      datwindy%varname = trim(tmpvarname)
00461      datwindy%varDimIDs(1) = nodedimid
00462      datwindy%varDimIDs(2) = crdtime%dimID
00463      datwindy%varDims(1) = SIZE(wvely, 1)
00464      datwindy%varDims(2) = crdtime%varDims
00465      datwindy%start(1) = 1
00466      datwindy%count(1) = datwindy%varDims(1)
00467      datwindy%start(2) = 1
00468      datwindy%count(2) = datwindy%varDims(2)
00469
00470      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
00471                        datwindy%varDimIDs, datwindy%varID)
00472      CALL netcdfcheckerr(ierr)
00473      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name', '10-m northward wind component')
00474      CALL netcdfcheckerr(ierr)
00475      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
00476      CALL netcdfcheckerr(ierr)
00477      ierr = nf90_put_att(ncid, datwindy%varID, 'units', 'm s-1')
00478      CALL netcdfcheckerr(ierr)
00479      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue', rmissv)
00480      CALL netcdfcheckerr(ierr)
00481      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates', 'time lat lon')
00482      CALL netcdfcheckerr(ierr)
00483      ierr = nf90_put_att(ncid, datwindy%varID, 'location', 'node')
00484      CALL netcdfcheckerr(ierr)
00485      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh', 'adcirc_mesh')
00486      CALL netcdfcheckerr(ierr)
00487
00488      !PV      ALLOCATE(datWindY%var(datWindY%varDims(1), datWindY%varDims(2)))
00489      !PV      datWindY%var = wvelY
00490
00491      !=====
00492      !===== (3) Set Deflate parameters if requested by the user
00493      !=====
00494      #ifdef NETCDF_CAN_DEFLATE
00495      IF (ncformat == nc4form) THEN
00496          ierr = nf90_def_var_deflate(ncid, crdlons%varID, ncshuffle, ncdeflate, ncdlevel)
00497          CALL netcdfcheckerr(ierr)
00498          ierr = nf90_def_var_deflate(ncid, crdlats%varID, ncshuffle, ncdeflate, ncdlevel)
00499          CALL netcdfcheckerr(ierr)
00500          ierr = nf90_def_var_deflate(ncid, crdxcs%varID, ncshuffle, ncdeflate, ncdlevel)
00501          CALL netcdfcheckerr(ierr)
00502          ierr = nf90_def_var_deflate(ncid, crdyys%varID, ncshuffle, ncdeflate, ncdlevel)
00503          CALL netcdfcheckerr(ierr)
00504          ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
00505          CALL netcdfcheckerr(ierr)

```



```

00506         ierr = nf90_def_var_deflate(ncid, datatmpres%varID, ncshuffle, ncdeflate, ncdlevel)
00507         CALL netcdfcheckerr(ierr)
00508         ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, ncdlevel)
00509         CALL netcdfcheckerr(ierr)
00510         ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, ncdlevel)
00511         CALL netcdfcheckerr(ierr)
00512     END IF
00513 #endif
00514
00515     !=====
00516     !----- (4) Global metadata definitions and variables
00517     !=====
00518     ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
00519     CALL netcdfcheckerr(ierr)
00520     ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
    ')')
00521     CALL netcdfcheckerr(ierr)
00522     ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
00523     CALL netcdfcheckerr(ierr)
00524     ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
00525     CALL netcdfcheckerr(ierr)
00526     ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
00527     CALL netcdfcheckerr(ierr)
00528     ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
00529     CALL netcdfcheckerr(ierr)
00530     ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
00531     CALL netcdfcheckerr(ierr)
00532     ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
00533     CALL netcdfcheckerr(ierr)
00534     ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
00535     CALL netcdfcheckerr(ierr)
00536     ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
00537     CALL netcdfcheckerr(ierr)
00538     ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
00539     CALL netcdfcheckerr(ierr)
00540     ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
00541     CALL netcdfcheckerr(ierr)
00542     ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
00543     CALL netcdfcheckerr(ierr)
00544
00545     CALL date_and_time(values = tvals)
00546     WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
00547     moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
moddatetimestr)
00548
00549     ierr = nf90_put_att(ncid, nf90_global, 'creation_date', trim(moddatetimestr))
00550     CALL netcdfcheckerr(ierr)
00551     ierr = nf90_put_att(ncid, nf90_global, 'modification_date', trim(moddatetimestr))
00552     CALL netcdfcheckerr(ierr)
00553
00554     !----- Finalize the definitions in the NetCDF file
00555     ierr = nf90_enddef(ncid)
00556     CALL netcdfcheckerr(ierr)
00557
00558     !=====
00559     !----- (5) Put the static data into the NetCDF file and then close it
00560     !=====
00561     ierr = nf90_put_var(ncid, crdtime%varID, crdtime%var, crdtime%start, crdtime%count)
00562     CALL netcdfcheckerr(ierr)
00563
00564     ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
00565     CALL netcdfcheckerr(ierr)
00566
00567     ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
00568     CALL netcdfcheckerr(ierr)
00569
00570     ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
00571     CALL netcdfcheckerr(ierr)
00572
00573     ierr = nf90_put_var(ncid, crdyics%varID, crdyics%var, crdyics%start, crdyics%count)
00574     CALL netcdfcheckerr(ierr)
00575
00576     ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00577     CALL netcdfcheckerr(ierr)
00578
00579     !PV     ierr = NF90_PUT_VAR(ncID, datElements%varID, datElements%var, datElements%start,
datElements%count)
00580     !PV     CALL NetCDFCheckErr(ierr)
00581
00582     !PV     ierr = NF90_PUT_VAR(ncID, datAtmPres%varID, datAtmPres%var, datAtmPres%start, datAtmPres%count)
00583     !PV     CALL NetCDFCheckErr(ierr)

```

```

00584
00585 !PV      ierr = NF90_PUT_VAR(ncID, datWindX%varID, datWindX%var, datWindX%start, datWindX%count)
00586 !PV      CALL NetCDFCheckErr(ierr)
00587
00588 !PV      ierr = NF90_PUT_VAR(ncID, datWindY%varID, datWindY%var, datWindY%start, datWindY%count)
00589 !PV      CALL NetCDFCheckErr(ierr)
00590
00591
00592      !----- (16) Set all the "initialized" flags to .TRUE.
00593      crdlons%initialized = .true.
00594      crdlats%initialized = .true.
00595      crdxcs%initialized = .true.
00596      crdyys%initialized = .true.
00597      datelements%initialized = .true.
00598      datatmpres%initialized = .true.
00599      datwindx%initialized = .true.
00600      datwindy%initialized = .true.
00601
00602      myfile%fileName = adccircoutfile
00603      myfile%initialized = .true.
00604
00605      !----- Close the NetCDF file
00606      ierr = nf90_close(ncid)
00607      CALL netcdfcheckerr(ierr)
00608
00609      CALL unsetmessagesource()
00610
00611      END IF !firstCall
00612
00613      END SUBROUTINE initadccircnetcdfoutfile
00614
00615      !=====
00616
00617      !-----
00618      ! SUBROUTINE NEW ADCIRC NETCDF OUT FILE
00619      !-----
00620
00621      SUBROUTINE newadccircnetcdfoutfile(ncID, adccircOutFile)
00622
00623      IMPLICIT NONE
00624
00625      INTEGER, INTENT(OUT)          :: ncID
00626      CHARACTER(LEN=*) , INTENT(INOUT) :: adccircOutFile
00627
00628      LOGICAL                      :: fileFound = .false.
00629      CHARACTER(LEN=FNAMELEN)      :: outFile, sys_cmd
00630      CHARACTER(LEN=14)            :: fext, date_time
00631      INTEGER                      :: pos, ierr, tvals(8)
00632
00633      CALL setmessagesource("NewAdccircNetCDFOutFile")
00634
00635      !-----
00636      ! Set some variables that depend upon the type of NetCDF supported.
00637      #if defined(HAVE_NETCDF4)
00638      fext = ".nc4"
00639      ncformat = nc4form
00640      #else
00641      fext = ".nc"
00642      ncformat = nc3form
00643      #endif
00644
00645      !-----
00646      ! Remove the extension of the adccircOutFile and add a ".nc" or ".nc4"
00647      ! extension in the filename; re-define the adccircOutFile variable.
00648      pos = scan(trim(adccircoutfile), ".", back=.true.)
00649      IF (pos > 0) THEN
00650      adccircoutfile = adccircoutfile(1:pos - 1) // trim(fext)
00651      ELSE
00652      adccircoutfile = trim(adccircoutfile) // trim(fext)
00653      END IF
00654
00655      !-----
00656      ! If the adccircOutFile exists then rename it to:
00657      !   adccircOutFile-YYYYMMDDhhmmss.
00658      ! The user can remove these files afterwards.
00659      INQUIRE(file=adccircoutfile, exist=filefound)
00660      IF (filefound) THEN
00661      CALL date_and_time(values = tvals)
00662      WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
00663      outfile = trim(adccircoutfile) // "-" // trim(date_time)

```

```

00683     sys_cmd = "mv " // trim(adircoutfile) // " " // trim(outfile)
00684     ierr = system(trim(sys_cmd))
00685     IF (ierr == 0) THEN
00686         WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adircoutfile) // ' to ' // trim(outfile)
00687         CALL logmessage(info, scratchmessage)
00688         filefound = .false.
00689     ELSE
00690         WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adircoutfile) // ' to ' //
trim(outfile)
00691         CALL logmessage(error, scratchmessage)
00692     END IF
00693 END IF
00694
00695     IF (filefound) THEN
00696         WRITE(scratchmessage, '(a)') 'The NetCDF ouput file ' // trim(adircoutfile) // ' exists. Remove the
file to proceed.'
00697         CALL allmessage(error, scratchmessage)
00698
00699         CALL unsetmessagesource()
00700
00701         CALL netcdfterminate
00702     END IF
00703
00704     WRITE(scratchmessage, '(a)') 'Creating the file ' // trim(adircoutfile) // ' and putting it in define
mode.'
00705     CALL logmessage(info, scratchmessage)
00706
00707     ! Create the NetCDF file
00708     ierr = nf90_create(adircoutfile, ncformat, ncid)
00709     CALL netcdfcheckerr(ierr)
00710
00711     CALL unsetmessagesource()
00712
00713 END SUBROUTINE newadcircnetcdfoutfile
00714
00715 !=====
00716
00717 !-----
00718 ! SUBROUTINE NETCDF CHECKERR
00719 !-----
00720 !-----
00721 SUBROUTINE base_netcdfcheckerr(ierr, file, line)
00722
00723     IMPLICIT NONE
00724
00725     INTEGER, INTENT(IN)          :: ierr
00726     CHARACTER(LEN=*) , INTENT(IN) :: file
00727     INTEGER, INTENT(IN)          :: line
00728
00729     CHARACTER(LEN=1024)          :: tmpSTR
00730
00731     CALL setmessagesource("NetCDFCheckErr")
00732
00733     IF (ierr /= nf90_noerr) THEN
00734         CALL allmessage(error, nf90_strerror(ierr))
00735         WRITE(tmpstr, '(a, a, i5)') trim(file), ': ', line
00736         CALL allmessage(info, tmpstr)
00737         CALL netcdfterminate()
00738     END IF
00739
00740     CALL unsetmessagesource()
00741
00742 END SUBROUTINE base_netcdfcheckerr
00743
00744 !=====
00745
00746 !-----
00747 ! SUBROUTINE NETCDF TERMINATE
00748 !-----
00749 !-----
00750 SUBROUTINE netcdfterminate()
00751
00752     USE version
00753
00754     IMPLICIT NONE
00755
00756     CALL setmessagesource("NetCDFTerminate")
00757
00758     CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")
00759
00760     CALL exit(1)

```

```

00788
00789     CALL unsetmessagesource()
00790
00791 END SUBROUTINE netcdfterminate
00792
00793 !=====
00794
00795
00796 !-----
00797 !  SUBROUTINE WRITE NETCDF RECORD
00798 !-----
00811 !-----
00812 SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)
00813
00814     USE timedateutils, ONLY : gettimeconvsec
00815
00816     IMPLICIT NONE
00817
00818     CHARACTER(LEN=*) , INTENT(IN) :: adcircOutFile
00819
00820     INTEGER :: timeLoc
00821     INTEGER :: ncID, ierr, nodes
00822     INTEGER :: start(2), kount(2)
00823
00824
00825     CALL setmessagesource("WriteNetCDFRecord")
00826
00827     ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
00828     CALL netcdfcheckerr(ierr)
00829
00830     ! Set up the 2D netcdf data extents
00831     ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
00832     start(1) = 1
00833     start(2) = timeloc
00834     kount(1) = nodes
00835     kount(2) = 1
00836
00837     ierr = nf90_put_var(ncid, datatmpres%varID, wpress, start, kount)
00838     CALL netcdfcheckerr(ierr)
00839
00840     ierr = nf90_put_var(ncid, datwindx%varID, wvelx, start, kount)
00841     CALL netcdfcheckerr(ierr)
00842
00843     ierr = nf90_put_var(ncid, datwindy%varID, wvely, start, kount)
00844     CALL netcdfcheckerr(ierr)
00845
00846     ! Close netCDF file
00847     ierr = nf90_close(ncid)
00848     CALL netcdfcheckerr(ierr)
00849
00850     CALL unsetmessagesource()
00851
00852 END SUBROUTINE writenetcdfrecord
00853
00854 !=====
00855
00856 !-----
00857 !  SUBROUTINE SET RECORD COUNTER AND STORE TIME
00858 !-----
00881 !-----
00882 SUBROUTINE setrecordcounterandstoretime(ncID, f, t)
00883
00884     IMPLICIT NONE
00885
00886     INTEGER, INTENT(IN) :: ncID
00887     TYPE(FileData_T), INTENT(INOUT) :: f
00888     TYPE(TimeData_T), INTENT(INOUT) :: t
00889
00890     REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
00891     LOGICAL :: timeFound ! true if current time is in array of stored times
00892
00893     INTEGER :: ndim ! number of dimensions in the netcdf file
00894     INTEGER :: nvar ! number of variables in the netcdf file
00895     INTEGER :: natt ! number of attributes in the netcdf file
00896
00897     INTEGER :: counti(1), starti(1)
00898     INTEGER :: ierr ! success or failure of netcdf call
00899     INTEGER :: i ! loop counter
00900
00901     CALL setmessagesource("SetRecordCounterAndStoreTime")

```

```

00903
00904 ! Inquire the time variable
00905 ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
00906 CALL netcdfcheckerr(ierr)
00907
00908 ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
00909 CALL netcdfcheckerr(ierr)
00910
00911 ierr = nf90_inq_varid(ncid, 'time', t%timeID)
00912 CALL netcdfcheckerr(ierr)
00913
00914 ! Determine the relationship between the current simulation time
00915 ! and the time array stored in the netcdf file. Set the record
00916 ! counter based on this relationship.
00917 IF (f%fileRecCounter /= 0) THEN
00918   ALLOCATE(storedtimes(f%fileRecCounter))
00919   ierr = nf90_get_var(ncid, t%timeID, storedtimes)
00920   CALL netcdfcheckerr(ierr)
00921   timefound = .false.
00922
00923   DO i = 1, f%fileRecCounter
00924     IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
00925       timefound = .true.
00926       EXIT
00927     ENDIF
00928   END DO
00929
00930   IF (timefound .EQV. .false.) THEN
00931     ! Increment the record counter so that we can store data at the
00932     ! next location in the netcdf file (i.e., all of the times
00933     ! in the netcdf file were found to be earlier than the current
00934     ! adcirc simulation time).
00935     f%fileRecCounter = f%fileRecCounter + 1
00936   ELSE
00937     ! set the counter at the index that reflects the
00938     ! current time within the netcdf file (or is between two times
00939     ! found in the netcdf file).
00940     ! WARNING: all subsequent data will remain in the file, we
00941     ! are just overwriting it ... if we don't overwrite all of it,
00942     ! the pre-existing data will still be there, which is probably
00943     ! not what the user intended ... but apparently there is no
00944     ! way to delete data from netcdf files:
00945     ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
00946     scratchformat = '("Overwriting pre-existing data in netcdf file ",a,' // &
00947     ' " for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.')"
00948     WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
00949     CALL allmessage(info, scratchmessage)
00950     f%fileRecCounter = i
00951   ENDIF
00952
00953   DEALLOCATE(storedtimes)
00954 ELSE
00955   ! set the counter at 1 so we can record our first time value
00956   f%fileRecCounter = 1
00957 ENDIF
00958
00959 ! Store simulation time in netcdf file
00960 starti(1) = f%fileRecCounter
00961 counti(1) = t%timeLen
00962 ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
00963 CALL netcdfcheckerr(ierr)
00964
00965 CALL unsetmessagesource()
00966
00967 END SUBROUTINE setrecordcounterandstoretime
00968
00969 !=====
00970
00971 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](#).

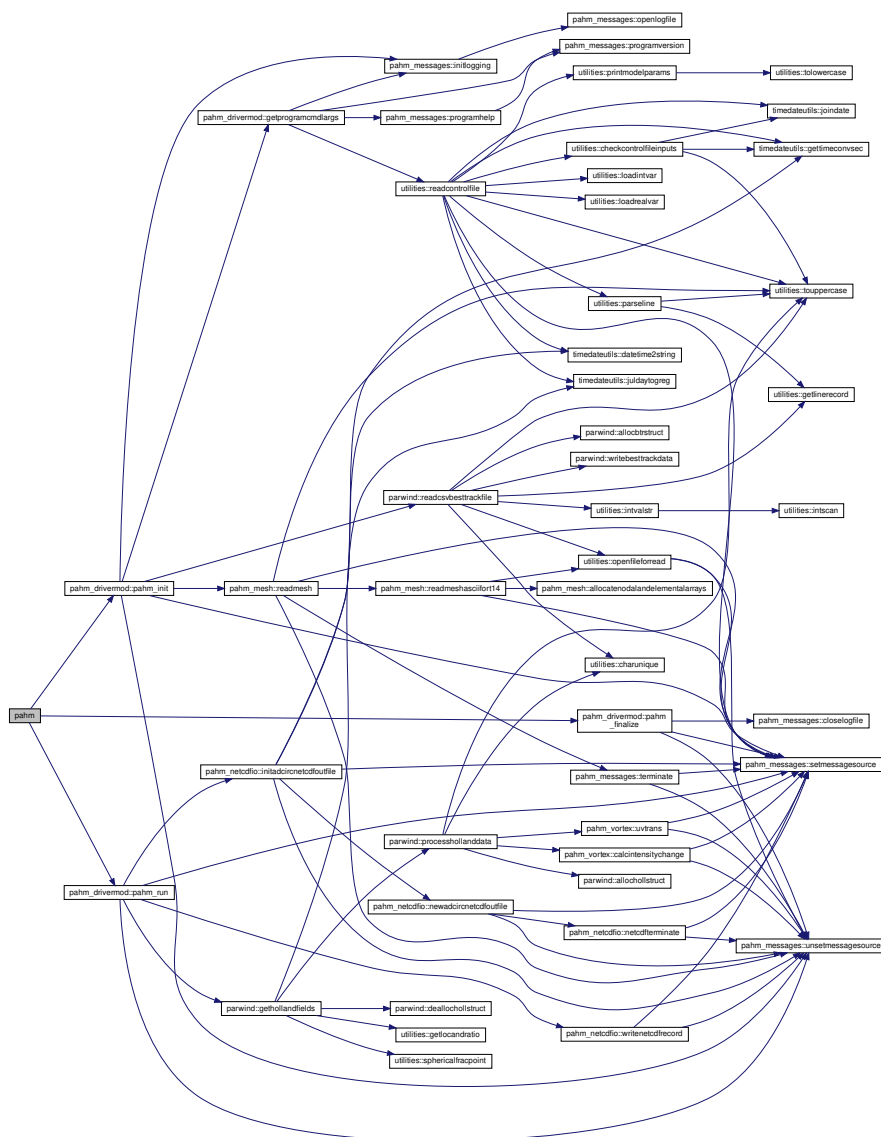
9.26.2 Function/Subroutine Documentation

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.](#)

```

00001 !-----
00002 !               P R O G R A M   P A H M
00003 !-----
00024 !-----
00025
00026 PROGRAM pahm
00027
00028     USE pahm_drivermod, ONLY : pahm_init, pahm_run, pahm_finalize
00029
00030     IMPLICIT NONE
00031
00032     CALL pahm_init ()

```

```
00033
00034  CALL pahm_run()
00035
00036  CALL pahm_finalize()
00037
00038 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::deallocchollstruct](#) (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.](#)

```

00001 !-----
00002 !               M O D U L E   P A R W I N D
00003 !-----
00006 !-----
00007
00008 MODULE parwind
00009
00010     USE pahm_sizes
00011     USE pahm_messages
00012
00013     ! switch to turn on or off geostrophic balance in GAHM
00014     ! on (default): Coriolis term included, phiFactors will be calculated before being used
00015     ! off          : parameter is set to 'TRUE', phiFactors will be set to constant 1
00016     !LOGICAL :: geostrophicSwitch = .TRUE.
00017     !INTEGER :: geoFactor = 1          !turn on or off gostrophic balance
00018
00019     REAL(sz) :: windreftime !jgf46.29 seconds since beginning of year, this          !PV check
00020                               !corresponds to time=0 of the simulation
00021
00022     !-----
00023     ! The BestTrackData_T structure holds all data read from the best track files(s)
00024     ! in ATCF format (a-deck/b-deck)
00025     !-----
00026     TYPE besttrackdata_t
00027     CHARACTER(LEN=FNAMELEN)      :: filename          ! full path to the best track file
00028     CHARACTER(LEN=10)            :: thisstorm         ! the name of the "named" storm
00029     LOGICAL                      :: loaded = .false.   ! .TRUE. if we have loaded the data from file
00030     INTEGER                      :: numrec            ! number of records in the structure
00031
00032     !----- input data from best track file (ATCF format)
00033     CHARACTER(LEN=2), ALLOCATABLE :: basin(:)         ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00034     INTEGER, ALLOCATABLE          :: cynum(:)         ! annual cyclone number: 1 - 99
00035     CHARACTER(LEN=10), ALLOCATABLE :: dtg(:)         ! warning Date-Time-Group (DTG), YYYYMMDDHH
00036     INTEGER, ALLOCATABLE          :: technum(:)       ! objective technique sorting number, minutes for
00037     best track: 00 - 99
00038     CHARACTER(LEN=4), ALLOCATABLE :: tech(:)         ! acronym for each objective technique or CARQ or
00039     WRNG,
00040     INTEGER, ALLOCATABLE          :: tau(:)          ! BEST for best track, up to 4 chars.
00041     ! forecast period: -24 through 240 hours, 0 for
00042     ! negative taus used for CARQ and WRNG records.
00043     INTEGER, ALLOCATABLE          :: intlat(:)       ! latitude for the DTG: 0 - 900 tenths of degrees
00044     INTEGER, ALLOCATABLE          :: intlon(:)       ! latitude for the DTG: 0 - 900 tenths of degrees
00045     CHARACTER(LEN=1), ALLOCATABLE :: ew(:)          ! E/W
00046     CHARACTER(LEN=1), ALLOCATABLE :: ns(:)          ! N/S
00047     INTEGER, ALLOCATABLE          :: intvmax(:)      ! maximum sustained wind speed in knots: 0 - 300
00048     kts
00049     INTEGER, ALLOCATABLE          :: intmslp(:)      ! minimum sea level pressure, 850 - 1050 mb
00050     CHARACTER(LEN=2), ALLOCATABLE :: ty(:)          ! Highest level of tc development:
00051     ! DB - disturbance,
00052     ! TD - tropical depression,
00053     ! TS - tropical storm,
00054     ! TY - typhoon,
00055     ! ST - super typhoon,
00056     ! TC - tropical cyclone,
00057     ! HU - hurricane,
00058     ! SD - subtropical depression,
00059     ! SS - subtropical storm,
00060     ! EX - extratropical systems,
00061     ! PT - post tropical,
00062     ! IN - inland,
00063     ! DS - dissipating,
00064     ! LO - low,
00065     ! WV - tropical wave,
00066     ! ET - extrapolated,
00067     ! MD - monsoon depression,
00068     ! XX - unknown.
00069     INTEGER, ALLOCATABLE          :: rad(:)          ! wind intensity for the radii defined in this
00070     record: 34, 50 or 64 kt
00071     CHARACTER(LEN=3), ALLOCATABLE :: windcode(:)    ! radius code:
00072     ! AAA - full circle
00073     ! NEQ, SEQ, SWQ, NWQ - quadrant
00074     INTEGER, ALLOCATABLE          :: intradl(:)      ! if full circle, radius of specified wind
00075     intensity, or radius of
00076     ! first quadrant wind intensity as specified by
00077     WINDCODE. 0 - 999 n mi

```

```

00073     INTEGER, ALLOCATABLE          :: intrad2(:)      ! if full circle this field not used, or radius
of 2nd quadrant wind
00074                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00075     INTEGER, ALLOCATABLE          :: intrad3(:)      ! if full circle this field not used, or radius
of 3rd quadrant wind
00076                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00077     INTEGER, ALLOCATABLE          :: intrad4(:)      ! if full circle this field not used, or radius
of 4th quadrant wind
00078                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00079     INTEGER, ALLOCATABLE          :: intpouter(:)     ! pressure in millibars of the last closed
isobar, 900 - 1050 mb
00080     INTEGER, ALLOCATABLE          :: introuter(:)    ! radius of the last closed isobar, 0 - 999 n mi
00081     INTEGER, ALLOCATABLE          :: intrmw(:)       ! radius of max winds, 0 - 999 n mi
00082     INTEGER, ALLOCATABLE          :: gusts(:)        ! gusts, 0 - 999 kt
00083     INTEGER, ALLOCATABLE          :: eye(:)          ! eye diameter, 0 - 120 n mi
00084     CHARACTER(LEN=3), ALLOCATABLE :: subregion(:)    ! subregion code: W,A,B,S,P,C,E,L,Q
00085                                     !   A - Arabian Sea
00086                                     !   B - Bay of Bengal
00087                                     !   C - Central Pacific
00088                                     !   E - Eastern Pacific
00089                                     !   L - Atlantic
00090                                     !   P - South Pacific (135E - 120W)
00091                                     !   Q - South Atlantic
00092                                     !   S - South IO (20E - 135E)
00093                                     !   W - Western Pacific
00094     INTEGER, ALLOCATABLE          :: maxseas(:)      ! max seas: 0 - 999 ft
00095     CHARACTER(LEN=3), ALLOCATABLE :: initials(:)    ! forecaster's initials used for tau 0 WRNG or
OFCL, up to 3 chars
00096     INTEGER, ALLOCATABLE          :: dir(:)         ! storm direction, 0 - 359 degrees
00097     INTEGER, ALLOCATABLE          :: intspeed(:)    ! storm speed, 0 - 999 kts
00098     CHARACTER(LEN=10), ALLOCATABLE :: stormname(:)  ! literal storm name, number, NONAME or INVEST,
or TCcyx where:
00099                                     !   cy = Annual cyclone number 01 - 99
00100                                     !   x  = Subregion code: W,A,B,S,P,C,E,L,Q.
00101     INTEGER, ALLOCATABLE          :: cyclenum(:)    ! the cycle number !PV check if this is OK
00102
00103 !   !----- converted data from the above values (if needed)
00104     INTEGER, DIMENSION(:), ALLOCATABLE :: year, month, day, hour
00105     REAL(sz), DIMENSION(:), ALLOCATABLE :: lat, lon
00106 END TYPE besttrackdata_t
00107
00108 ! Array of info about the best track data (extension to use multiple storms)
00109 TYPE(besttrackdata_t), ALLOCATABLE, TARGET :: besttrackdata(:)
00110
00111 !-----
00112 ! The HollandData_T structure holds all required data for the Holland model
00113 ! The data are filtered to only include unique DTGs
00114 !-----
00115 TYPE hollanddata_t
00116     CHARACTER(LEN=FNAMELEN)      :: filename        ! full path to the best track file
00117     CHARACTER(LEN=10)            :: thisstorm       ! the name of the "named" storm
00118     LOGICAL                      :: loaded = .false. ! .TRUE. if we have loaded the data from file
00119     INTEGER                      :: numrec          ! number of records in the structure
00120
00121     CHARACTER(LEN=2),            ALLOCATABLE :: basin(:)      ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00122     INTEGER, ALLOCATABLE          :: stormnumber(:)          ! annual cyclone number: 1 - 99
00123     CHARACTER(LEN=10),            ALLOCATABLE :: dtg(:)      ! warning Date-Time-Group (DTG), YYYYMMDDHH
00124     INTEGER, DIMENSION(:),        ALLOCATABLE :: year, month, day, hour
00125     REAL(sz), ALLOCATABLE          :: casttime(:)           ! converted to decimal E/N (lon, lat)
00126     CHARACTER(LEN=4),            ALLOCATABLE :: casttype(:) ! BEST, OFCL, CALM, ...
00127     INTEGER,                      ALLOCATABLE :: fctinc(:)   ! forecast period: -24 through 240 hours, 0
for best-track
00128
00129     INTEGER, DIMENSION(:),        ALLOCATABLE :: ilat, ilon  ! latitude, longitude for the GTD
00130     REAL(sz), DIMENSION(:),        ALLOCATABLE :: lat, lon    ! converted to decimal E/N (lon, lat)
00131
00132     INTEGER,                      ALLOCATABLE :: ispeed(:)    ! maximum sustained wind speed in knots: 0 -
300 kts
00133     REAL(sz),                      ALLOCATABLE :: speed(:)    ! converted from kts to m/s
00134
00135     INTEGER,                      ALLOCATABLE :: icpress(:)   ! minimum sea level pressure, 850 - 1050 mb
00136     REAL(sz),                      ALLOCATABLE :: cpress(:)   ! converted to Pa
00137
00138     INTEGER,                      ALLOCATABLE :: irrp(:)      ! radius of the last closed isobar, 0 - 999 n
mi
00139     REAL(sz),                      ALLOCATABLE :: rrp(:)      ! converted from nm to m
00140
00141     INTEGER,                      ALLOCATABLE :: irmw(:)      ! radius of max winds, 0 - 999 n mi

```

```

00142     REAL(sz),                ALLOCATABLE :: rmw(:)                ! converted from nm to m
00143
00144     REAL(sz), DIMENSION(:), ALLOCATABLE :: cprdt                ! central pressure intensity change (Pa / h)
00145     REAL(sz), DIMENSION(:), ALLOCATABLE :: trvx, trvy          ! translational velocity components (x, y) of
the                                     ! moving hurricane (m/s)
00146
00147     END TYPE hollanddata_t
00148
00149
00150     CONTAINS
00151
00152
00153     !-----
00154     !  SUBROUTINE  READ  BEST  TRACK  FILE
00155     !-----
00164     !-----
00165     SUBROUTINE readbesttrackfile()
00166
00167     USE pahm_global, ONLY : lun_btrk, lun_btrkl, nbtrfiles, besttrackfilename
00168     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
00169     USE sortutils, ONLY : arth, indexx, arrayequal
00170
00171     IMPLICIT NONE
00172
00173     CHARACTER(LEN=FNAMELEN)      :: inpFile
00174     CHARACTER(LEN=512)           :: inpLine, line
00175     CHARACTER(LEN=512)           :: fmtStr
00176
00177     INTEGER                      :: lenLine
00178     INTEGER                      :: nLines                ! Number of lines counter
00179     INTEGER                      :: iFile, iCnt            ! loop counters
00180     INTEGER                      :: iUnit, errIO, ios, status
00181
00182     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00183     INTEGER, ALLOCATABLE          :: idxArrStr(:)
00184     INTEGER                      :: nUnique, maxCnt, kCnt, kMax
00185
00186     INTEGER, ALLOCATABLE          :: idx0(:), idx1(:)
00187
00188     !----- Initialize variables
00189     iunit = lun_btrk
00190     errio = 0
00191
00192     fmtstr = ' (a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x,
a2, '
00193     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i4, 2x, i3, 2x, i4, 2x, i3, '
00194     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, i3, 2x, i3, 2x, a11, 2x, i3)'
00195     !-----
00196
00197     CALL setmessagesource("ReadBestTrackFile")
00198
00199     ! Allocate the best track structure array. This structure holds all the
00200     ! input values for the storm track as read in from the track input file
00201     ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00202     ! (as appropriate).
00203     ALLOCATE(besttrackdata(nbtrfiles))
00204
00205     ! This is the main loop. We loop through all the best track files
00206     ! (user input)
00207     DO ifile = 1, nbtrfiles
00208         inpfile = besttrackfilename(ifile)
00209
00210         CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)
00211
00212         IF (errio /= 0) THEN
00213             WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
00214             CALL allmessage(error, scratchmessage)
00215
00216             CALL unsetmessagesource()
00217
00218             CALL terminate()
00219         ELSE
00220             WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
00221             CALL logmessage(info, scratchmessage)
00222         END IF
00223
00224         besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00225         besttrackdata(ifile)%thisStorm = ""
00226         besttrackdata(ifile)%loaded = .false.
00227         besttrackdata(ifile)%numRec = -1
00228

```

```

00229      ! Count the number of non-empty or commented out lines in the file.
00230      ! Comments are are considered those lines with the first non-blank character of "!" or "#"
00231      nlines = 0
00232      DO
00233          READ(unit=iunit, fmt='(a)', err=10, END=5, IOSTAT=errIO) inpline
00234
00235          lenline = getlinerecord(inpline, line)
00236          IF (lenline /= 0) nlines = nlines + 1
00237      END DO
00238      5 rewind(unit=iunit)
00239
00240      ! Array allocation in the structure bestTrackData
00241      CALL allocbtrstruct(besttrackdata(ifile), nlines)
00242
00243      icnt = 0
00244      DO WHILE (.true.)
00245          READ(unit=iunit, fmt='(a)', err=10, END=20, IOSTAT=errIO) inpline
00246
00247          lenline = getlinerecord(inpline, line)
00248
00249          IF (lenline /= 0) THEN
00250              icnt = icnt + 1
00251              READ(line, fmt=fmtstr, err=11, iostat=ios)
00252              besttrackdata(ifile)%basin(icnt),      besttrackdata(ifile)%cyNum(icnt),      &
00253              besttrackdata(ifile)%dtg(icnt),        besttrackdata(ifile)%techNum(icnt),      &
00254              besttrackdata(ifile)%tech(icnt),       besttrackdata(ifile)%tau(icnt),      &
00255              besttrackdata(ifile)%intLat(icnt),     besttrackdata(ifile)%ns(icnt),      &
00256              besttrackdata(ifile)%intLon(icnt),     besttrackdata(ifile)%ew(icnt),      &
00257              besttrackdata(ifile)%intVMax(icnt),    besttrackdata(ifile)%intMslp(icnt), &
00258              besttrackdata(ifile)%ty(icnt),         besttrackdata(ifile)%rad(icnt),     &
00259              besttrackdata(ifile)%windCode(icnt),   besttrackdata(ifile)%intRad1(icnt), &
00260              besttrackdata(ifile)%intRad2(icnt),    besttrackdata(ifile)%intRad3(icnt), &
00261              besttrackdata(ifile)%intRad4(icnt),    besttrackdata(ifile)%intPOuter(icnt), &
00262              besttrackdata(ifile)%intROuter(icnt),  besttrackdata(ifile)%intRmw(icnt),  &
00263              besttrackdata(ifile)%gusts(icnt),      besttrackdata(ifile)%eye(icnt),     &
00264              besttrackdata(ifile)%subregion(icnt),  besttrackdata(ifile)%maxseas(icnt), &
00265              besttrackdata(ifile)%initials(icnt),   besttrackdata(ifile)%dir(icnt),     &
00266              besttrackdata(ifile)%intSpeed(icnt),   besttrackdata(ifile)%stormName(icnt), &
00267              besttrackdata(ifile)%cycleNum(icnt)
00268
00269              !----- Convert lat/lon values to S/N and W/E notations
00270              IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
00271                  besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00272              ELSE
00273                  besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00274              END IF
00275
00276              IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00277                  besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00278              ELSE
00279                  besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00280              END IF
00281              !-----
00282
00283              !----- Get the year, month, day, hour from the DGT string
00284              READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00285              besttrackdata(ifile)%year(icnt)
00286              IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00287              READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00288              besttrackdata(ifile)%month(icnt)
00289              IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00290              READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00291              besttrackdata(ifile)%day(icnt)
00292              IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00293              READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00294              besttrackdata(ifile)%hour(icnt)
00295              IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00296              !-----
00297              END IF
00298          END DO
00299
00300          10 IF (errIO /= 0) THEN
00301              WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00302              ' ', while processing line: ' // trim(adjustl(inpline))
00303              CALL allmessage(error, scratchmessage)
00304
00305              CLOSE(iunit)
00306
00307              CALL unsetmessagesource()
00308
00309              CALL terminate()

```

```

00306         END IF
00307
00308     11 IF (ios /= 0) THEN
00309         WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00310             ', while processing line: ' // trim(adjustl(line))
00311         CALL allmessage(error, scratchmessage)
00312
00313         CLOSE(iunit)
00314
00315         CALL unsetmessagesource()
00316
00317         CALL terminate()
00318     END IF
00319
00320 20 CLOSE(iunit)
00321
00322     besttrackdata(ifile)%thisStorm = "
00323     besttrackdata(ifile)%loaded    = .true.
00324     besttrackdata(ifile)%numRec    = nlines
00325
00326
00327     !-----
00328     ! Get the unique storm name and store it in the thisStorm string
00329     ALLOCATE(chkarrstr(nlines))
00330     ALLOCATE(idxarrstr(nlines))
00331
00332     nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00333
00334     maxcnt = -1
00335     DO kcnt = 1, nunique
00336         kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00337         IF (kmax > maxcnt) THEN
00338             maxcnt = kmax
00339             besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00340         END IF
00341     END DO
00342
00343     DEALLOCATE(chkarrstr)
00344     DEALLOCATE(idxarrstr)
00345     !-----
00346
00347     !-----
00348     ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00349     ! stored in ascending order
00350     ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00351     ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00352
00353     CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00354
00355     IF (status /= 0) THEN
00356         CALL unsetmessagesource()
00357
00358         CALL terminate()
00359     END IF
00360
00361     ! Create the index array to be used in the comparison below
00362     idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00363
00364     IF (.NOT. arrayequal(idx0, idx1)) THEN
00365         besttrackdata(ifile)%basin = besttrackdata(ifile)%basin(idx1)
00366         besttrackdata(ifile)%cyNum = besttrackdata(ifile)%cyNum(idx1)
00367         besttrackdata(ifile)%dtg   = besttrackdata(ifile)%dtg(idx1)
00368         besttrackdata(ifile)%techNum = besttrackdata(ifile)%techNum(idx1)
00369         besttrackdata(ifile)%tech   = besttrackdata(ifile)%tech(idx1)
00370         besttrackdata(ifile)%tau     = besttrackdata(ifile)%tau(idx1)
00371         besttrackdata(ifile)%intLat  = besttrackdata(ifile)%intLat(idx1)
00372         besttrackdata(ifile)%ns     = besttrackdata(ifile)%ns(idx1)
00373         besttrackdata(ifile)%intLon  = besttrackdata(ifile)%intLon(idx1)
00374         besttrackdata(ifile)%ew     = besttrackdata(ifile)%ew(idx1)
00375         besttrackdata(ifile)%intVMax = besttrackdata(ifile)%intVMax(idx1)
00376         besttrackdata(ifile)%intMslp = besttrackdata(ifile)%intMslp(idx1)
00377         besttrackdata(ifile)%ty     = besttrackdata(ifile)%ty(idx1)
00378         besttrackdata(ifile)%rad     = besttrackdata(ifile)%rad(idx1)
00379         besttrackdata(ifile)%windCode = besttrackdata(ifile)%windCode(idx1)
00380         besttrackdata(ifile)%intRad1 = besttrackdata(ifile)%intRad1(idx1)
00381         besttrackdata(ifile)%intRad2 = besttrackdata(ifile)%intRad2(idx1)
00382         besttrackdata(ifile)%intRad3 = besttrackdata(ifile)%intRad3(idx1)
00383         besttrackdata(ifile)%intRad4 = besttrackdata(ifile)%intRad4(idx1)
00384         besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idx1)
00385         besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idx1)
00386         besttrackdata(ifile)%intRmw   = besttrackdata(ifile)%intRmw(idx1)

```

```

00387     besttrackdata(ifile)%gusts      = besttrackdata(ifile)%gusts(idxl)
00388     besttrackdata(ifile)%eye        = besttrackdata(ifile)%eye(idxl)
00389     besttrackdata(ifile)%subregion  = besttrackdata(ifile)%subregion(idxl)
00390     besttrackdata(ifile)%maxseas    = besttrackdata(ifile)%maxseas(idxl)
00391     besttrackdata(ifile)%initials   = besttrackdata(ifile)%initials(idxl)
00392     besttrackdata(ifile)%dir        = besttrackdata(ifile)%dir(idxl)
00393     besttrackdata(ifile)%intSpeed    = besttrackdata(ifile)%intSpeed(idxl)
00394     besttrackdata(ifile)%stormName  = besttrackdata(ifile)%stormName(idxl)
00395     besttrackdata(ifile)%cycleNum   = besttrackdata(ifile)%cycleNum(idxl)
00396     END IF
00397
00398     DEALLOCATE(idx0)
00399     DEALLOCATE(idxl)
00400     !-----
00401
00402     CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
00403     END DO ! End of "iFile" loop
00404
00405     CALL unsetmessagesource()
00406
00407 END SUBROUTINE readbesttrackfile
00408
00409 SUBROUTINE readcsvbesttrackfile()
00410
00411     USE pahn_global, ONLY : nbtrfiles, besttrackfilename
00412     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique, &
00413                          intvalstr
00414     USE sortutils, ONLY : arth, indexx, arrayequal
00415     USE csv_module
00416
00417     IMPLICIT NONE
00418
00419     TYPE(csv_file)                :: f
00420     CHARACTER(LEN=64), ALLOCATABLE :: sval2D(:, :)
00421     LOGICAL                        :: statusOK
00422
00423     CHARACTER(LEN=FNAMELEN)        :: inpFile
00424     CHARACTER(LEN=512)             :: line
00425     CHARACTER(LEN=64)              :: tmpStr
00426
00427     INTEGER                        :: iFile, nLines, lenLine
00428     INTEGER                        :: iCnt, jCnt, kCnt, kMax      ! loop counters
00429     INTEGER                        :: ios, status
00430
00431     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00432     INTEGER, ALLOCATABLE           :: idxArrStr(:)
00433     INTEGER                        :: nUnique, maxCnt
00434
00435     INTEGER, ALLOCATABLE           :: idx0(:), idx1(:)
00436
00437
00438     CALL setmessagesource("ReadCsvBestTrackFile")
00439
00440     ! Allocate the best track structure array. This structure holds all the
00441     ! input values for the storm track as read in from the track input file
00442     ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00443     ! (as appropriate).
00444     ALLOCATE(besttrackdata(nbtrfiles))
00445
00446     ! This is the main loop. We loop through all the best track files
00447     ! (user input)
00448     DO ifile = 1, nbtrfiles
00449         inpfile = besttrackfilename(ifile)
00450
00451         besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00452         besttrackdata(ifile)%thisStorm = ""
00453         besttrackdata(ifile)%loaded   = .false.
00454         besttrackdata(ifile)%numRec   = -1
00455
00456         CALL f%Read(trim(adjustl(inpfile)), status_ok=statusok)
00457         CALL f%Get(sval2d, status_ok=statusok)
00458
00459         ! Array allocation in the structure bestTrackData
00460         nlines = f%n_rows
00461         CALL allocbtrstruct(besttrackdata(ifile), nlines)
00462
00463         DO icnt = 1, nlines
00464             DO jcnt = 1, f%n_cols
00465                 line = line // trim(adjustl(sval2d(icnt, jcnt)))
00466             END DO
00467             jcnt = 0

```

```

00468
00469     lenline = len_trim(adjustl(line))
00470
00471     IF (lenline /= 0) THEN
00472         !--- col: 1
00473         besttrackdata(ifile)%basin(icnt) = trim(adjustl(sval2d(icnt, 1)))
00474         !--- col: 2
00475         besttrackdata(ifile)%cyNum(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 2))))
00476         !--- col: 3
00477         besttrackdata(ifile)%dtg(icnt) = trim(adjustl(sval2d(icnt, 3)))
00478         !--- col: 4
00479         besttrackdata(ifile)%techNum(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 4))))
00480         !--- col: 5
00481         besttrackdata(ifile)%tech(icnt) = trim(adjustl(sval2d(icnt, 5)))
00482         !--- col: 6
00483         besttrackdata(ifile)%tau(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 6))))
00484         !--- col: 7
00485         tmpstr = trim(adjustl(sval2d(icnt, 7)))
00486         READ(tmpstr, '(i3, a1)') &
00487             besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
00488         !--- col: 8
00489         tmpstr = trim(adjustl(sval2d(icnt, 8)))
00490         READ(tmpstr, '(i3, a1)') &
00491             besttrackdata(ifile)%intLon(icnt), besttrackdata(ifile)%ew(icnt)
00492         !--- col: 9
00493         besttrackdata(ifile)%intVMax(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 9))))
00494         !--- col: 10
00495         besttrackdata(ifile)%intMslp(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 10))))
00496         !--- col: 11
00497         besttrackdata(ifile)%ty(icnt) = trim(adjustl(sval2d(icnt, 11)))
00498         !--- col: 12
00499         besttrackdata(ifile)%rad(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 12))))
00500         !--- col: 13
00501         besttrackdata(ifile)%windCode(icnt) = trim(adjustl(sval2d(icnt, 13)))
00502         !--- col: 14
00503         besttrackdata(ifile)%intRad1(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 14))))
00504         !--- col: 15
00505         besttrackdata(ifile)%intRad2(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 15))))
00506         !--- col: 16
00507         besttrackdata(ifile)%intRad3(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 16))))
00508         !--- col: 17
00509         besttrackdata(ifile)%intRad4(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 17))))
00510         !--- col: 18
00511         besttrackdata(ifile)%intPOuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 18))))
00512         !--- col: 19
00513         besttrackdata(ifile)%intROuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 19))))
00514         !--- col: 20
00515         besttrackdata(ifile)%intRmw(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 20))))
00516         !--- col: 21
00517         besttrackdata(ifile)%gusts(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 21))))
00518         !--- col: 22
00519         besttrackdata(ifile)%eye(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 22))))
00520         !--- col: 23
00521         besttrackdata(ifile)%subregion(icnt) = trim(adjustl(sval2d(icnt, 23)))
00522         !--- col: 24
00523         besttrackdata(ifile)%maxseas(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 24))))
00524         !--- col: 25
00525         besttrackdata(ifile)%initials(icnt) = trim(adjustl(sval2d(icnt, 25)))
00526         !--- col: 26
00527         besttrackdata(ifile)%dir(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 26))))
00528         !--- col: 27
00529         besttrackdata(ifile)%intSpeed(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 27))))
00530         !--- col: 28
00531         besttrackdata(ifile)%stormName(icnt) = trim(adjustl(sval2d(icnt, 28)))
00532
00533         ! This is for the cycleNum, the last column we consider
00534         IF (icnt == 1) THEN
00535             kcmt = icnt
00536             besttrackdata(ifile)%cycleNum(icnt) = icnt
00537         ELSE
00538             kcmt = kcmt + 1
00539             IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00540                 besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00541                 kcmt = kcmt - 1
00542             ELSE
00543                 besttrackdata(ifile)%cycleNum(icnt) = kcmt
00544             END IF
00545         END IF
00546
00547         !----- Convert lat/lon values to S/N and W/E notations
00548         IF (toupper(besttrackdata(ifile)%ns(icnt)) == 'S') THEN

```

```

00549         besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00550     ELSE
00551         besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00552     END IF
00553
00554     IF (toupper(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00555         besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00556     ELSE
00557         besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00558     END IF
00559     !-----
00560
00561     !----- Get the year, month, day, hour from the DGT string
00562     READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00563     besttrackdata(ifile)%year(icnt)
00564     IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00565     READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00566     besttrackdata(ifile)%month(icnt)
00567     IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00568     READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00569     besttrackdata(ifile)%day(icnt)
00570     IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00571     READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00572     besttrackdata(ifile)%hour(icnt)
00573     IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00574     !-----
00575     END IF
00576     END DO
00577
00578     besttrackdata(ifile)%thisStorm = "
00579     besttrackdata(ifile)%loaded = .true.
00580     besttrackdata(ifile)%numRec = nlines
00581
00582     !-----
00583     ! Get the unique storm name and store it in the thisStorm string
00584     ALLOCATE(chkarrstr(nlines))
00585     ALLOCATE(idxarrstr(nlines))
00586
00587     nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00588
00589     maxcnt = -1
00590     DO kcnt = 1, nunique
00591         kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00592         IF (kmax > maxcnt) THEN
00593             maxcnt = kmax
00594             besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00595         END IF
00596     END DO
00597
00598     DEALLOCATE(chkarrstr)
00599     DEALLOCATE(idxarrstr)
00600
00601     !-----
00602     ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00603     ! stored in ascending order
00604     ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00605     ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00606
00607     CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00608
00609     IF (status /= 0) THEN
00610         CALL unsetmessagesource()
00611         CALL terminate()
00612     END IF
00613
00614     ! Create the index array to be used in the comparison below
00615     idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00616
00617     IF (.NOT. arrayequal(idx0, idx1)) THEN
00618         besttrackdata(ifile)%basin = besttrackdata(ifile)%basin(idx1)
00619         besttrackdata(ifile)%cyNum = besttrackdata(ifile)%cyNum(idx1)
00620         besttrackdata(ifile)%dtg = besttrackdata(ifile)%dtg(idx1)
00621         besttrackdata(ifile)%techNum = besttrackdata(ifile)%techNum(idx1)
00622         besttrackdata(ifile)%tech = besttrackdata(ifile)%tech(idx1)
00623         besttrackdata(ifile)%tau = besttrackdata(ifile)%tau(idx1)
00624         besttrackdata(ifile)%intLat = besttrackdata(ifile)%intLat(idx1)
00625         besttrackdata(ifile)%ns = besttrackdata(ifile)%ns(idx1)
00626         besttrackdata(ifile)%intLon = besttrackdata(ifile)%intLon(idx1)

```



```

00626     besttrackdata(ifile)%ew      = besttrackdata(ifile)%ew(idx1)
00627     besttrackdata(ifile)%intVMax  = besttrackdata(ifile)%intVMax(idx1)
00628     besttrackdata(ifile)%intMslp  = besttrackdata(ifile)%intMslp(idx1)
00629     besttrackdata(ifile)%ty       = besttrackdata(ifile)%ty(idx1)
00630     besttrackdata(ifile)%rad      = besttrackdata(ifile)%rad(idx1)
00631     besttrackdata(ifile)%windCode = besttrackdata(ifile)%windCode(idx1)
00632     besttrackdata(ifile)%intRad1  = besttrackdata(ifile)%intRad1(idx1)
00633     besttrackdata(ifile)%intRad2  = besttrackdata(ifile)%intRad2(idx1)
00634     besttrackdata(ifile)%intRad3  = besttrackdata(ifile)%intRad3(idx1)
00635     besttrackdata(ifile)%intRad4  = besttrackdata(ifile)%intRad4(idx1)
00636     besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idx1)
00637     besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idx1)
00638     besttrackdata(ifile)%intRmw   = besttrackdata(ifile)%intRmw(idx1)
00639     besttrackdata(ifile)%gusts    = besttrackdata(ifile)%gusts(idx1)
00640     besttrackdata(ifile)%eye      = besttrackdata(ifile)%eye(idx1)
00641     besttrackdata(ifile)%subregion = besttrackdata(ifile)%subregion(idx1)
00642     besttrackdata(ifile)%maxseas  = besttrackdata(ifile)%maxseas(idx1)
00643     besttrackdata(ifile)%initials = besttrackdata(ifile)%initials(idx1)
00644     besttrackdata(ifile)%dir      = besttrackdata(ifile)%dir(idx1)
00645     besttrackdata(ifile)%intSpeed = besttrackdata(ifile)%intSpeed(idx1)
00646     besttrackdata(ifile)%stormName = besttrackdata(ifile)%stormName(idx1)
00647     besttrackdata(ifile)%cycleNum = besttrackdata(ifile)%cycleNum(idx1)
00648     END IF
00649
00650     DEALLOCATE(idx0)
00651     DEALLOCATE(idx1)
00652     !-----
00653
00654     CALL f%Destroy()
00655
00656     CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
00657
00658     END DO ! End of "iFile" loop
00659
00660     CALL unsetmessagesource()
00661
00662     END SUBROUTINE readcsvbesttrackfile
00663
00664     !=====
00665     !-----
00666     ! SUBROUTINE PROCESS HOLLAND DATA
00667     !-----
00668     ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00669     !-----
00670     SUBROUTINE processhollanddata(idTrFile, strOut, status)
00671
00672     USE pahn_global, ONLY : nm2m, kt2ms, nbtrfiles
00673     USE utilities, ONLY : touppercase, charunique
00674     USE timedateutils, ONLY : timeconv
00675     USE pahn_vortex, ONLY : calcintensitychange, uvtrans
00676
00677     IMPLICIT NONE
00678
00679     INTEGER, INTENT(IN) :: idTrFile
00680     TYPE(hollanddata_t), INTENT(OUT) :: strOut
00681     INTEGER, INTENT(OUT) :: status ! error status
00682
00683     ! numUniqRec, outDTG, idxDTG are used to identify the unique DTG elements in the input structure
00684     INTEGER :: numUniqRec
00685     CHARACTER(LEN=10), ALLOCATABLE :: outDTG(:)
00686     INTEGER, ALLOCATABLE :: idxDTG(:)
00687
00688     INTEGER :: plIdx ! populated index for Holland Data array
00689     INTEGER :: iCnt ! loop counters
00690
00691     CHARACTER(LEN=4) :: castType !hindcast,forecast
00692     REAL(SZ), ALLOCATABLE :: castTime(:) ! seconds since start of year
00693
00694     REAL(SZ) :: spdVal, pressVal, rrpVal, rmwVal
00695
00696     status = 0 ! no error
00697
00698     CALL setmessagesource("ProcessHollandData")
00699
00700     IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
00701         IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
00702             status = 2
00703
00704             WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ',

```

```

idtrfile
00714     CALL allmessage(error, scratchmessage)
00715
00716     CALL unsetmessagesource()
00717
00718     RETURN
00719 END IF
00720 ELSE
00721     status = 1
00722
00723     WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
00724                                     ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ',
nBTrFiles
00725     CALL allmessage(error, scratchmessage)
00726
00727     CALL unsetmessagesource()
00728
00729     RETURN
00730 END IF
00731
00732 WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
00733 CALL logmessage(info, scratchmessage)
00734
00735 ! Most likely the array size will be larger if repeated times are found
00736 ! in the best track structure.
00737 ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
00738 ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))
00739
00740 ! Get unique lines that represent new points in time.
00741 ! Repeated time points occur in hindcasts for the purpose of
00742 ! describing winds in the quadrants of the storm. We don't use the
00743 ! quadrant-by-quadrant wind data. Repeated time data occur in the
00744 ! forecast because the time data is just the time that the forecast
00745 ! was made. The important parameter in the forecast file is the
00746 ! forecast increment.
00747 numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)
00748
00749 !-----
00750 ! Populate the Holland structure
00751 !-----
00752 CALL allochollstruct(strout, numuniqrec)
00753
00754 ALLOCATE(casttime(numuniqrec))
00755
00756 strout%fileName = besttrackdata(idtrfile)%fileName
00757 strout%thisStorm = besttrackdata(idtrfile)%thisStorm
00758 strout%loaded = .true.
00759 strout%numRec = numuniqrec
00760
00761 WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
00762 CALL logmessage(info, scratchmessage)
00763
00764 DO icnt = 1, numuniqrec
00765     plidx = idxdtg(icnt)
00766
00767     casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(plidx))))
00768
00769     ! Convert speeds from knots to m/s
00770     spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)
00771
00772     ! Convert pressure(s) from mbar to Pa
00773     pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)
00774
00775     ! Convert all distances from nm to km/m
00776     rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
00777     rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx)    ! in m
00778
00779     strout%basin(icnt) = besttrackdata(idtrfile)%basin(plidx)
00780     strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)
00781     strout%dtg(icnt) = besttrackdata(idtrfile)%dtg(plidx)
00782     strout%year(icnt) = besttrackdata(idtrfile)%year(plidx)
00783     strout%month(icnt) = besttrackdata(idtrfile)%month(plidx)
00784     strout%day(icnt) = besttrackdata(idtrfile)%day(plidx)
00785     strout%hour(icnt) = besttrackdata(idtrfile)%hour(plidx)
00786     strout%castType(icnt) = besttrackdata(idtrfile)%tech(plidx)
00787     strout%fcstInc(icnt) = besttrackdata(idtrfile)%tau(plidx)
00788     strout%iLat(icnt) = besttrackdata(idtrfile)%intLat(plidx)
00789     strout%lat(icnt) = besttrackdata(idtrfile)%lat(plidx)
00790     strout%iLon(icnt) = besttrackdata(idtrfile)%intLon(plidx)
00791     strout%lon(icnt) = besttrackdata(idtrfile)%lon(plidx)
00792

```

```

00793      strout%iSpeed(icnt)      = besttrackdata(idtrfile)%intVMax(plidx)
00794      strout%speed(icnt)       = spdval
00795      strout%iCPress(icnt)     = besttrackdata(idtrfile)%intMslp(plidx)
00796      strout%cPress(icnt)      = pressval
00797      strout%iRrp(icnt)       = besttrackdata(idtrfile)%intROuter(plidx)
00798      strout%rrp(icnt)        = rrpval
00799      strout%iRmw(icnt)       = besttrackdata(idtrfile)%intRmw(plidx)
00800      strout%rmw(icnt)        = rmwval
00801
00802      ! PV check if this SELECT code is actually needed. Need to check the different format
00803      ! of input files.
00804      SELECT CASE(casttype)
00805      CASE("BEST")              ! nowcast/hindcast
00806      ! PV check if this is needed
00807      CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0,
0.0_sz, casttime(icnt))
00808
00809      CASE("OFCL")              ! forecast
00810      ! PV check if this is needed
00811      IF (icnt > 1) THEN
00812      IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
cycle
00813      END IF
00814
00815      IF (strout%fcstInc(icnt) == 0) THEN
00816      CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
00817      strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
00818      ELSE
00819      casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) *
3600.0_sz)
00820      END IF
00821
00822      IF ((strout%iCPress(icnt) == 0) .OR. (strout%iRmw(icnt) == 0)) THEN
00823      CALL allmessage(error,
00824      'The storm hindcast/forecast input file ' // trim(strout%fileName) // &
00825      ' contains invalid data for central pressure or rMax.')
00826      CALL terminate()
00827      END IF
00828
00829      ! Adding a new type to allow the analyst to add lines
00830      ! that do nothing but produce zero winds and background barometric
00831      ! pressure. These lines can have a date/time like a BEST line or
00832      ! a date/time and forecast period like an OFCL line.
00833      CASE("CALM")
00834      ! PV check if this is needed
00835      WRITE(scratchmessage, '(a)') 'The file: ' // trim(strout%fileName) // ' contains at least one
"CALM" line.'
00836      CALL logmessage(echo, scratchmessage)
00837
00838      IF (icnt > 1) THEN
00839      IF ( (strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
cycle
00840      END IF
00841
00842      IF (strout%fcstInc(icnt) == 0) THEN
00843      CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
00844      strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
00845      ELSE
00846      casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) *
3600.0_sz)
00847      END IF
00848
00849      CASE DEFAULT              ! unrecognized
00850      WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of '
// &
00851      trim(adjustl(strout%fileName))
00852      CALL allmessage(error, scratchmessage)
00853
00854      CALL terminate()
00855      END SELECT
00856
00857      strout%castTime(icnt) = casttime(icnt)
00858      END DO ! numUniqRec
00859
00860      ! Calculate the cPress intensity change (dP/dt)
00861      CALL calcintensitychange(strout%cPress, casttime, strout%cPrDt, status, 2)
00862
00863      ! Calculate storm translation velocities based on change in position,
00864      ! approximate u and v translation velocities
00865      CALL uvtrans(strout%lat, strout%lon, casttime, strout%trVx, strout%trVy, status, 2)
00866

```

```

00867      DEALLOCATE(casttime)
00868      !-----
00869
00870      DEALLOCATE(outdtg)
00871      DEALLOCATE(idxdtg)
00872
00873      CALL unsetmessagesource()
00874
00875      END SUBROUTINE processhollanddata
00876
00877      !-----
00878
00879      !-----
00880      ! SUBROUTINE GET HOLLAND FIELDS
00881      !-----
00882      ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
00883      !-----
00897      SUBROUTINE gethollandfields()
00898
00899      USE pahm_mesh, ONLY : slam, sfea, xcslam, ycsfea, np, ismeshok
00900      USE pahm_global, ONLY : gravity, rhowater, rhoair, &
00901                          backgroundatmpress, bladjustfac, one2ten, &
00902                          deg2rad, rad2deg, basee, omega, mb2pa, mb2kpa, &
00903                          nbtrfiles, besttrackfilename, &
00904                          noutdt, mdbegsimtime, mdendsimtime, mdoutdt, &
00905                          wvelx, wvely, wpress, times
00906      USE utilities, ONLY : sphericaldistance, sphericalfracpoint, getlocandratio
00907      USE timedateutils, ONLY : juldaytogreg, gregtojulday
00908      USE pahm_netcdfio
00909
00910      IMPLICIT NONE
00911
00912      TYPE(hollanddata_t), ALLOCATABLE :: holStru(:) ! array of Holland data structures
00913      INTEGER :: stormNumber ! storm identification number
00914      REAL(SZ) :: h1B ! Holland B parameter
00915      REAL(SZ) :: rrp ! radius of the last closed isobar (m)
00916      REAL(SZ) :: rmw ! radius of max winds (m)
00917      REAL(SZ) :: speed ! maximum sustained wind speed (m/s)
00918      REAL(SZ) :: cPress ! central pressure (Pa)
00919      REAL(SZ) :: cPressDef ! pressure deficit: Ambient Press - cPress
00920      REAL(SZ) :: trVX, trVY, trSPD ! storm translation velocities (m/s)
00921      REAL(SZ) :: trSpdX, trSpdY ! adjusted translation velocities (m/s)
00922      REAL(SZ) :: lon, lat ! current eye location
00923
00924      REAL(SZ), ALLOCATABLE :: rad(:) ! distance of nodal points from the eye
00925      location
00926      INTEGER, ALLOCATABLE :: radIDX(:) ! indices of nodal points such that rad <=
00927      rrp
00928      INTEGER :: maxRadIDX ! total number of radIDX elements
00929      REAL(SZ) :: windMultiplier ! for storm 2 in lpfs ensemble DO WE NEED
00930      THIS?
00931      REAL(SZ) :: dx, dy, theta
00932      REAL(SZ) :: wtRatio
00933      REAL(SZ) :: coriolis
00934
00935      REAL(SZ) :: sfPress ! calculated surface MSL pressure (Pa)
00936      REAL(SZ) :: grVel ! wind speed (m/s) at gradient level (top
00937      of ABL)
00938      REAL(SZ) :: sfVelX, sfVelY ! calculated surface (10-m above ground)
00939      wind velocities (m/s)
00940
00941      INTEGER :: iCnt, stCnt, npCnt
00942      INTEGER :: i, j11, j12
00943      INTEGER :: status
00944
00945      CHARACTER(LEN=64) :: tmpTimeStr, tmpStr1, tmpStr2
00946
00947      CALL setmessagesource("GetHollandFields")
00948
00949      ! Check if the mesh variables are set and that nOutDT is greater than zero.
00950      IF (.NOT. ismeshok) THEN
00951          WRITE(scratchmessage, '(a)') 'The mesh variables are not established properly. ' // &
00952          'Call subroutine ReadMesh to read/create the mesh topology first.'
00953          CALL allmessage(error, scratchmessage)
00954          CALL unsetmessagesource()
00955          CALL terminate()
00956      ELSE

```

```

00955     IF ((np <= 0) .OR. (noutdt <= 0)) THEN
00956         WRITE(tmpstr1, '(a, i0)') 'np = ', np
00957         WRITE(tmpstr2, '(a, i0)') 'noutDT = ', noutdt
00958         WRITE(scratchmessage, '(a)') 'Variables "np" or "noutDT" are not defined properly: ' // &
00959             trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00960         CALL allmessage(error, scratchmessage)
00961
00962         CALL unsetmessagesource()
00963
00964         CALL terminate()
00965     END IF
00966 END IF
00967
00968 !-----
00969 ! Allocate storage for required arrays and set the Times array
00970 ! that contains the output times.
00971 ALLOCATE(times(noutdt))
00972 ALLOCATE(wvelx(np, noutdt), wvely(np, noutdt), wpress(np, noutdt))
00973
00974 !wVelX = RMISSV
00975 !wVelY = wVelX; wPress = wVelX
00976 wvelx = 0.0_sz
00977 wvely = wvelx
00978 wpress = backgroundatmpress * mb2pa
00979 DO icnt = 1, noutdt
00980     times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
00981 END DO
00982 !-----
00983
00984 !-----
00985 ! ALLOCATE THE HOLLAND DATA STRUCTURES AND STORE THE HOLLAND
00986 ! DATA INTO THE DATA STRUCTURE ARRAY FOR SUBSEQUENT USE
00987 !-----
00988 !
00989 ! Allocate the array of Holland data structures. The Holland
00990 ! structures are allocated by calling the ProcessHollandData
00991 ! subroutine.
00992 ALLOCATE(holstru(nbtrfiles))
00993
00994 ! Process and store the "best track" data into the array of Holland structures
00995 ! for subsequent use. All required data to generate the P-W model wind fields
00996 ! are contained in these structures. We take into consideration that might be
00997 ! more than one "best track" file for the simulation period.
00998 DO stcnt = 1, nbtrfiles
00999     CALL processhollanddata(stcnt, holstru(stcnt), status)
01000
01001     IF (.NOT. holstru(stcnt)%loaded) THEN
01002         WRITE(scratchmessage, '(a)') 'There was an error loading the Holland data structure for the best
track file: ' // &
01003             trim(adjustl(besttrackfilename(stcnt)))
01004         CALL allmessage(error, scratchmessage)
01005
01006         CALL deallochoollstruct(holstru(stcnt))
01007         DEALLOCATE(holstru)
01008
01009         CALL unsetmessagesource()
01010
01011         CALL terminate()
01012     ELSE IF (status /= 0) THEN
01013         WRITE(scratchmessage, '(a)') 'There was an error processing the Holland data structure for the
best track file: ' // &
01014             trim(adjustl(besttrackfilename(stcnt)))
01015         CALL allmessage(error, scratchmessage)
01016
01017         CALL deallochoollstruct(holstru(stcnt))
01018         DEALLOCATE(holstru)
01019
01020         CALL unsetmessagesource()
01021
01022         CALL terminate()
01023     ELSE
01024         WRITE(scratchmessage, '(a)') 'Processing the Holland data structure for the best track file: ' //
&
01025             trim(adjustl(besttrackfilename(stcnt)))
01026         CALL logmessage(info, scratchmessage)
01027     END IF
01028 END DO
01029 !-----
01030
01031 !-----
01032 ! THIS IS THE MAIN TIME LOOP

```

```

01033      !-----
01034      WRITE(scratchmessage, '(a)') 'Start of the main time loop'
01035      CALL allmessage(info, scratchmessage)
01036      DO icnt = 1, noutdt
01037          WRITE(tmpstr1, '(i5)') icnt
01038          WRITE(tmpstr2, '(i5)') noutdt
01039          tmpstr1 = '(' // trim(tmpstr1) // '/' // trim(adjustl(tmpstr2)) // ')'
01040          WRITE(tmptimestr, '(f20.3)') times(icnt)
01041          WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(tmpstr1)) // " " //
trim(adjustl(tmptimestr))
01042          CALL allmessage(scratchmessage)
01043
01044      DO stcnt = 1, nbtrfiles
01045          ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
01046          ! factor for the subsequent linear interpolation in time. In order for this to
01047          ! work, the array holStru%castTime should be ordered in ascending order.
01048          CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, j11, j12, wtratio)
01049
01050          ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
01051          ! by exiting this loop
01052          IF ((j11 <= 0) .OR. (j12 <= 0)) THEN
01053              WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(tmptimestr)) // &
01054              ' , skipping generating data for this time'
01055              CALL logmessage(info, scratchmessage)
01056
01057              EXIT
01058          END IF
01059
01060          ! Perform linear interpolation in time
01061          stormnumber = holstru(stcnt)%stormNumber(j11)
01062
01063          CALL sphericalfracpoint(holstru(stcnt)%lat(j11), holstru(stcnt)%lon(j11), &
01064                                  holstru(stcnt)%lat(j12), holstru(stcnt)%lon(j12), &
01065                                  wtratio, lat, lon)
01066          !lat = holStru(stCnt)%lat(j11) + &
01067          !      wtratio * (holStru(stCnt)%lat(j12) - holStru(stCnt)%lat(j11))
01068          !lon = holStru(stCnt)%lon(j11) + &
01069          !      wtratio * (holStru(stCnt)%lon(j12) - holStru(stCnt)%lon(j11))
01070
01071          ! Radius of the last closed isobar
01072          rrp = holstru(stcnt)%rrp(j11) + &
01073              wtratio * (holstru(stcnt)%rrp(j12) - holstru(stcnt)%rrp(j11))
01074
01075          ! Radius of maximum winds
01076          rmw = holstru(stcnt)%rmw(j11) + &
01077              wtratio * (holstru(stcnt)%rmw(j12) - holstru(stcnt)%rmw(j11))
01078
01079          ! Get all the distances of the mesh nodes from (lat, lon)
01080          rad = sphericaldistance(sfea, slam, lat, lon)
01081          ! ... and the indices of the nodal points where rad <= rrp
01082          radidx = pack([(i, i = 1, np)], rad <= rrp)
01083          maxradidx = SIZE(radidx)
01084
01085          ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
01086          IF (maxradidx == 0) THEN
01087              WRITE(tmpstr1, '(f20.3)') rrp
01088              tmpstr1 = '(' // trim(adjustl(tmpstr1)) // ' m)'
01089              WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar'
01090              ' // &
01091              trim(adjustl(tmpstr1)) // ' for storm: ' // &
01092              trim(adjustl(holstru(stcnt)%thisStorm))
01093              CALL logmessage(info, scratchmessage)
01094
01095              EXIT
01096          END IF
01097
01098          speed = holstru(stcnt)%speed(j11) + &
01099              wtratio * (holstru(stcnt)%speed(j12) - holstru(stcnt)%speed(j11))
01100
01101          cpress = holstru(stcnt)%cPress(j11) + &
01102              wtratio * (holstru(stcnt)%cPress(j12) - holstru(stcnt)%cPress(j11))
01103
01104          trvx = holstru(stcnt)%trVx(j11) + &
01105              wtratio * (holstru(stcnt)%trVx(j12) - holstru(stcnt)%trVx(j11))
01106          trvy = holstru(stcnt)%trVy(j11) + &
01107              wtratio * (holstru(stcnt)%trVy(j12) - holstru(stcnt)%trVy(j11))
01108
01109          ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
01110          ! background pressure and return. PV: check if this is actually needed
01111          IF (cpress < 0.0_sz) THEN
01112              wpress(:, icnt) = backgroundatmpress * mb2pa

```

```

01112         wvelx(:, icnt) = 0.0_sz
01113         wvely(:, icnt) = 0.0_sz
01114
01115         WRITE(scratchmessage, '(a)') 'Calm period found, generating zero atmospheric fields for this
time'
01116         CALL logmessage(info, scratchmessage)
01117
01118         EXIT
01119     END IF
01120
01121     ! Calculate and limit central pressure deficit; some track files (e.g., Charley 2004)
01122     ! may have a central pressure greater than the ambient pressure that this subroutine assumes
01123     cpressdef = backgroundatmpress * mb2pa - cpress
01124     IF (cpressdef < 100.0_sz) cpressdef = 100.0_sz
01125
01126     ! Subtract the translational speed of the storm from the observed max wind speed to avoid
01127     ! distortion in the Holland curve fit. The translational speed will be added back later.
01128     trspd = sqrt(trvx * trvx + trvy * trvy)
01129     speed = speed - trspd
01130
01131     ! Convert wind speed from 10 meter altitude (which is what the
01132     ! NHC forecast contains) to wind speed at the top of the atmospheric
01133     ! boundary layer (which is what the Holland curve fit requires).
01134     speed = speed / bladjustfac
01135
01136     ! Calculate Holland parameters and limit the result to its appropriate range.
01137     hlb = rhoair * basee * (speed**2) / cpressdef
01138     IF (hlb < 1.0_sz) hlb = 1.0_sz
01139     IF (hlb > 2.5_sz) hlb = 2.5_sz
01140
01141     ! If we are running storm 2 in the Lake Pontchartrain !PV Do we need this?
01142     ! Forecast System ensemble, the final wind speeds should be multiplied by 1.2.
01143     windmultiplier = 1.0_sz
01144     IF (stormnumber == 2) windmultiplier = 1.2_sz
01145
01146     DO npcnt = 1, maxradidx
01147         i = radidx(npncnt)
01148
01149         dx = sphericaldistance(lat, lon, lat, slam(i))
01150         dy = sphericaldistance(lat, lon, sfea(i), lon)
01151         theta = atan2(dy, dx)
01152
01153         ! Compute coriolis
01154         coriolis = 2.0_sz * omega * sin(sfea(i) * deg2rad)
01155
01156         ! Compute the pressure (Pa) at a distance rad(i); all distances are in meters
01157         sfpress = cpress + cpressdef * exp(-(rmw / rad(i))**hlb)
01158
01159         ! Compute wind speed (speed - trSPD) at gradient level (m/s) and at a distance rad(i);
01160         ! all distances are in meters. Using absolute value for coriolis for Southern Hemisphere
01161         grvel = sqrt(speed**2 * (rmw / rad(i))**hlb * exp(1.0_sz - (rmw / rad(i))**hlb) +
01162             (rad(i) * abs(coriolis) / 2.0_sz)**2) -
01163             rad(i) * abs(coriolis) / 2.0_sz
01164
01165         ! Determine translation speed that should be added to final !PV CHECK ON THIS
01166         ! storm wind speed. This is tapered to zero as the storm wind tapers
01167         ! to zero toward the eye of the storm and at long distances from the storm.
01168         trspdx = (abs(grvel) / speed) * trvx
01169         trspdy = (abs(grvel) / speed) * trvy
01170
01171         ! Apply mutliplier for Storm #2 in LPFS ensemble.
01172         grvel = grvel * windmultiplier
01173
01174         ! Find the wind velocity components.
01175         sfvelx = -grvel * sin(theta)
01176         sfvely = grvel * cos(theta)
01177         !print *, sfVelX, sfVelY
01178         ! Convert wind velocity from the gradient level (top of atmospheric boundary layer)
01179         ! which, is what the Holland curve fit produces, to 10-m wind velocity.
01180         sfvelx = sfvelx * bladjustfac
01181         sfvely = sfvely * bladjustfac
01182         !print *, sfVelX, sfVelY
01183         ! Convert from 1-minute averaged winds to 10-minute averaged winds.
01184         sfvelx = sfvelx * one2ten
01185         sfvely = sfvely * one2ten
01186         !print *, sfVelX, sfVelY
01187         ! Add back the storm translation speed.
01188         sfvelx = sfvelx + trspdx
01189         sfvely = sfvely + trspdy
01190
01191         !print *, sfVelX, sfVelY, wVelX(i, iCnt), wVelY(i, iCnt)

```

```

01192      !PV Need to interpolate between storms if this nodal point
01193      !   is affected by more than on storm
01194      wpress(i, icnt) = sfpress
01195      wvelx(i, icnt) = sfvelx
01196      wvely(i, icnt) = sfvely
01197
01198      !print *, sfVelX, sfVelY, wVelX(i, iCnt), wVelY(i, iCnt)
01199      !print *, '-----'
01200      END DO ! npCnt = 1, maxRadIDX
01201
01202      END DO ! stCnt = 1, nBTrFiles
01203      END DO ! iCnt = 1, nOutDT
01204      WRITE(scratchmessage, '(a)') 'End of the main time loop'
01205      CALL allmessage(info, scratchmessage)
01206
01207      !----- Deallocate the arrays
01208      IF (ALLOCATED(rad)) DEALLOCATE(rad)
01209      IF (ALLOCATED(radidx)) DEALLOCATE(radidx)
01210      DO icnt = 1, nbtrfiles
01211          CALL deallochoollstruct(holstru(icnt))
01212      END DO
01213      DEALLOCATE(holstru)
01214      !-----
01215
01216      CALL unsetmessagesource()
01217
01218      END SUBROUTINE gethollandfields
01219
01220      !=====
01221
01222      !-----
01223      ! SUBROUTINE WRITE BEST TRACK DATA
01224      !-----
01225      ! author Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>
01226      !-----
01241      SUBROUTINE writebesttrackdata(inpFile, btrStruc, suffix)
01242
01243      USE pahm_global, ONLY : lun_btrk, lun_btrk1
01244
01245      IMPLICIT NONE
01246
01247      ! Global variables
01248      CHARACTER(LEN=*) :: inpFile
01249      TYPE(besttrackdata_t), INTENT(IN) :: btrStruc
01250      CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: suffix
01251
01252      ! Local variables
01253      CHARACTER(LEN=FNAMELEN) :: outFile
01254      CHARACTER(LEN=64) :: fsuf
01255      INTEGER :: iCnt
01256      INTEGER :: iUnit, errIO
01257      CHARACTER(LEN=512) :: fmtStr
01258
01259
01260      !----- Initialize variables
01261      iunit = lun_btrk1
01262      errio = 0
01263
01264      fmtstr = '(a2, ",", 1x, i2.2, ",", 1x, a10, ",", 1x, i2, ",", 1x, a4, ",", 1x, i3, ",", 1x, i3, a1,
01265      ", ", 1x, i4, a1, ",", '
01266      fmtstr = trim(fmtstr) // ' 1x, i3, ",", 1x, i4, ",", 1x, a2, ",", 1x, i3, ",", 1x, a3, ",", '
01267      fmtstr = trim(fmtstr) // ' 4(1x, i4, ",", 1x, i4, ",", 1x, i4, ",", 1x, i3, ",", 1x, i4, ",", 1x,
01268      i3, ",", '
01269      fmtstr = trim(fmtstr) // ' 1x, a3, ",", 1x, i3, ",", 1x, a3, ",", 1x, i3, ",", 1x, a11, ",", 1x,
01270      i3, ",", '
01271      !-----
01272
01273      fsuf = '_adj'
01274      IF (PRESENT(suffix)) fsuf = adjustl(suffix)
01275
01276      CALL setmessagesource("WriteBestTrackData")
01277
01278      IF (.NOT. btrstruc%loaded) THEN
01279          WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be
01280          written."
01281          CALL allmessage(info, scratchmessage)
01282
01283          RETURN
01284      END IF
01285
01286      outfile = trim(adjustl(inpfile)) // trim(fsuf)

```



```

01283
01284 WRITE(scratchmessage, '(a)') 'Writing the "adjusted" best track data to: ' // trim(adjustl(outfile))
01285 CALL logmessage(info, scratchmessage)
01286
01287 OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)
01288
01289 IF (errio /= 0) THEN
01290   WRITE(scratchmessage, '(a)') 'Error opening the outFile: ' // trim(outfile) // &
01291   ' ', skip writting the "adjusted" best track fields'
01292   CALL allmessage(error, scratchmessage)
01293
01294   RETURN
01295 END IF
01296
01297 DO icnt = 1, btrstruc%numRec
01298   WRITE(iunit, fmtstr)
01299   btrstruc%basin(icnt),      btrstruc%cyNum(icnt),      &
01300   btrstruc%dtg(icnt),      btrstruc%techNum(icnt),      &
01301   btrstruc%tech(icnt),      btrstruc%tau(icnt),      &
01302   btrstruc%intLat(icnt),    btrstruc%ns(icnt),      &
01303   btrstruc%intLon(icnt),    btrstruc%ew(icnt),      &
01304   btrstruc%intVMax(icnt),    btrstruc%intMslp(icnt),    &
01305   btrstruc%ty(icnt),        btrstruc%rad(icnt),      &
01306   btrstruc%windCode(icnt),  btrstruc%intRad1(icnt),    &
01307   btrstruc%intRad2(icnt),    btrstruc%intRad3(icnt),    &
01308   btrstruc%intRad4(icnt),    btrstruc%intPOuter(icnt),  &
01309   btrstruc%intROuter(icnt),  btrstruc%intRmw(icnt),    &
01310   btrstruc%gusts(icnt),      btrstruc%eye(icnt),      &
01311   btrstruc%subregion(icnt),  btrstruc%maxseas(icnt),    &
01312   btrstruc%initials(icnt),   btrstruc%dir(icnt),      &
01313   btrstruc%intSpeed(icnt),   btrstruc%stormName(icnt), &
01314   btrstruc%cycleNum(icnt)
01315 END DO
01316
01317 CLOSE(iunit)
01318
01319 CALL unsetmessagesource()
01320
01321 END SUBROUTINE writebesttrackdata
01322
01323 !=====
01324
01325 !-----
01326 ! SUBROUTINE ALLOC BTR STRUCT
01327 !-----
01328 !-----
01329 !-----
01330 SUBROUTINE allocbtrstruct(str, nRec)
01331
01332   IMPLICIT NONE
01333
01334   TYPE(besttrackdata_t) :: str
01335   INTEGER, INTENT(IN)   :: nRec
01336
01337   str%numRec = nrec
01338   str%loaded = .false.
01339
01340   !----- Input parameters
01341   IF (.NOT. ALLOCATED(str%basin)) ALLOCATE(str%basin(nrec))
01342   IF (.NOT. ALLOCATED(str%cyNum)) ALLOCATE(str%cyNum(nrec))
01343   IF (.NOT. ALLOCATED(str%dtg)) ALLOCATE(str%dtg(nrec))
01344   IF (.NOT. ALLOCATED(str%techNum)) ALLOCATE(str%techNum(nrec))
01345   IF (.NOT. ALLOCATED(str%tech)) ALLOCATE(str%tech(nrec))
01346   IF (.NOT. ALLOCATED(str%tau)) ALLOCATE(str%tau(nrec))
01347   IF (.NOT. ALLOCATED(str%intLat)) ALLOCATE(str%intLat(nrec))
01348   IF (.NOT. ALLOCATED(str%intLon)) ALLOCATE(str%intLon(nrec))
01349   IF (.NOT. ALLOCATED(str%ew)) ALLOCATE(str%ew(nrec))
01350   IF (.NOT. ALLOCATED(str%ns)) ALLOCATE(str%ns(nrec))
01351   IF (.NOT. ALLOCATED(str%intVMax)) ALLOCATE(str%intVMax(nrec))
01352   IF (.NOT. ALLOCATED(str%intMslp)) ALLOCATE(str%intMslp(nrec))
01353   IF (.NOT. ALLOCATED(str%ty)) ALLOCATE(str%ty(nrec))
01354   IF (.NOT. ALLOCATED(str%rad)) ALLOCATE(str%rad(nrec))
01355   IF (.NOT. ALLOCATED(str%windCode)) ALLOCATE(str%windCode(nrec))
01356   IF (.NOT. ALLOCATED(str%intRad1)) ALLOCATE(str%intRad1(nrec))
01357   IF (.NOT. ALLOCATED(str%intRad2)) ALLOCATE(str%intRad2(nrec))
01358   IF (.NOT. ALLOCATED(str%intRad3)) ALLOCATE(str%intRad3(nrec))
01359   IF (.NOT. ALLOCATED(str%intRad4)) ALLOCATE(str%intRad4(nrec))
01360   IF (.NOT. ALLOCATED(str%intPOuter)) ALLOCATE(str%intPOuter(nrec))
01361   IF (.NOT. ALLOCATED(str%intROuter)) ALLOCATE(str%intROuter(nrec))
01362   IF (.NOT. ALLOCATED(str%intRmw)) ALLOCATE(str%intRmw(nrec))
01363   IF (.NOT. ALLOCATED(str%gusts)) ALLOCATE(str%gusts(nrec))
01364   IF (.NOT. ALLOCATED(str%eye)) ALLOCATE(str%eye(nrec))

```

```

01365     IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
01366     IF (.NOT. ALLOCATED(str%maxseas)) ALLOCATE(str%maxseas(nrec))
01367     IF (.NOT. ALLOCATED(str%initials)) ALLOCATE(str%initials(nrec))
01368     IF (.NOT. ALLOCATED(str%dir)) ALLOCATE(str%dir(nrec))
01369     IF (.NOT. ALLOCATED(str%intSpeed)) ALLOCATE(str%intSpeed(nrec))
01370     IF (.NOT. ALLOCATED(str%stormName)) ALLOCATE(str%stormName(nrec))
01371     IF (.NOT. ALLOCATED(str%cycleNum)) ALLOCATE(str%cycleNum(nrec))
01372
01373     !----- Converted parameters
01374     IF (.NOT. ALLOCATED(str%year)) ALLOCATE(str%year(nrec))
01375     IF (.NOT. ALLOCATED(str%month)) ALLOCATE(str%month(nrec))
01376     IF (.NOT. ALLOCATED(str%day)) ALLOCATE(str%day(nrec))
01377     IF (.NOT. ALLOCATED(str%hour)) ALLOCATE(str%hour(nrec))
01378     IF (.NOT. ALLOCATED(str%lat)) ALLOCATE(str%lat(nrec))
01379     IF (.NOT. ALLOCATED(str%lon)) ALLOCATE(str%lon(nrec))
01380
01381     END SUBROUTINE allocbtrstruct
01382
01383 !=====
01384
01385 !-----
01386 ! SUBROUTINE DEALLOC BTR STRUCT
01387 !-----
01388 !-----
01389 !-----
01390 SUBROUTINE deallocbtrstruct(str)
01391
01392     IMPLICIT NONE
01393
01394     TYPE(besttrackdata_t) :: str
01395
01396     str%numRec = -1
01397     str%loaded = .false.
01398
01399     !----- Input parameters
01400     IF (ALLOCATED(str%basin)) DEALLOCATE(str%basin)
01401     IF (ALLOCATED(str%cyNum)) DEALLOCATE(str%cyNum)
01402     IF (ALLOCATED(str%dtg)) DEALLOCATE(str%dtg)
01403     IF (ALLOCATED(str%techNum)) DEALLOCATE(str%techNum)
01404     IF (ALLOCATED(str%tech)) DEALLOCATE(str%tech)
01405     IF (ALLOCATED(str%tau)) DEALLOCATE(str%tau)
01406     IF (ALLOCATED(str%intLat)) DEALLOCATE(str%intLat)
01407     IF (ALLOCATED(str%intLon)) DEALLOCATE(str%intLon)
01408     IF (ALLOCATED(str%ew)) DEALLOCATE(str%ew)
01409     IF (ALLOCATED(str%ns)) DEALLOCATE(str%ns)
01410     IF (ALLOCATED(str%intVMax)) DEALLOCATE(str%intVMax)
01411     IF (ALLOCATED(str%intMslp)) DEALLOCATE(str%intMslp)
01412     IF (ALLOCATED(str%ty)) DEALLOCATE(str%ty)
01413     IF (ALLOCATED(str%rad)) DEALLOCATE(str%rad)
01414     IF (ALLOCATED(str%windCode)) DEALLOCATE(str%windCode)
01415     IF (ALLOCATED(str%intRad1)) DEALLOCATE(str%intRad1)
01416     IF (ALLOCATED(str%intRad2)) DEALLOCATE(str%intRad2)
01417     IF (ALLOCATED(str%intRad3)) DEALLOCATE(str%intRad3)
01418     IF (ALLOCATED(str%intRad4)) DEALLOCATE(str%intRad4)
01419     IF (ALLOCATED(str%intPOuter)) DEALLOCATE(str%intPOuter)
01420     IF (ALLOCATED(str%intROuter)) DEALLOCATE(str%intROuter)
01421     IF (ALLOCATED(str%intRmw)) DEALLOCATE(str%intRmw)
01422     IF (ALLOCATED(str%gusts)) DEALLOCATE(str%gusts)
01423     IF (ALLOCATED(str%eye)) DEALLOCATE(str%eye)
01424     IF (ALLOCATED(str%subregion)) DEALLOCATE(str%subregion)
01425     IF (ALLOCATED(str%maxseas)) DEALLOCATE(str%maxseas)
01426     IF (ALLOCATED(str%initials)) DEALLOCATE(str%initials)
01427     IF (ALLOCATED(str%dir)) DEALLOCATE(str%dir)
01428     IF (ALLOCATED(str%intSpeed)) DEALLOCATE(str%intSpeed)
01429     IF (ALLOCATED(str%stormName)) DEALLOCATE(str%stormName)
01430     IF (ALLOCATED(str%cycleNum)) DEALLOCATE(str%cycleNum)
01431
01432     !----- Converted parameters
01433     IF (ALLOCATED(str%year)) DEALLOCATE(str%year)
01434     IF (ALLOCATED(str%month)) DEALLOCATE(str%month)
01435     IF (ALLOCATED(str%day)) DEALLOCATE(str%day)
01436     IF (ALLOCATED(str%hour)) DEALLOCATE(str%hour)
01437     IF (ALLOCATED(str%lat)) DEALLOCATE(str%lat)
01438     IF (ALLOCATED(str%lon)) DEALLOCATE(str%lon)
01439
01440     END SUBROUTINE deallocbtrstruct
01441
01442 !=====
01443
01444 !-----
01445 ! SUBROUTINE ALLOC HOLL STRUCT
01446 !-----

```

```

01448 !-----
01449 SUBROUTINE allochollstruct(str, nRec)
01450
01451   IMPLICIT NONE
01452
01453   TYPE(hollanddata_t) :: str
01454   INTEGER, INTENT(IN) :: nRec
01455
01456   str%numRec = nrec
01457   str%loaded = .false.
01458
01459   !----- Input parameters
01460   IF (.NOT. ALLOCATED(str%basin))      ALLOCATE(str%basin(nrec))
01461
01462   IF (.NOT. ALLOCATED(str%dtg))        ALLOCATE(str%dtg(nrec))
01463   IF (.NOT. ALLOCATED(str%stormNumber)) ALLOCATE(str%stormNumber(nrec))
01464   IF (.NOT. ALLOCATED(str%year))       ALLOCATE(str%year(nrec))
01465   IF (.NOT. ALLOCATED(str%month))      ALLOCATE(str%month(nrec))
01466   IF (.NOT. ALLOCATED(str%day))        ALLOCATE(str%day(nrec))
01467   IF (.NOT. ALLOCATED(str%hour))       ALLOCATE(str%hour(nrec))
01468
01469   IF (.NOT. ALLOCATED(str%castTime))   ALLOCATE(str%castTime(nrec))
01470   IF (.NOT. ALLOCATED(str%castType))   ALLOCATE(str%castType(nrec))
01471   IF (.NOT. ALLOCATED(str%fcstInc))    ALLOCATE(str%fcstInc(nrec))
01472
01473   IF (.NOT. ALLOCATED(str%iLat))       ALLOCATE(str%iLat(nrec))
01474   IF (.NOT. ALLOCATED(str%lat))        ALLOCATE(str%lat(nrec))
01475   IF (.NOT. ALLOCATED(str%iLon))       ALLOCATE(str%iLon(nrec))
01476   IF (.NOT. ALLOCATED(str%lon))        ALLOCATE(str%lon(nrec))
01477
01478   IF (.NOT. ALLOCATED(str%iSpeed))     ALLOCATE(str%iSpeed(nrec))
01479   IF (.NOT. ALLOCATED(str%speed))      ALLOCATE(str%speed(nrec))
01480
01481   IF (.NOT. ALLOCATED(str%iCPress))    ALLOCATE(str%iCPress(nrec))
01482   IF (.NOT. ALLOCATED(str%cPress))     ALLOCATE(str%cPress(nrec))
01483
01484   IF (.NOT. ALLOCATED(str%iRrp))       ALLOCATE(str%iRrp(nrec))
01485   IF (.NOT. ALLOCATED(str%rrp))        ALLOCATE(str%rrp(nrec))
01486
01487   IF (.NOT. ALLOCATED(str%iRmw))       ALLOCATE(str%iRmw(nrec))
01488   IF (.NOT. ALLOCATED(str%rmw))        ALLOCATE(str%rmw(nrec))
01489
01490   IF (.NOT. ALLOCATED(str%cPrDt))      ALLOCATE(str%cPrDt(nrec))
01491
01492   IF (.NOT. ALLOCATED(str%trVx))       ALLOCATE(str%trVx(nrec))
01493   IF (.NOT. ALLOCATED(str%trVy))       ALLOCATE(str%trVy(nrec))
01494
01495 END SUBROUTINE allochollstruct
01496
01497 !=====
01498
01499 !-----
01500 ! SUBROUTINE DEALLOC HOLL STRUCT
01501 !-----
01502 !-----
01503 !-----
01504 SUBROUTINE deallocollstruct(str)
01505
01506   IMPLICIT NONE
01507
01508   TYPE(hollanddata_t), INTENT(OUT) :: str
01509
01510   str%numRec = -1
01511   str%loaded = .false.
01512
01513   !----- Input parameters
01514   IF (ALLOCATED(str%basin))      DEALLOCATE(str%basin)
01515
01516   IF (ALLOCATED(str%dtg))        DEALLOCATE(str%dtg)
01517   IF (ALLOCATED(str%stormNumber)) DEALLOCATE(str%stormNumber)
01518   IF (ALLOCATED(str%year))       DEALLOCATE(str%year)
01519   IF (ALLOCATED(str%month))      DEALLOCATE(str%month)
01520   IF (ALLOCATED(str%day))        DEALLOCATE(str%day)
01521   IF (ALLOCATED(str%hour))       DEALLOCATE(str%hour)
01522
01523   IF (ALLOCATED(str%castTime))   DEALLOCATE(str%castTime)
01524   IF (ALLOCATED(str%castType))   DEALLOCATE(str%castType)
01525   IF (ALLOCATED(str%fcstInc))    DEALLOCATE(str%fcstInc)
01526
01527   IF (ALLOCATED(str%iLat))       DEALLOCATE(str%iLat)
01528   IF (ALLOCATED(str%lat))        DEALLOCATE(str%lat)
01529   IF (ALLOCATED(str%iLon))       DEALLOCATE(str%iLon)

```

```

01530      IF (ALLOCATED(str%lon))          DEALLOCATE(str%lon)
01531
01532      IF (ALLOCATED(str%iSpeed))        DEALLOCATE(str%iSpeed)
01533      IF (ALLOCATED(str%speed))         DEALLOCATE(str%speed)
01534
01535      IF (ALLOCATED(str%iCPress))        DEALLOCATE(str%iCPress)
01536      IF (ALLOCATED(str%cPress))         DEALLOCATE(str%cPress)
01537
01538      IF (ALLOCATED(str%iRrp))           DEALLOCATE(str%iRrp)
01539      IF (ALLOCATED(str%rrp))            DEALLOCATE(str%rrp)
01540
01541      IF (ALLOCATED(str%iRmw))           DEALLOCATE(str%iRmw)
01542      IF (ALLOCATED(str%rmw))            DEALLOCATE(str%rmw)
01543
01544      IF (ALLOCATED(str%cPrDt))          DEALLOCATE(str%cPrDt)
01545
01546      IF (ALLOCATED(str%trVx))           DEALLOCATE(str%trVx)
01547      IF (ALLOCATED(str%trVy))           DEALLOCATE(str%trVy)
01548
01549      END SUBROUTINE deallocollstruct
01550
01551      !=====
01552
01553      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::deallocchollstruct](#) (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.](#)

```

00001 !-----
00002 !               M O D U L E   P A R W I N D
00003 !-----
00014 !-----
00015
00016 MODULE parwind
00017
00018     USE pahm_sizes
00019     USE pahm_messages
00020
00021     ! switch to turn on or off geostrophic balance in GAHM
00022     ! on (default): Coriolis term included, phiFactors will be calculated before being used
00023     ! off          : parameter is set to 'TRUE', phiFactors will be set to constant 1
00024     !LOGICAL :: geostrophicSwitch = .TRUE.
00025     !INTEGER :: geoFactor = 1           !turn on or off geostrophic balance
00026
00027     REAL(SZ) :: WindRefTime !jgf46.29 seconds since beginning of year, this           !PV check
00028                  !corresponds to time=0 of the simulation
00029
00030     !-----
00031     ! The BestTrackData_T structure holds all data read from the best track files(s)
00032     ! in ATCF format (a-deck/b-deck)
00033     !-----
00034     TYPE besttrackdata_t
00035         CHARACTER(LEN=FNAMLEN) :: fileName           ! full path to the best track file
00036         CHARACTER(LEN=10)      :: thisStorm          ! the name of the "named" storm
00037         LOGICAL                :: loaded = .false.   ! .TRUE. if we have loaded the data from file
00038         INTEGER                :: numRec             ! number of records in the structure
00039
00040         !----- input data from best track file (ATCF format)
00041         CHARACTER(LEN=2), ALLOCATABLE :: basin(:)      ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00042         INTEGER, ALLOCATABLE          :: cyNum(:)      ! annual cyclone number: 1 - 99
00043         CHARACTER(LEN=10), ALLOCATABLE :: dtg(:)       ! warning Date-Time-Group (DTG), YYYYMMDDHH
00044         INTEGER, ALLOCATABLE          :: techNum(:)    ! objective technique sorting number, minutes for
00045         CHARACTER(LEN=4), ALLOCATABLE :: tech(:)       ! acronym for each objective technique or CARQ or
00046         WRNG,
00047         INTEGER, ALLOCATABLE          :: tau(:)        ! BEST for best track, up to 4 chars.
00048         ! best-track,
00049         ! negative taus used for CARQ and WRNG records.
00049         INTEGER, ALLOCATABLE          :: intLat(:)     ! latitude for the DTG: 0 - 900 tenths of degrees
00050         INTEGER, ALLOCATABLE          :: intLon(:)     ! latitude for the DTG: 0 - 900 tenths of degrees
00051         CHARACTER(LEN=1), ALLOCATABLE :: ew(:)        ! E/W
00052         CHARACTER(LEN=1), ALLOCATABLE :: ns(:)        ! N/S
00053
00054         INTEGER, ALLOCATABLE          :: intVMax(:)    ! maximum sustained wind speed in knots: 0 - 300
00055         kts
00055         INTEGER, ALLOCATABLE          :: intMslp(:)    ! minimum sea level pressure, 850 - 1050 mb
00056         CHARACTER(LEN=2), ALLOCATABLE :: ty(:)        ! Highest level of tc development:
00057         ! DB - disturbance,
00058         ! TD - tropical depression,
00059         ! TS - tropical storm,
00060         ! TY - typhoon,
00061         ! ST - super typhoon,
00062         ! TC - tropical cyclone,
00063         ! HU - hurricane,
00064         ! SD - subtropical depression,
00065         ! SS - subtropical storm,
00066         ! EX - extratropical systems,
00067         ! PT - post tropical,

```

```

00068                                     ! IN - inland,
00069                                     ! DS - dissipating,
00070                                     ! LO - low,
00071                                     ! WV - tropical wave,
00072                                     ! ET - extrapolated,
00073                                     ! MD - monsoon depression,
00074                                     ! XX - unknown.
00075     INTEGER, ALLOCATABLE               :: rad(:)      ! wind intensity for the radii defined in this
record: 34, 50 or 64 kt
00076     CHARACTER(LEN=3), ALLOCATABLE     :: windCode(:) ! radius code:
00077                                     ! AAA - full circle
00078                                     ! NEQ, SEQ, SWQ, NWQ - quadrant
00079     INTEGER, ALLOCATABLE               :: intRad1(:)  ! if full circle, radius of specified wind
intensity, or radius of
00080                                     ! first quadrant wind intensity as specified by
WINDCODE. 0 - 999 n mi
00081     INTEGER, ALLOCATABLE               :: intRad2(:)  ! if full circle this field not used, or radius
of 2nd quadrant wind
00082                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00083     INTEGER, ALLOCATABLE               :: intRad3(:)  ! if full circle this field not used, or radius
of 3rd quadrant wind
00084                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00085     INTEGER, ALLOCATABLE               :: intRad4(:)  ! if full circle this field not used, or radius
of 4th quadrant wind
00086                                     ! intensity as specified by WINDCODE. 0 - 999 n
mi
00087     INTEGER, ALLOCATABLE               :: intPOuter(:) ! pressure in millibars of the last closed
isobar, 900 - 1050 mb
00088     INTEGER, ALLOCATABLE               :: intROuter(:) ! radius of the last closed isobar, 0 - 999 n mi
00089     INTEGER, ALLOCATABLE               :: intRmw(:)    ! radius of max winds, 0 - 999 n mi
00090     INTEGER, ALLOCATABLE               :: gusts(:)     ! gusts, 0 - 999 kt
00091     INTEGER, ALLOCATABLE               :: eye(:)       ! eye diameter, 0 - 120 n mi
00092     CHARACTER(LEN=3), ALLOCATABLE     :: subregion(:) ! subregion code: W,A,B,S,P,C,E,L,Q
00093                                     ! A - Arabian Sea
00094                                     ! B - Bay of Bengal
00095                                     ! C - Central Pacific
00096                                     ! E - Eastern Pacific
00097                                     ! L - Atlantic
00098                                     ! P - South Pacific (135E - 120W)
00099                                     ! Q - South Atlantic
00100                                     ! S - South IO (20E - 135E)
00101                                     ! W - Western Pacific
00102     INTEGER, ALLOCATABLE               :: maxseas(:)   ! max seas: 0 - 999 ft
00103     CHARACTER(LEN=3), ALLOCATABLE     :: initials(:)  ! forecaster's initials used for tau 0 WRNG or
OFCL, up to 3 chars
00104     INTEGER, ALLOCATABLE               :: dir(:)       ! storm direction, 0 - 359 degrees
00105     INTEGER, ALLOCATABLE               :: intSpeed(:)  ! storm speed, 0 - 999 kts
00106     CHARACTER(LEN=10), ALLOCATABLE    :: stormName(:) ! literal storm name, number, NONAME or INVEST,
or TCcyx where:
00107                                     ! cy = Annual cyclone number 01 - 99
00108                                     ! x = Subregion code: W,A,B,S,P,C,E,L,Q.
00109     INTEGER, ALLOCATABLE               :: cycleNum(:)  ! the cycle number !PV check if this is OK
00110
00111 ! !----- converted data from the above values (if needed)
00112     INTEGER, DIMENSION(:), ALLOCATABLE :: year, month, day, hour
00113     REAL(SZ), DIMENSION(:), ALLOCATABLE :: lat, lon
00114     END TYPE besttrackdata_t
00115
00116 ! Array of info about the best track data (extension to use multiple storms)
00117     TYPE(BestTrackData_T), ALLOCATABLE, TARGET :: bestTrackData(:)
00118
00119 !-----
00120 ! The HollandData_T structure holds all required data for the Holland model
00121 ! The data are filtered to only include unique DTGs
00122 !-----
00123     TYPE hollanddata_t
00124     CHARACTER(LEN=FNAMLEN)             :: fileName    ! full path to the best track file
00125     CHARACTER(LEN=10)                  :: thisStorm   ! the name of the "named" storm
00126     LOGICAL                            :: loaded = .false. ! .TRUE. if we have loaded the data from file
00127     INTEGER                             :: numRec     ! number of records in the structure
00128
00129     CHARACTER(LEN=2), ALLOCATABLE       :: basin(:)   ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00130     INTEGER, ALLOCATABLE                 :: stormNumber(:) ! annual cyclone number: 1 - 99
00131     CHARACTER(LEN=10), ALLOCATABLE      :: dtg(:)     ! warning Date-Time-Group (DTG), YYYYMMDDHH
00132     INTEGER, DIMENSION(:), ALLOCATABLE  :: year, month, day, hour
00133     REAL(SZ), ALLOCATABLE                :: castTime(:) ! converted to decimal E/N (lon, lat)
00134     CHARACTER(LEN=4), ALLOCATABLE       :: castType(:) ! BEST, OFCL, CALM, ...
00135     INTEGER, ALLOCATABLE                 :: fcstInc(:) ! forecast period: -24 through 240 hours, 0
for best-track

```

```

00136
00137     INTEGER, DIMENSION(:), ALLOCATABLE :: iLat, iLon      ! latitude, longitude for the GTD
00138     REAL(SZ), DIMENSION(:), ALLOCATABLE :: lat, lon       ! converted to decimal E/N (lon, lat)
00139
00140     INTEGER, ALLOCATABLE :: iSpeed(:)                   ! maximum sustained wind speed in knots: 0 -
00141 300 kts
00142     REAL(SZ), ALLOCATABLE :: speed(:)                   ! converted from kts to m/s
00143
00143     INTEGER, ALLOCATABLE :: iCPress(:)                  ! minimum sea level pressure, 850 - 1050 mb
00144     REAL(SZ), ALLOCATABLE :: cPress(:)                  ! converted to Pa
00145
00146     INTEGER, ALLOCATABLE :: iRrp(:)                     ! radius of the last closed isobar, 0 - 999 n
00147  mi
00147     REAL(SZ), ALLOCATABLE :: rrp(:)                     ! converted from nm to m
00148
00149     INTEGER, ALLOCATABLE :: iRmw(:)                     ! radius of max winds, 0 - 999 n mi
00150     REAL(SZ), ALLOCATABLE :: rmw(:)                     ! converted from nm to m
00151
00152     REAL(SZ), DIMENSION(:), ALLOCATABLE :: cPrDt        ! central pressure intensity change (Pa / h)
00153     REAL(SZ), DIMENSION(:), ALLOCATABLE :: trVx, trVy    ! translational velocity components (x, y) of
the
00154                                     ! moving hurricane (m/s)
00155     END TYPE hollanddata_t
00156
00157     CONTAINS
00158
00159     !-----
00160     ! SUBROUTINE READ BEST TRACK FILE
00161     !-----
00162     !-----
00163     !-----
00177     SUBROUTINE readbesttrackfile()
00178
00179     USE pahm_global, ONLY : lun_btrk, lun_btrk1, nbtrfiles, besttrackfilename
00180     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
00181     USE sortutils, ONLY : arth, indexx, arrayequal
00182
00183     IMPLICIT NONE
00184
00185     CHARACTER(LEN=FNAMELEN) :: inpFile
00186     CHARACTER(LEN=512)      :: inpLine, line
00187     CHARACTER(LEN=512)      :: fmtStr
00188
00189     INTEGER :: lenLine
00190     INTEGER :: nLines      ! Number of lines counter
00191     INTEGER :: iFile, iCnt  ! loop counters
00192     INTEGER :: iUnit, errIO, ios, status
00193
00194     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00195     INTEGER, ALLOCATABLE :: idxArrStr(:)
00196     INTEGER :: nUnique, maxCnt, kCnt, kMax
00197
00198     INTEGER, ALLOCATABLE :: idx0(:), idx1(:)
00199
00200     !----- Initialize variables
00201     iunit = lun_btrk
00202     errio = 0
00203
00204     fmtstr = ' (a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x,
00205 a2, '
00206     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i4, 2x, i3, 2x, i4, 2x, i3, '
00207     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, i3, 2x, i3, 2x, a11, 2x, i3)'
00208     !-----
00209     CALL setmessagesource("ReadBestTrackFile")
00210
00211     ! Allocate the best track structure array. This structure holds all the
00212     ! input values for the storm track as read in from the track input file
00213     ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00214     ! (as appropriate).
00215     ALLOCATE(besttrackdata(nbtrfiles))
00216
00217     ! This is the main loop. We loop through all the best track files
00218     ! (user input)
00219     DO ifile = 1, nbtrfiles
00220         inpfile = besttrackfilename(ifile)
00221
00222         CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)
00223
00224         IF (errio /= 0) THEN
00225

```

```

00226     WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
00227     CALL allmessage(error, scratchmessage)
00228
00229     CALL unsetmessagesource()
00230
00231     CALL terminate()
00232 ELSE
00233     WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
00234     CALL logmessage(info, scratchmessage)
00235 END IF
00236
00237 besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00238 besttrackdata(ifile)%thisStorm = ""
00239 besttrackdata(ifile)%loaded = .false.
00240 besttrackdata(ifile)%numRec = -1
00241
00242 ! Count the number of non-empty or commented out lines in the file.
00243 ! Comments are considered those lines with the first non-blank character of "!" or "#"
00244 nlines = 0
00245 DO
00246     READ(unit=iunit, fmt='(a)', err=10, END=5, IOSTAT=errIO) inpline
00247
00248     lenline = getlinerecord(inpline, line)
00249     IF (lenline /= 0) nlines = nlines + 1
00250 END DO
00251 5 rewind(unit=iunit)
00252
00253 ! Array allocation in the structure bestTrackData
00254 CALL allocbtrstruct(besttrackdata(ifile), nlines)
00255
00256 icnt = 0
00257 DO WHILE (.true.)
00258     READ(unit=iunit, fmt='(a)', err=10, END=20, IOSTAT=errIO) inpline
00259
00260     lenline = getlinerecord(inpline, line)
00261
00262     IF (lenline /= 0) THEN
00263         icnt = icnt + 1
00264         READ(line, fmt=fmtstr, err=11, iostat=ios)
00265         besttrackdata(ifile)%basin(icnt),      besttrackdata(ifile)%cyNum(icnt),      &
00266         besttrackdata(ifile)%dtg(icnt),        besttrackdata(ifile)%techNum(icnt),    &
00267         besttrackdata(ifile)%tech(icnt),       besttrackdata(ifile)%tau(icnt),      &
00268         besttrackdata(ifile)%intLat(icnt),     besttrackdata(ifile)%ns(icnt),      &
00269         besttrackdata(ifile)%intLon(icnt),     besttrackdata(ifile)%ew(icnt),      &
00270         besttrackdata(ifile)%intVMax(icnt),    besttrackdata(ifile)%intMslp(icnt), &
00271         besttrackdata(ifile)%ty(icnt),         besttrackdata(ifile)%rad(icnt),     &
00272         besttrackdata(ifile)%windCode(icnt),   besttrackdata(ifile)%intRad1(icnt), &
00273         besttrackdata(ifile)%intRad2(icnt),    besttrackdata(ifile)%intRad3(icnt), &
00274         besttrackdata(ifile)%intRad4(icnt),    besttrackdata(ifile)%intPOuter(icnt), &
00275         besttrackdata(ifile)%intROuter(icnt),  besttrackdata(ifile)%intRmw(icnt),  &
00276         besttrackdata(ifile)%gusts(icnt),     besttrackdata(ifile)%eye(icnt),    &
00277         besttrackdata(ifile)%subregion(icnt),  besttrackdata(ifile)%maxseas(icnt), &
00278         besttrackdata(ifile)%initials(icnt),   besttrackdata(ifile)%dir(icnt),     &
00279         besttrackdata(ifile)%intSpeed(icnt),   besttrackdata(ifile)%stormName(icnt), &
00280         besttrackdata(ifile)%cycleNum(icnt)
00281
00282         !----- Convert lat/lon values to S/N and W/E notations
00283         IF (toupper(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
00284             besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00285         ELSE
00286             besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00287         END IF
00288
00289         IF (toupper(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00290             besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00291         ELSE
00292             besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00293         END IF
00294         !-----
00295
00296         !----- Get the year, month, day, hour from the DGT string
00297         READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00298         besttrackdata(ifile)%year(icnt)
00299         IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00300         READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00301         besttrackdata(ifile)%month(icnt)
00302         IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00303         READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00304         besttrackdata(ifile)%day(icnt)
00305         IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00306         READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)

```



```

besttrackdata(ifile)%hour(icnt)
00304     IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00305     !-----
00306     END IF
00307 END DO
00308
00309 10 IF (errio /= 0) THEN
00310     WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00311         ', while processing line: ' // trim(adjustl(inpline))
00312     CALL allmessage(error, scratchmessage)
00313
00314     CLOSE(iunit)
00315
00316     CALL unsetmessagesource()
00317
00318     CALL terminate()
00319 END IF
00320
00321 11 IF (ios /= 0) THEN
00322     WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00323         ', while processing line: ' // trim(adjustl(line))
00324     CALL allmessage(error, scratchmessage)
00325
00326     CLOSE(iunit)
00327
00328     CALL unsetmessagesource()
00329
00330     CALL terminate()
00331 END IF
00332
00333 20 CLOSE(iunit)
00334
00335 besttrackdata(ifile)%thisStorm = "
00336 besttrackdata(ifile)%loaded   = .true.
00337 besttrackdata(ifile)%numRec   = nlines
00338
00339
00340 !-----
00341 ! Get the unique storm name and store it in the thisStorm string
00342 ALLOCATE(chkarrstr(nlines))
00343 ALLOCATE(idxarrstr(nlines))
00344
00345 nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00346
00347 maxcnt = -1
00348 DO kcnt = 1, nunique
00349     kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00350     IF (kmax > maxcnt) THEN
00351         maxcnt = kmax
00352         besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00353     END IF
00354 END DO
00355
00356 DEALLOCATE(chkarrstr)
00357 DEALLOCATE(idxarrstr)
00358 !-----
00359
00360 !-----
00361 ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00362 ! stored in ascending order
00363 ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00364 ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00365
00366 CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00367
00368 IF (status /= 0) THEN
00369     CALL unsetmessagesource()
00370
00371     CALL terminate()
00372 END IF
00373
00374 ! Create the index array to be used in the comparison below
00375 idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00376
00377 IF (.NOT. arrayequal(idx0, idx1)) THEN
00378     besttrackdata(ifile)%basin   = besttrackdata(ifile)%basin(idx1)
00379     besttrackdata(ifile)%cyNum   = besttrackdata(ifile)%cyNum(idx1)
00380     besttrackdata(ifile)%dtg     = besttrackdata(ifile)%dtg(idx1)
00381     besttrackdata(ifile)%techNum = besttrackdata(ifile)%techNum(idx1)
00382     besttrackdata(ifile)%tech    = besttrackdata(ifile)%tech(idx1)
00383     besttrackdata(ifile)%tau     = besttrackdata(ifile)%tau(idx1)

```

```

00384     besttrackdata(ifile)%intLat = besttrackdata(ifile)%intLat(idxl)
00385     besttrackdata(ifile)%ns = besttrackdata(ifile)%ns(idxl)
00386     besttrackdata(ifile)%intLon = besttrackdata(ifile)%intLon(idxl)
00387     besttrackdata(ifile)%ew = besttrackdata(ifile)%ew(idxl)
00388     besttrackdata(ifile)%intVMax = besttrackdata(ifile)%intVMax(idxl)
00389     besttrackdata(ifile)%intMslp = besttrackdata(ifile)%intMslp(idxl)
00390     besttrackdata(ifile)%ty = besttrackdata(ifile)%ty(idxl)
00391     besttrackdata(ifile)%rad = besttrackdata(ifile)%rad(idxl)
00392     besttrackdata(ifile)%windCode = besttrackdata(ifile)%windCode(idxl)
00393     besttrackdata(ifile)%intRad1 = besttrackdata(ifile)%intRad1(idxl)
00394     besttrackdata(ifile)%intRad2 = besttrackdata(ifile)%intRad2(idxl)
00395     besttrackdata(ifile)%intRad3 = besttrackdata(ifile)%intRad3(idxl)
00396     besttrackdata(ifile)%intRad4 = besttrackdata(ifile)%intRad4(idxl)
00397     besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idxl)
00398     besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idxl)
00399     besttrackdata(ifile)%intRmw = besttrackdata(ifile)%intRmw(idxl)
00400     besttrackdata(ifile)%gusts = besttrackdata(ifile)%gusts(idxl)
00401     besttrackdata(ifile)%eye = besttrackdata(ifile)%eye(idxl)
00402     besttrackdata(ifile)%subregion = besttrackdata(ifile)%subregion(idxl)
00403     besttrackdata(ifile)%maxseas = besttrackdata(ifile)%maxseas(idxl)
00404     besttrackdata(ifile)%initials = besttrackdata(ifile)%initials(idxl)
00405     besttrackdata(ifile)%dir = besttrackdata(ifile)%dir(idxl)
00406     besttrackdata(ifile)%intSpeed = besttrackdata(ifile)%intSpeed(idxl)
00407     besttrackdata(ifile)%stormName = besttrackdata(ifile)%stormName(idxl)
00408     besttrackdata(ifile)%cycleNum = besttrackdata(ifile)%cycleNum(idxl)
00409     END IF
00410
00411     DEALLOCATE(idxl)
00412     DEALLOCATE(idxl)
00413     !-----
00414
00415     CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
00416     END DO ! End of "iFile" loop
00417
00418     CALL unsetmessagesource()
00419
00420     END SUBROUTINE readbesttrackfile
00421
00422 !=====
00423
00424 !-----
00425 ! SUBROUTINE READ CSV BEST TRACK FILE
00426 !-----
00427 !-----
00428
00429 SUBROUTINE readcsvbesttrackfile()
00430
00431     USE pahn_global, ONLY : nbtrfiles, besttrackfilename
00432     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique, &
00433                           intvalstr
00434     USE sortutils, ONLY : arth, indexx, arrayequal
00435     USE csv_module
00436
00437     IMPLICIT NONE
00438
00439     TYPE(csv_file) :: f
00440     CHARACTER(LEN=64), ALLOCATABLE :: sval2D(:, :)
00441     LOGICAL :: statusOK
00442
00443     CHARACTER(LEN=FNAMELEN) :: inpFile
00444     CHARACTER(LEN=512) :: line
00445     CHARACTER(LEN=64) :: tmpStr
00446
00447     INTEGER :: iFile, nLines, lenLine
00448     INTEGER :: iCnt, jCnt, kCnt, kMax ! loop counters
00449     INTEGER :: ios, status
00450
00451     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00452     INTEGER, ALLOCATABLE :: idxArrStr(:)
00453     INTEGER :: nUnique, maxCnt
00454
00455     INTEGER, ALLOCATABLE :: idx0(:), idx1(:)
00456
00457     CALL setmessagesource("ReadCsvBestTrackFile")
00458
00459     ! Allocate the best track structure array. This structure holds all the
00460     ! input values for the storm track as read in from the track input file
00461     ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00462     ! (as appropriate).
00463     ALLOCATE(besttrackdata(nbtrfiles))
00464
00465

```

```

00478      ! This is the main loop. We loop through all the best track files
00479      ! (user input)
00480      DO ifile = 1, nbtrfiles
00481         inpfname = besttrackfilename(ifile)
00482
00483         besttrackdata(ifile)%fileName = trim(adjustl(inpfname))
00484         besttrackdata(ifile)%thisStorm = ""
00485         besttrackdata(ifile)%loaded = .false.
00486         besttrackdata(ifile)%numRec = -1
00487
00488         CALL f%Read(trim(adjustl(inpfname)), status_ok=statusok)
00489         CALL f%Get(sval2d, status_ok=statusok)
00490
00491         ! Array allocation in the structure bestTrackData
00492         nlines = f%n_rows
00493         CALL allocbtrstruct(besttrackdata(ifile), nlines)
00494
00495         DO icnt = 1, nlines
00496            DO jcnt = 1, f%n_cols
00497               line = line // trim(adjustl(sval2d(icnt, jcnt)))
00498            END DO
00499            jcnt = 0
00500
00501            lenline = len_trim(adjustl(line))
00502
00503            IF (lenline /= 0) THEN
00504               !--- col: 1
00505               besttrackdata(ifile)%basin(icnt) = trim(adjustl(sval2d(icnt, 1)))
00506               !--- col: 2
00507               besttrackdata(ifile)%cyNum(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 2))))
00508               !--- col: 3
00509               besttrackdata(ifile)%dtg(icnt) = trim(adjustl(sval2d(icnt, 3)))
00510               !--- col: 4
00511               besttrackdata(ifile)%techNum(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 4))))
00512               !--- col: 5
00513               besttrackdata(ifile)%tech(icnt) = trim(adjustl(sval2d(icnt, 5)))
00514               !--- col: 6
00515               besttrackdata(ifile)%tau(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 6))))
00516               !--- col: 7
00517               tmpstr = trim(adjustl(sval2d(icnt, 7)))
00518               READ(tmpstr, '(i3, a1)') &
00519                 besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
00520               !--- col: 8
00521               tmpstr = trim(adjustl(sval2d(icnt, 8)))
00522               READ(tmpstr, '(i3, a1)') &
00523                 besttrackdata(ifile)%intLon(icnt), besttrackdata(ifile)%ew(icnt)
00524               !--- col: 9
00525               besttrackdata(ifile)%intVMax(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 9))))
00526               !--- col: 10
00527               besttrackdata(ifile)%intMslp(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 10))))
00528               !--- col: 11
00529               besttrackdata(ifile)%ty(icnt) = trim(adjustl(sval2d(icnt, 11)))
00530               !--- col: 12
00531               besttrackdata(ifile)%rad(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 12))))
00532               !--- col: 13
00533               besttrackdata(ifile)%windCode(icnt) = trim(adjustl(sval2d(icnt, 13)))
00534               !--- col: 14
00535               besttrackdata(ifile)%intRad1(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 14))))
00536               !--- col: 15
00537               besttrackdata(ifile)%intRad2(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 15))))
00538               !--- col: 16
00539               besttrackdata(ifile)%intRad3(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 16))))
00540               !--- col: 17
00541               besttrackdata(ifile)%intRad4(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 17))))
00542               !--- col: 18
00543               besttrackdata(ifile)%intPOuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 18))))
00544               !--- col: 19
00545               besttrackdata(ifile)%intROuter(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 19))))
00546               !--- col: 20
00547               besttrackdata(ifile)%intRmw(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 20))))
00548               !--- col: 21
00549               besttrackdata(ifile)%gusts(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 21))))
00550               !--- col: 22
00551               besttrackdata(ifile)%eye(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 22))))
00552               !--- col: 23
00553               besttrackdata(ifile)%subregion(icnt) = trim(adjustl(sval2d(icnt, 23)))
00554               !--- col: 24
00555               besttrackdata(ifile)%maxseas(icnt) = intvalstr(trim(adjustl(sval2d(icnt, 24))))
00556               !--- col: 25
00557               besttrackdata(ifile)%initials(icnt) = trim(adjustl(sval2d(icnt, 25)))
00558               !--- col: 26

```

```

00559     besttrackdata(ifile)%dir(icnt)      = intvalstr(trim(adjustl(sval2d(icnt, 26))))
00560     !--- col: 27
00561     besttrackdata(ifile)%intSpeed(icnt)  = intvalstr(trim(adjustl(sval2d(icnt, 27))))
00562     !--- col: 28
00563     besttrackdata(ifile)%stormName(icnt) = trim(adjustl(sval2d(icnt, 28)))
00564
00565     ! This is for the cycleNum, the last column we consider
00566     IF (icnt == 1) THEN
00567         kcmt = icnt
00568         besttrackdata(ifile)%cycleNum(icnt) = icnt
00569     ELSE
00570         kcmt = kcmt + 1
00571         IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00572             besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00573             kcmt = kcmt - 1
00574         ELSE
00575             besttrackdata(ifile)%cycleNum(icnt) = kcmt
00576         END IF
00577     END IF
00578
00579     !----- Convert lat/lon values to S/N and W/E notations
00580     IF (toupper(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
00581         besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00582     ELSE
00583         besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00584     END IF
00585
00586     IF (toupper(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00587         besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00588     ELSE
00589         besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00590     END IF
00591     !-----
00592
00593     !----- Get the year, month, day, hour from the DGT string
00594     READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00595     besttrackdata(ifile)%year(icnt)
00596     IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00597     READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00598     besttrackdata(ifile)%month(icnt)
00599     IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00600     READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00601     besttrackdata(ifile)%day(icnt)
00602     IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00603     READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00604     besttrackdata(ifile)%hour(icnt)
00605     IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00606     !-----
00607
00608     END IF
00609     END DO
00610
00611     besttrackdata(ifile)%thisStorm = "
00612     besttrackdata(ifile)%loaded   = .true.
00613     besttrackdata(ifile)%numRec   = nlines
00614
00615     !-----
00616     ! Get the unique storm name and store it in the thisStorm string
00617     ALLOCATE(chkarrstr(nlines))
00618     ALLOCATE(idxarrstr(nlines))
00619
00620     nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00621
00622     maxcnt = -1
00623     DO kcmt = 1, nunique
00624         kmax = count(chkarrstr(kcmt) == besttrackdata(ifile)%stormName)
00625         IF (kmax > maxcnt) THEN
00626             maxcnt = kmax
00627             besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcmt)))
00628         END IF
00629     END DO
00630
00631     DEALLOCATE(chkarrstr)
00632     DEALLOCATE(idxarrstr)
00633     !-----
00634
00635     !-----
00636     ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00637     ! stored in ascending order
00638     ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00639     ALLOCATE(idx1(besttrackdata(ifile)%numRec))

```

```

00636
00637     CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00638
00639     IF (status /= 0) THEN
00640         CALL unsetmessagesource()
00641
00642         CALL terminate()
00643     END IF
00644
00645     ! Create the index array to be used in the comparison below
00646     idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00647
00648     IF (.NOT. arrayequal(idx0, idx1)) THEN
00649         besttrackdata(ifile)%basin = besttrackdata(ifile)%basin(idx1)
00650         besttrackdata(ifile)%cyNum = besttrackdata(ifile)%cyNum(idx1)
00651         besttrackdata(ifile)%dtg = besttrackdata(ifile)%dtg(idx1)
00652         besttrackdata(ifile)%techNum = besttrackdata(ifile)%techNum(idx1)
00653         besttrackdata(ifile)%tech = besttrackdata(ifile)%tech(idx1)
00654         besttrackdata(ifile)%tau = besttrackdata(ifile)%tau(idx1)
00655         besttrackdata(ifile)%intLat = besttrackdata(ifile)%intLat(idx1)
00656         besttrackdata(ifile)%ns = besttrackdata(ifile)%ns(idx1)
00657         besttrackdata(ifile)%intLon = besttrackdata(ifile)%intLon(idx1)
00658         besttrackdata(ifile)%ew = besttrackdata(ifile)%ew(idx1)
00659         besttrackdata(ifile)%intVMax = besttrackdata(ifile)%intVMax(idx1)
00660         besttrackdata(ifile)%intMslp = besttrackdata(ifile)%intMslp(idx1)
00661         besttrackdata(ifile)%ty = besttrackdata(ifile)%ty(idx1)
00662         besttrackdata(ifile)%rad = besttrackdata(ifile)%rad(idx1)
00663         besttrackdata(ifile)%windCode = besttrackdata(ifile)%windCode(idx1)
00664         besttrackdata(ifile)%intRad1 = besttrackdata(ifile)%intRad1(idx1)
00665         besttrackdata(ifile)%intRad2 = besttrackdata(ifile)%intRad2(idx1)
00666         besttrackdata(ifile)%intRad3 = besttrackdata(ifile)%intRad3(idx1)
00667         besttrackdata(ifile)%intRad4 = besttrackdata(ifile)%intRad4(idx1)
00668         besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idx1)
00669         besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idx1)
00670         besttrackdata(ifile)%intRmw = besttrackdata(ifile)%intRmw(idx1)
00671         besttrackdata(ifile)%gusts = besttrackdata(ifile)%gusts(idx1)
00672         besttrackdata(ifile)%eye = besttrackdata(ifile)%eye(idx1)
00673         besttrackdata(ifile)%subregion = besttrackdata(ifile)%subregion(idx1)
00674         besttrackdata(ifile)%maxseas = besttrackdata(ifile)%maxseas(idx1)
00675         besttrackdata(ifile)%initials = besttrackdata(ifile)%initials(idx1)
00676         besttrackdata(ifile)%dir = besttrackdata(ifile)%dir(idx1)
00677         besttrackdata(ifile)%intSpeed = besttrackdata(ifile)%intSpeed(idx1)
00678         besttrackdata(ifile)%stormName = besttrackdata(ifile)%stormName(idx1)
00679         besttrackdata(ifile)%cycleNum = besttrackdata(ifile)%cycleNum(idx1)
00680     END IF
00681
00682     DEALLOCATE(idx0)
00683     DEALLOCATE(idx1)
00684     !-----
00685
00686     CALL f%Destroy()
00687
00688     CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile), '_fort22fmt')
00689
00690     END DO ! End of "iFile" loop
00691
00692     CALL unsetmessagesource()
00693
00694 END SUBROUTINE readcsvbesttrackfile
00695
00696 !=====
00697
00698 !-----
00699 ! SUBROUTINE PROCESS HOLLAND DATA
00700 !-----
00701
00702 SUBROUTINE processhollanddata(idTrFile, strOut, status)
00703
00704     USE pahm_global, ONLY : nm2m, kt2ms, nbtrfiles
00705     USE utilities, ONLY : touppercase, charunique
00706     USE timedateutils, ONLY : timeconv
00707     USE pahm_vortex, ONLY : calcintensitychange, uvtrans
00708
00709     IMPLICIT NONE
00710
00711     INTEGER, INTENT(IN) :: idTrFile
00712     TYPE(HollandData_T), INTENT(OUT) :: strOut
00713     INTEGER, INTENT(OUT) :: status ! error status
00714
00715     ! numUniqRec, outDTG, idxDTG are used to identify the unique DTG elements in the input structure
00716     INTEGER :: numUniqRec

```

```

00736 CHARACTER(LEN=10), ALLOCATABLE :: outDTG(:)
00737 INTEGER, ALLOCATABLE :: idxDTG(:)
00738
00739 INTEGER :: plIdx ! populated index for Holland Data array
00740 INTEGER :: iCnt ! loop counters
00741
00742 CHARACTER(LEN=4) :: castType !hindcast,forecast
00743 REAL(SZ), ALLOCATABLE :: castTime(:) ! seconds since start of year
00744
00745 REAL(SZ) :: spdVal, pressVal, rrpVal, rmwVal
00746
00747 status = 0 ! no error
00748
00749
00750 CALL setmessagesource("ProcessHollandData")
00751
00752 IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
00753     IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
00754         status = 2
00755
00756         WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ',
idtrfile
00757         CALL allmessage(error, scratchmessage)
00758
00759         CALL unsetmessagesource()
00760
00761         RETURN
00762     END IF
00763 ELSE
00764     status = 1
00765
00766     WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
00767         ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ',
nbtrfiles
00768     CALL allmessage(error, scratchmessage)
00769
00770     CALL unsetmessagesource()
00771
00772     RETURN
00773 END IF
00774
00775 WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
00776 CALL logmessage(info, scratchmessage)
00777
00778 ! Most likely the array size will be larger if repeated times are found
00779 ! in the best track structure.
00780 ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
00781 ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))
00782
00783 ! Get unique lines that represent new points in time.
00784 ! Repeated time points occur in hindcasts for the purpose of
00785 ! describing winds in the quadrants of the storm. We don't use the
00786 ! quadrant-by-quadrant wind data. Repeated time data occur in the
00787 ! forecast because the time data is just the time that the forecast
00788 ! was made. The important parameter in the forecast file is the
00789 ! forecast increment.
00790 numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)
00791
00792 !-----
00793 ! Populate the Holland structure
00794 !-----
00795 CALL allochollstruct(strout, numuniqrec)
00796
00797 ALLOCATE(casttime(numuniqrec))
00798
00799 strout%fileName = besttrackdata(idtrfile)%fileName
00800 strout%thisStorm = besttrackdata(idtrfile)%thisStorm
00801 strout%loaded = .true.
00802 strout%numRec = numuniqrec
00803
00804 WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
00805 CALL logmessage(info, scratchmessage)
00806
00807 DO icnt = 1, numuniqrec
00808     plidx = idxdtg(icnt)
00809
00810     casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(plidx))))
00811
00812     ! Convert speeds from knots to m/s
00813     spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)
00814

```

```

00815      ! Convert pressure(s) from mbar to Pa
00816      pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)
00817
00818      ! Convert all distances from nm to km/m
00819      rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
00820      rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx)    ! in m
00821
00822      strout%basin(icnt)      = besttrackdata(idtrfile)%basin(plidx)
00823      strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)
00824      strout%dtg(icnt)       = besttrackdata(idtrfile)%dtg(plidx)
00825      strout%year(icnt)      = besttrackdata(idtrfile)%year(plidx)
00826      strout%month(icnt)     = besttrackdata(idtrfile)%month(plidx)
00827      strout%day(icnt)       = besttrackdata(idtrfile)%day(plidx)
00828      strout%hour(icnt)      = besttrackdata(idtrfile)%hour(plidx)
00829      strout%castType(icnt)  = besttrackdata(idtrfile)%tech(plidx)
00830      strout%fcstInc(icnt)   = besttrackdata(idtrfile)%tau(plidx)
00831      strout%iLat(icnt)      = besttrackdata(idtrfile)%intLat(plidx)
00832      strout%lat(icnt)       = besttrackdata(idtrfile)%lat(plidx)
00833      strout%iLon(icnt)      = besttrackdata(idtrfile)%intLon(plidx)
00834      strout%lon(icnt)       = besttrackdata(idtrfile)%lon(plidx)
00835
00836      strout%iSpeed(icnt)    = besttrackdata(idtrfile)%intVMax(plidx)
00837      strout%speed(icnt)     = spdval
00838      strout%iCPress(icnt)   = besttrackdata(idtrfile)%intMslp(plidx)
00839      strout%cPress(icnt)    = pressval
00840      strout%iRrp(icnt)      = besttrackdata(idtrfile)%intROuter(plidx)
00841      strout%rrp(icnt)       = rrpval
00842      strout%iRmw(icnt)      = besttrackdata(idtrfile)%intRmw(plidx)
00843      strout%rmw(icnt)       = rmwval
00844
00845      ! PV check if this SELECT code is actually needed. Need to check the different format
00846      ! of input files.
00847      SELECT CASE(casttype)
00848      CASE("BEST")          ! nowcast/hindcast
00849          ! PV check if this is needed
00850          CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0,
00851            0.0_sz, casttime(icnt))
00852
00853      CASE("OFCL")          ! forecast
00854          ! PV check if this is needed
00855          IF (icnt > 1) THEN
00856              IF ((strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
00857                  cycle
00858              END IF
00859
00860              IF (strout%fcstInc(icnt) == 0) THEN
00861                  CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
00862                    strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
00863              ELSE
00864                  casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) *
00865                    3600.0_sz)
00866              END IF
00867
00868              IF ((strout%iCPress(icnt) == 0) .OR. (strout%iRmw(icnt) == 0)) THEN
00869                  CALL allmessage(error,
00870                    'The storm hindcast/forecast input file ' // trim(strout%fileName) // &
00871                    ' contains invalid data for central pressure or rMax.')
00872                  CALL terminate()
00873              END IF
00874
00875      ! Adding a new type to allow the analyst to add lines
00876      ! that do nothing but produce zero winds and background barometric
00877      ! pressure. These lines can have a date/time like a BEST line or
00878      ! a date/time and forecast period like an OFCL line.
00879      CASE("CALM")
00880          ! PV check if this is needed
00881          WRITE(scratchmessage, '(a)') 'The file: ' // trim(strout%fileName) // ' contains at least one
00882          "CALM" line.'
00883          CALL logmessage(echo, scratchmessage)
00884
00885          IF (icnt > 1) THEN
00886              IF ((strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
00887                  cycle
00888              END IF
00889
00890              IF (strout%fcstInc(icnt) == 0) THEN
00891                  CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
00892                    strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
00893              ELSE
00894                  casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1) *
00895                    3600.0_sz)

```

```

00890         END IF
00891
00892         CASE DEFAULT           ! unrecognized
00893             WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of '
// &
00894             trim(adjustl(strout%fileName))
00895             CALL allmessage(error, scratchmessage)
00896
00897             CALL terminate()
00898         END SELECT
00899
00900         strout%castTime(icnt) = casttime(icnt)
00901     END DO      ! numUniqRec
00902
00903     ! Calculate the cPress intensity change (dP/dt)
00904     CALL calcintensitychange(strout%cPress, casttime, strout%cPrDt, status, 2)
00905
00906     ! Calculate storm translation velocities based on change in position,
00907     ! approximate u and v translation velocities
00908     CALL uvtrans(strout%lat, strout%lon, casttime, strout%trVx, strout%trVy, status, 2)
00909
00910     DEALLOCATE(casttime)
00911 !-----
00912
00913     DEALLOCATE(outdtg)
00914     DEALLOCATE(idxdtg)
00915
00916     CALL unsetmessagesource()
00917
00918     END SUBROUTINE processhollanddata
00919
00920 !=====
00921
00922 !-----
00923 ! SUBROUTINE GET HOLLAND FIELDS
00924 !-----
00925
00926 !-----
00927
00928 SUBROUTINE gethollandfields(timeIDX)
00929
00930     USE pahm_mesh, ONLY : slam, sfea, xcslam, ycsfea, np, ismeshok
00931     USE pahm_global, ONLY : gravity, rhowater, rhoair,                                &
00932                             backgroundatmpress, bladjustfac, one2ten,                &
00933                             deg2rad, rad2deg, basee, omega, mb2pa, mb2kpa,            &
00934                             nbtrfiles, besttrackfilename,                          &
00935                             noutdt, mdbegsimtime, mdendsimtime, mdoutdt,            &
00936                             wvelx, wvely, wpress, times
00937     USE utilities, ONLY : sphericaldistance, sphericalfracpoint, getlocandratio
00938     USE timedateutils, ONLY : juldaytogreg, gregtojulday
00939     !USE PaHM_NetCDFIO
00940
00941     IMPLICIT NONE
00942
00943     INTEGER, INTENT(IN)                :: timeIDX
00944
00945     TYPE(hollanddata_t), ALLOCATABLE  :: holStru(:)           ! array of Holland data structures
00946     INTEGER                           :: stormNumber          ! storm identification number
00947     REAL(SZ)                          :: h1B                 ! Holland B parameter
00948     REAL(SZ)                          :: rrp                  ! radius of the last closed isobar (m)
00949     REAL(SZ)                          :: rmw                  ! radius of max winds (m)
00950     REAL(SZ)                          :: speed                ! maximum sustained wind speed (m/s)
00951     REAL(SZ)                          :: cPress               ! central pressure (Pa)
00952     REAL(SZ)                          :: cPressDef            ! pressure deficit: Ambient Press - cPress
00953     (Pa)
00954     REAL(SZ)                          :: trVX, trVY, trSPD     ! storm translation velocities (m/s)
00955     REAL(SZ)                          :: trSpdX, trSpdY        ! adjusted translation velocities (m/s)
00956     REAL(SZ)                          :: lon, lat              ! current eye location
00957
00958     REAL(SZ), ALLOCATABLE              :: rad(:)              ! distance of nodal points from the eye
00959     location
00960     INTEGER, ALLOCATABLE              :: radIDX(:)            ! indices of nodal points such that rad <=
00961     rrp
00962     INTEGER                           :: maxRadIDX             ! total number of radIDX elements
00963     REAL(SZ)                          :: windMultiplier        ! for storm 2 in lpfs ensemble DO WE NEED
00964     THIS?
00965     REAL(SZ)                          :: dx, dy, theta
00966     REAL(SZ)                          :: wtRatio
00967     REAL(SZ)                          :: coriolis
00968
00969     REAL(SZ)                          :: sfPress               ! calculated surface MSL pressure (Pa)
00970     REAL(SZ)                          :: grVel                 ! wind speed (m/s) at gradient level (top
00971     of ABL)

```



```

00986      REAL(SZ)                                :: sfVelX, sfVelY      ! calculated surface (10-m above ground)
      wind velocities (m/s)

00987
00988      INTEGER                                :: iCnt, stCnt, npCnt
00989      INTEGER                                :: i, j11, j12
00990      INTEGER                                :: status
00991
00992      CHARACTER(LEN=64)                      :: tmpTimeStr, tmpStr1, tmpStr2
00993
00994      LOGICAL, SAVE                          :: firstCall = .true.
00995
00996      CALL setmessagesource("GetHollandFields")
00997
00998      ! Check if timeIDX is within bounds (1 <= timeIDX <= nOutDT). If it is not then exit the program.
00999      IF ((timeidx < 1) .OR. (timeidx > noutdt)) THEN
01000          WRITE(tmpstr1, '(a, i0)') 'timeIDX = ', timeidx
01001          WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
01002          WRITE(scratchmessage, '(a)') 'timeIDX should be: 1 <= timeIDX <= nOutDT : ' // &
01003              trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
01004          CALL allmessage(error, scratchmessage)
01005
01006          CALL unsetmessagesource()
01007
01008          CALL terminate()
01009      END IF
01010
01011      ! This part of the code should only be executed just once
01012      IF (firstcall) THEN
01013          firstcall = .false.
01014
01015          ! Check if the mesh variables are set and that nOutDT is greater than zero.
01016          IF (.NOT. ismeshok) THEN
01017              WRITE(scratchmessage, '(a)') 'The mesh variables are not established properly. ' // &
01018                  'Call subroutine ReadMesh to read/create the mesh topology first.'
01019              CALL allmessage(error, scratchmessage)
01020
01021              CALL unsetmessagesource()
01022
01023              CALL terminate()
01024          ELSE
01025              IF ((np <= 0) .OR. (noutdt <= 0)) THEN
01026                  WRITE(tmpstr1, '(a, i0)') 'np = ', np
01027                  WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
01028                  WRITE(scratchmessage, '(a)') 'Variables "np" or "nOutDT" are not defined properly: ' // &
01029                      trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
01030                  CALL allmessage(error, scratchmessage)
01031
01032                  CALL unsetmessagesource()
01033
01034                  CALL terminate()
01035              END IF
01036          END IF
01037
01038          ! Allocate storage for the Times array that contains the output times.
01039          ALLOCATE(times(noutdt))
01040          DO icnt = 1, noutdt
01041              times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
01042          END DO
01043      END IF
01044
01045      !-----
01046      ! Allocate storage for required arrays.
01047      IF (.NOT. ALLOCATED(wvelx)) ALLOCATE(wvelx(np))
01048      IF (.NOT. ALLOCATED(wvely)) ALLOCATE(wvely(np))
01049      IF (.NOT. ALLOCATED(wpress)) ALLOCATE(wpress(np))
01050
01051      ! Initialize the arrays. Here we are resetting the fields to their defaults.
01052      ! This subroutine is called repeatedly and its time the following fields
01053      ! are recalculated.
01054      wvelx = 0.0_sz
01055      wvely = wvelx
01056      wpress = backgroundatmpress * mb2pa
01057      !-----
01058
01059      !-----
01060      ! ALLOCATE THE HOLLAND DATA STRUCTURES AND STORE THE HOLLAND
01061      ! DATA INTO THE DATA STRUCTURE ARRAY FOR SUBSEQUENT USE
01062      !-----
01063      !
01064      !
01065      ! Allocate the array of Holland data structures. The Holland

```

```

01066      ! structures are allocated by calling the ProcessHollandData
01067      ! subroutine.
01068      ALLOCATE(holstru(nbtrfiles))
01069
01070      ! Process and store the "best track" data into the array of Holland structures
01071      ! for subsequent use. All required data to generate the P-W model wind fields
01072      ! are contained in these structures. We take into consideration that might be
01073      ! more than one "best track" file for the simulation period.
01074      DO stcnt = 1, nbtrfiles
01075          CALL processhollanddata(stcnt, holstru(stcnt), status)
01076
01077          IF (.NOT. holstru(stcnt)%loaded) THEN
01078              WRITE(scratchmessage, '(a)') 'There was an error loading the Holland data structure for the best
track file: ' // &
01079                  trim(adjustl(besttrackfilename(stcnt)))
01080              CALL allmessage(error, scratchmessage)
01081
01082              CALL deallochoollstruct(holstru(stcnt))
01083              DEALLOCATE(holstru)
01084
01085              CALL unsetmessagesource()
01086
01087              CALL terminate()
01088          ELSE IF (status /= 0) THEN
01089              WRITE(scratchmessage, '(a)') 'There was an error processing the Holland data structure for the
best track file: ' // &
01090                  trim(adjustl(besttrackfilename(stcnt)))
01091              CALL allmessage(error, scratchmessage)
01092
01093              CALL deallochoollstruct(holstru(stcnt))
01094              DEALLOCATE(holstru)
01095
01096              CALL unsetmessagesource()
01097
01098              CALL terminate()
01099          ELSE
01100              WRITE(scratchmessage, '(a)') 'Processing the Holland data structure for the best track file: ' //
&
01101                  trim(adjustl(besttrackfilename(stcnt)))
01102              CALL logmessage(info, scratchmessage)
01103          END IF
01104      END DO
01105      !-----
01106
01107      !-----
01108      ! THIS IS THE MAIN TIME LOOP      timeIDX
01109      !-----
01110      ! WRITE(scratchMessage, '(a)') 'Start of the main time loop'
01111      ! CALL AllMessage(INFO, scratchMessage)
01112      ! DO iCnt = 1, nOutDT
01113          icnt = timeidx
01114          WRITE(tmpstr1, '(i5)') icnt
01115          WRITE(tmpstr2, '(i5)') noutdt
01116          tmpstr1 = '(' // trim(tmpstr1) // '/' // trim(adjustl(tmpstr2)) // ')'
01117          WRITE(tmptimestr, '(f20.3)') times(icnt)
01118          WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(tmpstr1)) // " " //
trim(adjustl(tmptimestr))
01119          CALL allmessage(scratchmessage)
01120
01121          DO stcnt = 1, nbtrfiles
01122              ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
01123              ! factor for the subsequent linear interpolation in time. In order for this to
01124              ! work, the array holStru%castTime should be ordered in ascending order.
01125              CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, j11, j12, wtratio)
01126
01127              ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
01128              ! by exiting this loop
01129              IF ((j11 <= 0) .OR. (j12 <= 0)) THEN
01130                  WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(tmptimestr)) // &
' , skipping generating data for this time'
01131                  CALL logmessage(info, scratchmessage)
01132              END IF
01133
01134              EXIT
01135          END IF
01136
01137          ! Perform linear interpolation in time
01138          stormnumber = holstru(stcnt)%stormNumber(j11)
01139
01140          CALL sphericalfracpoint(holstru(stcnt)%lat(j11), holstru(stcnt)%lon(j11), &
holstru(stcnt)%lat(j12), holstru(stcnt)%lon(j12), &
wtratio, lat, lon)
01142

```

```

01143      !lat   = holStru(stCnt)%lat(jl1) + &
01144      !       wtratio * (holStru(stCnt)%lat(jl2) - holStru(stCnt)%lat(jl1))
01145      !lon   = holStru(stCnt)%lon(jl1) + &
01146      !       wtratio * (holStru(stCnt)%lon(jl2) - holStru(stCnt)%lon(jl1))
01147
01148      ! Radius of the last closed isobar
01149      rrp = holstru(stcnt)%rrp(jl1) + &
01150           wtratio * (holstru(stcnt)%rrp(jl2) - holstru(stcnt)%rrp(jl1))
01151
01152      ! Radius of maximum winds
01153      rmw = holstru(stcnt)%rmw(jl1) + &
01154           wtratio * (holstru(stcnt)%rmw(jl2) - holstru(stcnt)%rmw(jl1))
01155
01156      ! Get all the distances of the mesh nodes from (lat, lon)
01157      rad = sphericaldistance(sfea, slam, lat, lon)
01158      ! ... and the indices of the nodal points where rad <= rrp
01159      radidx = pack([(i, i = 1, np)], rad <= rrp)
01160      maxradidx = SIZE(radidx)
01161
01162      ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
01163      IF (maxradidx == 0) THEN
01164          WRITE(tmpstr1, '(f20.3)') rrp
01165          tmpstr1 = ' (rrp = ' // trim(adjustl(tmpstr1)) // ' m)'
01166          WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar'
01167      ' // &
01168          trim(adjustl(tmpstr1)) // ' for storm: ' // &
01169          trim(adjustl(holstru(stcnt)%thisStorm))
01170          CALL logmessage(info, scratchmessage)
01171
01172      EXIT
01173      END IF
01174
01175      speed = holstru(stcnt)%speed(jl1) + &
01176            wtratio * (holstru(stcnt)%speed(jl2) - holstru(stcnt)%speed(jl1))
01177
01178      cpress = holstru(stcnt)%cPress(jl1) + &
01179            wtratio * (holstru(stcnt)%cPress(jl2) - holstru(stcnt)%cPress(jl1))
01180
01181      trvx = holstru(stcnt)%trVx(jl1) + &
01182            wtratio * (holstru(stcnt)%trVx(jl2) - holstru(stcnt)%trVx(jl1))
01183      trvy = holstru(stcnt)%trVy(jl1) + &
01184            wtratio * (holstru(stcnt)%trVy(jl2) - holstru(stcnt)%trVy(jl1))
01185
01186      ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
01187      ! background pressure and return. PV: check if this is actually needed
01188      IF (cpress < 0.0_sz) THEN
01189          wvelx = 0.0_sz
01190          wvely = wvelx
01191          wpress = backgroundatmpress * mb2pa
01192
01193      WRITE(scratchmessage, '(a)') 'Calm period found, generating zero atmospheric fields for this
time'
01194      CALL logmessage(info, scratchmessage)
01195
01196      EXIT
01197      END IF
01198
01199      ! Calculate and limit central pressure deficit; some track files (e.g., Charley 2004)
01200      ! may have a central pressure greater than the ambient pressure that this subroutine assumes
01201      cpressdef = backgroundatmpress * mb2pa - cpress
01202      IF (cpressdef < 100.0_sz) cpressdef = 100.0_sz
01203
01204      ! Subtract the translational speed of the storm from the observed max wind speed to avoid
01205      ! distortion in the Holland curve fit. The translational speed will be added back later.
01206      trspd = sqrt(trvx * trvx + trvy * trvy)
01207      speed = speed - trspd
01208
01209      ! Convert wind speed from 10 meter altitude (which is what the
01210      ! NHC forecast contains) to wind speed at the top of the atmospheric
01211      ! boundary layer (which is what the Holland curve fit requires).
01212      speed = speed / bladjustfac
01213
01214      ! Calculate Holland parameters and limit the result to its appropriate range.
01215      hlb = rhoair * basee * (speed**2) / cpressdef
01216      IF (hlb < 1.0_sz) hlb = 1.0_sz
01217      IF (hlb > 2.5_sz) hlb = 2.5_sz
01218
01219      ! If we are running storm 2 in the Lake Pontchartrain !PV Do we need this?
01220      ! Forecast System ensemble, the final wind speeds should be multiplied by 1.2.
01221      windmultiplier = 1.0_sz
01222      IF (stormnumber == 2) windmultiplier = 1.2_sz

```

```

01222
01223      DO npcnt = 1, maxradidx
01224         i = radidx(npncnt)
01225
01226         dx = sphericaldistance(lat, lon, lat, slam(i))
01227         dy = sphericaldistance(lat, lon, sfea(i), lon)
01228         theta = atan2(dy, dx)
01229
01230         ! Compute coriolis
01231         coriolis = 2.0_sz * omega * sin(sfea(i) * deg2rad)
01232
01233         ! Compute the pressure (Pa) at a distance rad(i); all distances are in meters
01234         sfpress = cpress + cpressdef * exp(-(rmw / rad(i))**hlb)
01235
01236         ! Compute wind speed (speed - trSPD) at gradient level (m/s) and at a distance rad(i);
01237         ! all distances are in meters. Using absolute value for coriolis for Southern Hemisphere
01238         grvel = sqrt(speed**2 * (rmw / rad(i))**hlb * exp(1.0_sz - (rmw / rad(i))**hlb) + &
01239             (rad(i) * abs(coriolis) / 2.0_sz)**2) - &
01240             rad(i) * abs(coriolis) / 2.0_sz
01241
01242         ! Determine translation speed that should be added to final !PV CHECK ON THIS
01243         ! storm wind speed. This is tapered to zero as the storm wind tapers
01244         ! to zero toward the eye of the storm and at long distances from the storm.
01245         trspdx = (abs(grvel) / speed) * trvx
01246         trspdy = (abs(grvel) / speed) * trvy
01247
01248         ! Apply mutliplier for Storm #2 in LPFS ensemble.
01249         grvel = grvel * windmultiplier
01250
01251         ! Find the wind velocity components.
01252         sfvelx = -grvel * sin(theta)
01253         sfvely = grvel * cos(theta)
01254         !print *, sfVelX, sfVelY
01255         ! Convert wind velocity from the gradient level (top of atmospheric boundary layer)
01256         ! which, is what the Holland curve fit produces, to 10-m wind velocity.
01257         sfvelx = sfvelx * bladjustfac
01258         sfvely = sfvely * bladjustfac
01259         !print *, sfVelX, sfVelY
01260         ! Convert from 1-minute averaged winds to 10-minute averaged winds.
01261         sfvelx = sfvelx * one2ten
01262         sfvely = sfvely * one2ten
01263         !print *, sfVelX, sfVelY
01264         ! Add back the storm translation speed.
01265         sfvelx = sfvelx + trspdx
01266         sfvely = sfvely + trspdy
01267
01268         !print *, sfVelX, sfVelY, wVelX(i), wVelY(i)
01269         !PV Need to interpolate between storms if this nodal point
01270         ! is affected by more than on storm
01271         wpress(i) = sfpress
01272         wvelx(i) = sfvelx
01273         wvely(i) = sfvely
01274
01275         !print *, sfVelX, sfVelY, wVelX(i), wVelY(i)
01276         !print *, '-----'
01277      END DO ! npCnt = 1, maxRadIDX
01278
01279      END DO ! stCnt = 1, nbTrFiles
01280      END DO ! iCnt = 1, nOutDT
01281      ! WRITE(scratchMessage, '(a)') 'End of the main time loop'
01282      ! CALL AllMessage(INFO, scratchMessage)
01283
01284      !----- Deallocate the arrays
01285      IF (ALLOCATED(rad)) DEALLOCATE(rad)
01286      IF (ALLOCATED(radidx)) DEALLOCATE(radidx)
01287      DO icnt = 1, nbtrfiles
01288         CALL deallochoollstruct(holstru(icnt))
01289      END DO
01290      DEALLOCATE(holstru)
01291      !-----
01292
01293      CALL unsetmessagesource()
01294
01295      END SUBROUTINE gethollandfields
01296
01297      !=====
01298
01299      !-----
01300      ! SUBROUTINE WRITE BEST TRACK DATA
01301      !-----
01317

```

```

01318 SUBROUTINE writebesttrackdata(inpFile, btrStruc, suffix)
01319
01320   USE pahm_global, ONLY : lun_btrk, lun_btrk1
01321
01322   IMPLICIT NONE
01323
01324   ! Global variables
01325   CHARACTER(LEN=*) :: inpFile
01326   TYPE(besttrackdata_t), INTENT(IN) :: btrStruc
01327   CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: suffix
01328
01329   ! Local variables
01330   CHARACTER(LEN=FNAMELEN) :: outFile
01331   CHARACTER(LEN=64) :: fSuf
01332   INTEGER :: iCnt
01333   INTEGER :: iUnit, errIO
01334   CHARACTER(LEN=512) :: fmtStr
01335
01336   !----- Initialize variables
01337   iunit = lun_btrk1
01338   errio = 0
01339
01340   fmtstr = ' (a2, ",", 1x, i2.2, ",", 1x, a10, ",", 1x, i2, ",", 1x, a4, ",", 1x, i3, ",", 1x, i3, a1,
01341   ", ",", 1x, i4, a1, ",", '
01342   fmtstr = trim(fmtstr) // ' 1x, i3, ",", 1x, i4, ",", 1x, a2, ",", 1x, i3, ",", 1x, a3, ",", '
01343   fmtstr = trim(fmtstr) // ' 4(1x, i4, ",", 1x, i4, ",", 1x, i4, ",", 1x, i3, ",", 1x, i4, ",", 1x,
01344   i3, ",", '
01345   fmtstr = trim(fmtstr) // ' 1x, a3, ",", 1x, i3, ",", 1x, a3, ",", i3, ",", 1x, i3, ",", 1x, a11, ",", 1x,
01346   i3, ",", '
01347   !-----
01348   fsuf = '_adj'
01349   IF (PRESENT(suffix)) fsuf = adjustl(suffix)
01350   CALL setmessagesource("WriteBestTrackData")
01351
01352   IF (.NOT. btrstruc%loaded) THEN
01353     WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be
01354     written."
01355     CALL allmessage(info, scratchmessage)
01356     RETURN
01357   END IF
01358
01359   outfile = trim(adjustl(inpfile)) // trim(fsuf)
01360
01361   WRITE(scratchmessage, '(a)') 'Writing the "adjusted" best track data to: ' // trim(adjustl(outfile))
01362   CALL logmessage(info, scratchmessage)
01363
01364   OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)
01365
01366   IF (errio /= 0) THEN
01367     WRITE(scratchmessage, '(a)') 'Error opening the outFile: ' // trim(outfile) // &
01368     ', skip writing the "adjusted" best track fields'
01369     CALL allmessage(error, scratchmessage)
01370   END IF
01371
01372   RETURN
01373 END IF
01374
01375 DO icnt = 1, btrstruc%numRec
01376   WRITE(iunit, fmtstr)
01377   btrstruc%basin(icnt), btrstruc%cyNum(icnt), &
01378   btrstruc%dtg(icnt), btrstruc%techNum(icnt), &
01379   btrstruc%tech(icnt), btrstruc%tau(icnt), &
01380   btrstruc%intLat(icnt), btrstruc%ns(icnt), &
01381   btrstruc%intLon(icnt), btrstruc%ew(icnt), &
01382   btrstruc%intVMax(icnt), btrstruc%intMslp(icnt), &
01383   btrstruc%ty(icnt), btrstruc%rad(icnt), &
01384   btrstruc%windCode(icnt), btrstruc%intRad1(icnt), &
01385   btrstruc%intRad2(icnt), btrstruc%intRad3(icnt), &
01386   btrstruc%intRad4(icnt), btrstruc%intPOuter(icnt), &
01387   btrstruc%intROuter(icnt), btrstruc%intRmw(icnt), &
01388   btrstruc%gusts(icnt), btrstruc%eye(icnt), &
01389   btrstruc%subregion(icnt), btrstruc%maxseas(icnt), &
01390   btrstruc%initials(icnt), btrstruc%dir(icnt), &
01391   btrstruc%intSpeed(icnt), btrstruc%stormName(icnt), &
01392   btrstruc%cycleNum(icnt)
01393 END DO
01394 CLOSE(iunit)

```

```

01395
01396     CALL unsetmessagesource()
01397
01398 END SUBROUTINE writebesttrackdata
01399
01400 !=====
01401
01402 !-----
01403 ! SUBROUTINE ALLOC BTR STRUCT
01404 !-----
01405 !-----
01406 SUBROUTINE allocbtrstruct(str, nRec)
01407
01408 IMPLICIT NONE
01409
01410 TYPE(besttrackdata_t) :: str
01411 INTEGER, INTENT(IN) :: nRec
01412
01413 str%numRec = nrec
01414 str%loaded = .false.
01415
01416 !----- Input parameters
01417 IF (.NOT. ALLOCATED(str%basin)) ALLOCATE(str%basin(nrec))
01418 IF (.NOT. ALLOCATED(str%cyNum)) ALLOCATE(str%cyNum(nrec))
01419 IF (.NOT. ALLOCATED(str%dtg)) ALLOCATE(str%dtg(nrec))
01420 IF (.NOT. ALLOCATED(str%techNum)) ALLOCATE(str%techNum(nrec))
01421 IF (.NOT. ALLOCATED(str%tech)) ALLOCATE(str%tech(nrec))
01422 IF (.NOT. ALLOCATED(str%tau)) ALLOCATE(str%tau(nrec))
01423 IF (.NOT. ALLOCATED(str%intLat)) ALLOCATE(str%intLat(nrec))
01424 IF (.NOT. ALLOCATED(str%intLon)) ALLOCATE(str%intLon(nrec))
01425 IF (.NOT. ALLOCATED(str%ew)) ALLOCATE(str%ew(nrec))
01426 IF (.NOT. ALLOCATED(str%ns)) ALLOCATE(str%ns(nrec))
01427 IF (.NOT. ALLOCATED(str%intVMax)) ALLOCATE(str%intVMax(nrec))
01428 IF (.NOT. ALLOCATED(str%intMslp)) ALLOCATE(str%intMslp(nrec))
01429 IF (.NOT. ALLOCATED(str%ty)) ALLOCATE(str%ty(nrec))
01430 IF (.NOT. ALLOCATED(str%rad)) ALLOCATE(str%rad(nrec))
01431 IF (.NOT. ALLOCATED(str%windCode)) ALLOCATE(str%windCode(nrec))
01432 IF (.NOT. ALLOCATED(str%intRad1)) ALLOCATE(str%intRad1(nrec))
01433 IF (.NOT. ALLOCATED(str%intRad2)) ALLOCATE(str%intRad2(nrec))
01434 IF (.NOT. ALLOCATED(str%intRad3)) ALLOCATE(str%intRad3(nrec))
01435 IF (.NOT. ALLOCATED(str%intRad4)) ALLOCATE(str%intRad4(nrec))
01436 IF (.NOT. ALLOCATED(str%intPOuter)) ALLOCATE(str%intPOuter(nrec))
01437 IF (.NOT. ALLOCATED(str%intROuter)) ALLOCATE(str%intROuter(nrec))
01438 IF (.NOT. ALLOCATED(str%intRmw)) ALLOCATE(str%intRmw(nrec))
01439 IF (.NOT. ALLOCATED(str%gusts)) ALLOCATE(str%gusts(nrec))
01440 IF (.NOT. ALLOCATED(str%eye)) ALLOCATE(str%eye(nrec))
01441 IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
01442 IF (.NOT. ALLOCATED(str%maxseas)) ALLOCATE(str%maxseas(nrec))
01443 IF (.NOT. ALLOCATED(str%initials)) ALLOCATE(str%initials(nrec))
01444 IF (.NOT. ALLOCATED(str%dir)) ALLOCATE(str%dir(nrec))
01445 IF (.NOT. ALLOCATED(str%intSpeed)) ALLOCATE(str%intSpeed(nrec))
01446 IF (.NOT. ALLOCATED(str%stormName)) ALLOCATE(str%stormName(nrec))
01447 IF (.NOT. ALLOCATED(str%cycleNum)) ALLOCATE(str%cycleNum(nrec))
01448
01449 !----- Converted parameters
01450 IF (.NOT. ALLOCATED(str%year)) ALLOCATE(str%year(nrec))
01451 IF (.NOT. ALLOCATED(str%month)) ALLOCATE(str%month(nrec))
01452 IF (.NOT. ALLOCATED(str%day)) ALLOCATE(str%day(nrec))
01453 IF (.NOT. ALLOCATED(str%hour)) ALLOCATE(str%hour(nrec))
01454 IF (.NOT. ALLOCATED(str%lat)) ALLOCATE(str%lat(nrec))
01455 IF (.NOT. ALLOCATED(str%lon)) ALLOCATE(str%lon(nrec))
01456
01457 END SUBROUTINE allocbtrstruct
01458
01459 !=====
01460
01461 !-----
01462 ! SUBROUTINE DEALLOC BTR STRUCT
01463 !-----
01464 !-----
01465 SUBROUTINE deallocbtrstruct(str)
01466
01467 IMPLICIT NONE
01468
01469 TYPE(besttrackdata_t) :: str
01470
01471 str%numRec = -1
01472 str%loaded = .false.
01473
01474 !----- Input parameters
01475 IF (ALLOCATED(str%basin)) DEALLOCATE(str%basin)

```

```

01498     IF (ALLOCATED(str%cyNum))      DEALLOCATE(str%cyNum)
01499     IF (ALLOCATED(str%dtg))          DEALLOCATE(str%dtg)
01500     IF (ALLOCATED(str%techNum))      DEALLOCATE(str%techNum)
01501     IF (ALLOCATED(str%tech))          DEALLOCATE(str%tech)
01502     IF (ALLOCATED(str%tau))           DEALLOCATE(str%tau)
01503     IF (ALLOCATED(str%intLat))        DEALLOCATE(str%intLat)
01504     IF (ALLOCATED(str%intLon))        DEALLOCATE(str%intLon)
01505     IF (ALLOCATED(str%ew))            DEALLOCATE(str%ew)
01506     IF (ALLOCATED(str%ns))            DEALLOCATE(str%ns)
01507     IF (ALLOCATED(str%intVMax))       DEALLOCATE(str%intVMax)
01508     IF (ALLOCATED(str%intMslp))       DEALLOCATE(str%intMslp)
01509     IF (ALLOCATED(str%ty))            DEALLOCATE(str%ty)
01510     IF (ALLOCATED(str%rad))           DEALLOCATE(str%rad)
01511     IF (ALLOCATED(str%windCode))      DEALLOCATE(str%windCode)
01512     IF (ALLOCATED(str%intRad1))       DEALLOCATE(str%intRad1)
01513     IF (ALLOCATED(str%intRad2))       DEALLOCATE(str%intRad2)
01514     IF (ALLOCATED(str%intRad3))       DEALLOCATE(str%intRad3)
01515     IF (ALLOCATED(str%intRad4))       DEALLOCATE(str%intRad4)
01516     IF (ALLOCATED(str%intPOuter))     DEALLOCATE(str%intPOuter)
01517     IF (ALLOCATED(str%intROuter))     DEALLOCATE(str%intROuter)
01518     IF (ALLOCATED(str%intRmw))        DEALLOCATE(str%intRmw)
01519     IF (ALLOCATED(str%gusts))          DEALLOCATE(str%gusts)
01520     IF (ALLOCATED(str%eye))            DEALLOCATE(str%eye)
01521     IF (ALLOCATED(str%subregion))      DEALLOCATE(str%subregion)
01522     IF (ALLOCATED(str%maxseas))        DEALLOCATE(str%maxseas)
01523     IF (ALLOCATED(str%initials))      DEALLOCATE(str%initials)
01524     IF (ALLOCATED(str%dir))            DEALLOCATE(str%dir)
01525     IF (ALLOCATED(str%intSpeed))       DEALLOCATE(str%intSpeed)
01526     IF (ALLOCATED(str%stormName))     DEALLOCATE(str%stormName)
01527     IF (ALLOCATED(str%cycleNum))      DEALLOCATE(str%cycleNum)
01528
01529     !----- Converted parameters
01530     IF (ALLOCATED(str%year))           DEALLOCATE(str%year)
01531     IF (ALLOCATED(str%month))          DEALLOCATE(str%month)
01532     IF (ALLOCATED(str%day))            DEALLOCATE(str%day)
01533     IF (ALLOCATED(str%hour))           DEALLOCATE(str%hour)
01534     IF (ALLOCATED(str%lat))            DEALLOCATE(str%lat)
01535     IF (ALLOCATED(str%lon))            DEALLOCATE(str%lon)
01536
01537     END SUBROUTINE deallocbtrstruct
01538
01539 !=====
01540
01541 !-----
01542 ! SUBROUTINE ALLOC HOLL STRUCT
01543 !-----
01544 !-----
01545 SUBROUTINE alloc Hollstruct(str, nRec)
01546
01547     IMPLICIT NONE
01548
01549     TYPE(hollanddata_t) :: str
01550     INTEGER, INTENT(IN) :: nRec
01551
01552     str%numRec = nrec
01553     str%loaded = .false.
01554
01555     !----- Input parameters
01556     IF (.NOT. ALLOCATED(str%basin))    ALLOCATE(str%basin(nrec))
01557
01558     IF (.NOT. ALLOCATED(str%dtg))      ALLOCATE(str%dtg(nrec))
01559     IF (.NOT. ALLOCATED(str%stormNumber)) ALLOCATE(str%stormNumber(nrec))
01560     IF (.NOT. ALLOCATED(str%year))     ALLOCATE(str%year(nrec))
01561     IF (.NOT. ALLOCATED(str%month))    ALLOCATE(str%month(nrec))
01562     IF (.NOT. ALLOCATED(str%day))      ALLOCATE(str%day(nrec))
01563     IF (.NOT. ALLOCATED(str%hour))     ALLOCATE(str%hour(nrec))
01564
01565     IF (.NOT. ALLOCATED(str%castTime)) ALLOCATE(str%castTime(nrec))
01566     IF (.NOT. ALLOCATED(str%castType)) ALLOCATE(str%castType(nrec))
01567     IF (.NOT. ALLOCATED(str%fcstInc))  ALLOCATE(str%fcstInc(nrec))
01568
01569     IF (.NOT. ALLOCATED(str%iLat))     ALLOCATE(str%iLat(nrec))
01570     IF (.NOT. ALLOCATED(str%lat))      ALLOCATE(str%lat(nrec))
01571     IF (.NOT. ALLOCATED(str%iLon))     ALLOCATE(str%iLon(nrec))
01572     IF (.NOT. ALLOCATED(str%lon))      ALLOCATE(str%lon(nrec))
01573
01574     IF (.NOT. ALLOCATED(str%iSpeed))   ALLOCATE(str%iSpeed(nrec))
01575     IF (.NOT. ALLOCATED(str%speed))    ALLOCATE(str%speed(nrec))
01576
01577     IF (.NOT. ALLOCATED(str%iCPress))  ALLOCATE(str%iCPress(nrec))
01578     IF (.NOT. ALLOCATED(str%cPress))   ALLOCATE(str%cPress(nrec))
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590

```

```

01591
01592     IF (.NOT. ALLOCATED(str%iRrp))          ALLOCATE(str%iRrp(nrec))
01593     IF (.NOT. ALLOCATED(str%rrp))          ALLOCATE(str%rrp(nrec))
01594
01595     IF (.NOT. ALLOCATED(str%iRmw))          ALLOCATE(str%iRmw(nrec))
01596     IF (.NOT. ALLOCATED(str%rmw))          ALLOCATE(str%rmw(nrec))
01597
01598     IF (.NOT. ALLOCATED(str%cPrDt))         ALLOCATE(str%cPrDt(nrec))
01599
01600     IF (.NOT. ALLOCATED(str%trVx))          ALLOCATE(str%trVx(nrec))
01601     IF (.NOT. ALLOCATED(str%trVy))          ALLOCATE(str%trVy(nrec))
01602
01603     END SUBROUTINE allochollstruct
01604
01605 !=====
01606
01607 !-----
01608 ! SUBROUTINE DEALLOC HOLL STRUCT
01609 !-----
01610 !-----
01621 SUBROUTINE deallochollstruct(str)
01622
01623     IMPLICIT NONE
01624
01625     TYPE(hollanddata_t), INTENT(OUT) :: str
01626
01627     str%numRec = -1
01628     str%loaded = .false.
01629
01630     !---- Input parameters
01631     IF (ALLOCATED(str%basin))              DEALLOCATE(str%basin)
01632
01633     IF (ALLOCATED(str%dtg))                 DEALLOCATE(str%dtg)
01634     IF (ALLOCATED(str%stormNumber))         DEALLOCATE(str%stormNumber)
01635     IF (ALLOCATED(str%year))                DEALLOCATE(str%year)
01636     IF (ALLOCATED(str%month))               DEALLOCATE(str%month)
01637     IF (ALLOCATED(str%day))                 DEALLOCATE(str%day)
01638     IF (ALLOCATED(str%hour))                DEALLOCATE(str%hour)
01639
01640     IF (ALLOCATED(str%castTime))             DEALLOCATE(str%castTime)
01641     IF (ALLOCATED(str%castType))             DEALLOCATE(str%castType)
01642     IF (ALLOCATED(str%fcstInc))              DEALLOCATE(str%fcstInc)
01643
01644     IF (ALLOCATED(str%iLat))                 DEALLOCATE(str%iLat)
01645     IF (ALLOCATED(str%lat))                  DEALLOCATE(str%lat)
01646     IF (ALLOCATED(str%iLon))                 DEALLOCATE(str%iLon)
01647     IF (ALLOCATED(str%lon))                  DEALLOCATE(str%lon)
01648
01649     IF (ALLOCATED(str%iSpeed))               DEALLOCATE(str%iSpeed)
01650     IF (ALLOCATED(str%speed))                DEALLOCATE(str%speed)
01651
01652     IF (ALLOCATED(str%iCPress))              DEALLOCATE(str%iCPress)
01653     IF (ALLOCATED(str%cPress))               DEALLOCATE(str%cPress)
01654
01655     IF (ALLOCATED(str%iRrp))                 DEALLOCATE(str%iRrp)
01656     IF (ALLOCATED(str%rrp))                  DEALLOCATE(str%rrp)
01657
01658     IF (ALLOCATED(str%iRmw))                 DEALLOCATE(str%iRmw)
01659     IF (ALLOCATED(str%rmw))                  DEALLOCATE(str%rmw)
01660
01661     IF (ALLOCATED(str%cPrDt))                DEALLOCATE(str%cPrDt)
01662
01663     IF (ALLOCATED(str%trVx))                 DEALLOCATE(str%trVx)
01664     IF (ALLOCATED(str%trVy))                 DEALLOCATE(str%trVy)
01665
01666     END SUBROUTINE deallochollstruct
01667
01668 !=====
01669
01670 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.](#)

```

00001 !-----
00002 !           M O D U L E   S I Z E S
00003 !-----
00013 !-----
00014
00015 MODULE pahm_sizes
00016
00017     IMPLICIT NONE
00018
00019     !-----
00020     ! I N T E R F A C E S
00021     !-----
00022     INTERFACE comparereals
00023         MODULE PROCEDURE comparesinglereals
00024         MODULE PROCEDURE comparedoublereals
00025     END INTERFACE comparereals
00026
00027     INTERFACE fixnearwholereal
00028         MODULE PROCEDURE fixnearwholesinglereal
00029         MODULE PROCEDURE fixnearwholedoublereal
00030     END INTERFACE fixnearwholereal
00031     !-----
00032
00033     ! SP = single precision, HP = high (double) precision
00034     INTEGER, PARAMETER :: sp = selected_real_kind(6, 37) ! 6 digits, range \([10^{-37} , 10^{+37}) -
1)\), 32 bits
00035     INTEGER, PARAMETER :: hp = selected_real_kind(15, 307) ! 15 digits, range \([10^{-307} , 10^{+307}) -
1)\), 64 bits
00036
00037     ! Precision of integers:
00038     INTEGER, PARAMETER :: int16 = selected_int_kind(38) ! Range \([-2^{127},+2^{127} - 1]\), 39 digits
plus sign; 128 bits
00039     INTEGER, PARAMETER :: int8 = selected_int_kind(18) ! Range \([-2^{63},+2^{63} - 1]\), 19 digits
plus sign; 64 bits
00040     INTEGER, PARAMETER :: int4 = selected_int_kind( 9) ! Range \([-2^{31},+2^{31} - 1]\), 10 digits
plus sign; 32 bits
00041     INTEGER, PARAMETER :: int2 = selected_int_kind( 4) ! Range \([-2^{15},+2^{15} - 1]\), 5 digits
plus sign; 16 bits
00042     INTEGER, PARAMETER :: int1 = selected_int_kind( 2) ! Range \([-2^{7},+2^{7} - 1]\), 3 digits
plus sign; 8 bits
00043     INTEGER, PARAMETER :: long = int8
00044     INTEGER, PARAMETER :: llong = int16
00045
00046     INTEGER,PARAMETER :: wp = hp ! default real kind (for csv_module)
00047     INTEGER,PARAMETER :: ip = int8 ! default integer kind (for csv_module)
00048
00049     ! By default we perform all calculations in double precision
00050     ! SET NUMBER OF BYTES "SZ" IN REAL(SZ) DECLARATIONS
00051     ! SET "NBYTE" FOR PROCESSING INPUT DATA RECORD LENGTH
00052 #ifdef REAL4
00053     INTEGER, PARAMETER :: sz = sp
00054     INTEGER, PARAMETER :: nbyte = 4
00055 #else
00056     INTEGER, PARAMETER :: sz = hp
00057     INTEGER, PARAMETER :: nbyte = 8
00058 #endif
00059
00060     ! Used to initialize the mesh arrays and in NetCDF output files for missing values.
00061     ! Also used to initialize some input variables to check if these variables
00062     ! were supplied user defined values.
00063     REAL(sz), PARAMETER :: rmissv = -999999.0_sz
00064     INTEGER, PARAMETER :: imissv = -999999
00065
00066     CHARACTER(LEN=1), PARAMETER :: blank = ' '
00067
00068     ! Filename length (considers the presence of the full path in the filename)
00069     INTEGER, PARAMETER :: fnamelen = 1024
00070
00071     CONTAINS
00072
00073
00074
00075     !-----
00076     ! F U N C T I O N   CompareDoubleReals
00077     !-----
00100     !-----

```

```

00101  INTEGER FUNCTION comparedoublereals(rVal1, rVal2, eps) RESULT(myValOut)
00102
00103      IMPLICIT NONE
00104
00105      ! Global variables
00106      REAL(hp), INTENT(IN)           :: rval1, rval2
00107      REAL(hp), OPTIONAL, INTENT(IN) :: eps
00108
00109      ! Local variables
00110      REAL(hp)                       :: epssys, epsusr, value
00111
00112
00113      epssys = 2.0_hp * epsilon(rval1)
00114
00115      IF (PRESENT(eps)) THEN
00116          epsusr = abs(eps)
00117      ELSE
00118          epsusr = epssys
00119      ENDIF
00120
00121      IF ((abs(rval1) < 1.0_hp) .OR. (abs(rval2) < 1.0_hp)) THEN
00122          value = rval1 - rval2
00123      ELSE
00124          value = (rval1 - rval2) / max(rval1, rval2)
00125          IF (abs(value) < 1.0_hp) value = rval1 - rval2
00126      END IF
00127
00128      IF (abs(value) < epsusr) THEN
00129          myvalout = 0
00130      ELSE IF (rval1 < rval2) THEN
00131          myvalout = -1
00132      ELSE
00133          myvalout = 1
00134      END IF
00135
00136      RETURN
00137
00138  END FUNCTION comparedoublereals
00139
00140  !=====
00141
00142  !-----
00143  ! FUNCTION COMPARE SINGLE REALS
00144  !-----
00145  !-----
00146
00168  INTEGER FUNCTION comparesinglereals(rVal1, rVal2, eps) RESULT(myValOut)
00169
00170      IMPLICIT NONE
00171
00172      ! Global variables
00173      REAL(sp), INTENT(IN)           :: rval1, rval2
00174      REAL(sp), OPTIONAL, INTENT(IN) :: eps
00175
00176      ! Local variables
00177      REAL(sp)                       :: epssys, epsusr, value
00178
00179
00180      epssys = 2.0_sp * epsilon(rval1)
00181
00182      IF (PRESENT(eps)) THEN
00183          epsusr = abs(eps)
00184      ELSE
00185          epsusr = epssys
00186      ENDIF
00187
00188      IF ((abs(rval1) < 1.0_sp) .OR. (abs(rval2) < 1.0_sp)) THEN
00189          value = rval1 - rval2
00190      ELSE
00191          value = (rval1 - rval2) / max(rval1, rval2)
00192          IF (abs(value) < 1.0_sp) value = rval1 - rval2
00193      END IF
00194
00195      IF (abs(value) < epsusr) THEN
00196          myvalout = 0
00197      ELSE IF (rval1 < rval2) THEN
00198          myvalout = -1
00199      ELSE
00200          myvalout = 1
00201      END IF
00202
00203      RETURN

```

```

00204
00205   END FUNCTION comparesinglereals
00206
00207 !=====
00208
00209 !-----
00210 ! FUNCTION FIX NEAR WHOLE DOUBLE REAL
00211 !-----
00234 !-----
00235 REAL(hp) function fixnearwholedoublereal(rval, eps) result(myvalout)
00236
00237   IMPLICIT NONE
00238
00239   ! Global Variables
00240   REAL(hp), INTENT(IN)           :: rval
00241   REAL(hp), OPTIONAL, INTENT(IN) :: eps
00242
00243   ! Local Variables
00244   REAL(hp)                       :: epssys, epsusr, value
00245
00246
00247   epssys = 2.0_hp * epsilon(rval)
00248
00249   IF (PRESENT(eps)) THEN
00250     epsusr = abs(eps)
00251   ELSE
00252     epsusr = epssys
00253   ENDIF
00254
00255   myvalout = rval
00256   value = anint(myvalout)
00257   IF (compareereals(myvalout, value, epsusr) == 0) myvalout = value
00258
00259   RETURN
00260
00261   END FUNCTION fixnearwholedoublereal
00262
00263 !=====
00264
00265 !-----
00266 ! FUNCTION FIX NEAR WHOLE SINGLE REAL
00267 !-----
00290 !-----
00291 REAL(sp) function fixnearwholesinglereal(rval, eps) result(myvalout)
00292
00293   IMPLICIT NONE
00294
00295   ! Global Variables
00296   REAL(sp), INTENT(IN)           :: rval
00297   REAL(sp), OPTIONAL, INTENT(IN) :: eps
00298
00299   ! Local Variables
00300   REAL(sp)                       :: epssys, epsusr, value
00301
00302
00303   epssys = 2.0_sp * epsilon(rval)
00304
00305   IF (PRESENT(eps)) THEN
00306     epsusr = abs(eps)
00307   ELSE
00308     epsusr = epssys
00309   ENDIF
00310
00311   myvalout = rval
00312   value = anint(myvalout)
00313   IF (compareereals(myvalout, value, epsusr) == 0) myvalout = value
00314
00315   RETURN
00316
00317   END FUNCTION fixnearwholesinglereal
00318
00319 !=====
00320
00321 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](#).

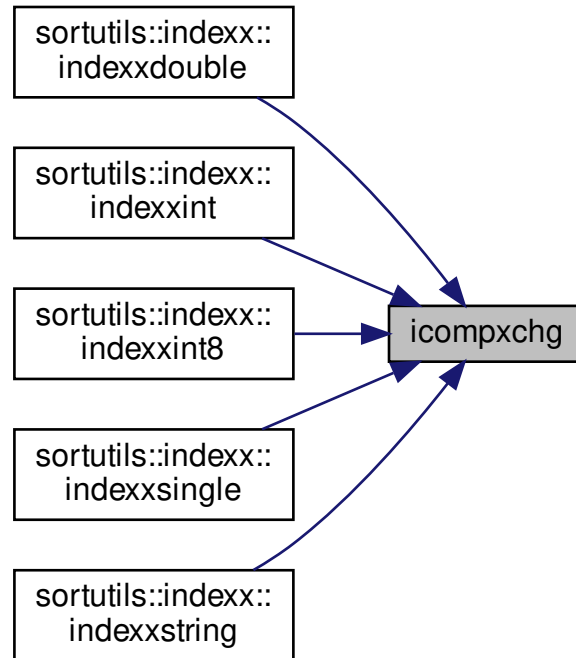
9.34.2 Function/Subroutine Documentation

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.](#)

```

00001 !-----
00002 !           M O D U L E   U T I L I T I E S
00003 !-----
00014 !-----
00015
00016 MODULE sortutils
00017
00018   USE pahm_sizes
00019   USE pahm_messages
00020
00021 !-----
00022 ! I N T E R F A C E S
00023 !-----
00024 INTERFACE indexx
00025   MODULE PROCEDURE indexxint
00026   MODULE PROCEDURE indexxint8
00027   MODULE PROCEDURE indexxstring
00028   MODULE PROCEDURE indexxsingle
00029   MODULE PROCEDURE indexxdouble
00030 END INTERFACE indexx
00031
00032 INTERFACE arth
00033   MODULE PROCEDURE arthint
00034   MODULE PROCEDURE arthsingle
00035   MODULE PROCEDURE arthdouble
00036 END INTERFACE arth
00037

```

```

00038  INTERFACE arraycopy
00039      MODULE PROCEDURE arraycopyint
00040      MODULE PROCEDURE arraycopysingle
00041      MODULE PROCEDURE arraycopydouble
00042  END INTERFACE arraycopy
00043
00044  INTERFACE arrayequal
00045      MODULE PROCEDURE arrayequalint
00046      MODULE PROCEDURE arrayequalsingle
00047      MODULE PROCEDURE arrayequaldouble
00048  END INTERFACE arrayequal
00049
00050  INTERFACE swap
00051      MODULE PROCEDURE swapint
00052      MODULE PROCEDURE swapsingle
00053      MODULE PROCEDURE swapdouble
00054      MODULE PROCEDURE swapintvec
00055      MODULE PROCEDURE swapsinglevec
00056      MODULE PROCEDURE swapdoublevec
00057  END INTERFACE swap
00058  !-----
00059
00060
00061  CONTAINS
00062
00063
00064  !-----
00065  ! SUBROUTINE INDEXXINT
00066  !-----
00084  !-----
00085  SUBROUTINE indexxint(arr1D, idx1D, status)
00086
00087      IMPLICIT NONE
00088
00089      ! Global variables
00090      INTEGER, DIMENSION(:), INTENT(IN) :: arr1D
00091      INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00092      INTEGER, OPTIONAL, INTENT(OUT) :: status
00093
00094      ! Local variables
00095      INTEGER, PARAMETER :: NN = 15, nstack = 50
00096      INTEGER :: a
00097      INTEGER :: nARR, nIDX, tmpIDX
00098      INTEGER :: k, i, j, l, r
00099      INTEGER :: ist, stack(NSTACK)
00100      CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00101
00102
00103      CALL setmessagesource("IndexxInt")
00104
00105      IF (PRESENT(status)) status = 0
00106
00107      narr = SIZE(arr1D, 1)
00108      nidx = SIZE(idx1D, 1)
00109
00110      IF (narr /= nidx) THEN
00111          WRITE(tmpStr1, '(a, i0)') 'nARR = ', narr
00112          WRITE(tmpStr2, '(a, i0)') 'nIDX = ', nidx
00113          WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00114              trim(adjustl(tmpStr1)) // ', ' // trim(adjustl(tmpStr2))
00115
00116          CALL allmessage(error, scratchmessage)
00117          CALL unsetmessagesource()
00118
00119          IF (PRESENT(status)) status = 1
00120
00121          RETURN
00122      END IF
00123
00124      idx1D = arth(1, 1, narr)
00125
00126      ist = 0
00127      l = 1
00128      r = narr
00129
00130      DO
00131          IF (r - l < nn) THEN
00132              DO j = 1 + l, r
00133                  tmpidx = idx1D(j)
00134                  a = arr1D(tmpidx)
00135                  DO i = j - 1, l, -1

```



```

00136         IF (arrld(idxld(i)) <= a) EXIT
00137         idxld(i + 1) = idxld(i)
00138     END DO
00139     idxld(i + 1) = tmpidx
00140 END DO
00141
00142 IF (ist == 0) THEN
00143     CALL unsetmessagesource()
00144
00145     RETURN
00146 END IF
00147
00148 r = stack(ist)
00149 l = stack(ist - 1)
00150 ist = ist - 2
00151 ELSE
00152     k = (l + r) / 2
00153
00154     CALL swap(idxld(k), idxld(l + 1))
00155     CALL icmpxchg(idxld(l), idxld(r))
00156     CALL icmpxchg(idxld(l + 1), idxld(r))
00157     CALL icmpxchg(idxld(l), idxld(l + 1))
00158
00159     i = l + 1
00160     j = r
00161     tmpidx = idxld(l + 1)
00162     a = arrld(tmpidx)
00163
00164     DO
00165         DO
00166             i = i + 1
00167             IF (arrld(idxld(i)) > a) EXIT
00168         END DO
00169
00170         DO
00171             j = j - 1
00172             IF (arrld(idxld(j)) < a) EXIT
00173         END DO
00174
00175         IF (j < i) EXIT
00176         CALL swap(idxld(i), idxld(j))
00177     END DO
00178
00179     idxld(l + 1) = idxld(j)
00180     idxld(j) = tmpidx
00181     ist = ist + 2
00182
00183 IF (ist > nstack) THEN
00184     WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00185     WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00186                                     trim(adjustl(tmpstr1))
00187
00188     CALL logmessage(error, scratchmessage)
00189     CALL unsetmessagesource()
00190
00191     IF (PRESENT(status)) status = 2
00192
00193     RETURN
00194
00195 END IF
00196
00197 IF (r - i + 1 >= j - 1) THEN
00198     stack(ist) = r
00199     stack(ist - 1) = i
00200     r = j - 1
00201 ELSE
00202     stack(ist) = j - 1
00203     stack(ist - 1) = l
00204     l = i
00205 END IF
00206 END IF
00207 END DO
00208
00209 CALL unsetmessagesource()
00210
00211 CONTAINS
00212
00213 SUBROUTINE icmpxchg(i, j)
00214
00215 IMPLICIT NONE
00216

```

```

00217
00218      ! Global variables
00219      INTEGER, INTENT(INOUT) :: i, j
00220
00221      ! Local variables
00222      INTEGER :: swp
00223
00224      IF (arr1d(j) < arr1d(i)) THEN
00225          swp = i
00226          i = j
00227          j = swp
00228      END IF
00229
00230      END SUBROUTINE icompchg
00231
00232  END SUBROUTINE indexxint
00233
00234      !=====
00235
00236      !-----
00237      ! SUBROUTINE INDEXXINT 8
00238      !-----
00239      !=====
00240
00241  SUBROUTINE indexxint8(arr1D, idx1D, status)
00242
00243      IMPLICIT NONE
00244
00245      ! Global variables
00246      INTEGER(INT8), DIMENSION(:), INTENT(IN) :: arr1D
00247      INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00248      INTEGER, OPTIONAL, INTENT(OUT) :: status
00249
00250      ! Local variables
00251      INTEGER, PARAMETER :: NN = 15, nstack = 50
00252      INTEGER(INT8) :: a
00253      INTEGER :: nARR, nIDX, tmpIDX
00254      INTEGER :: k, i, j, l, r
00255      INTEGER :: ist, stack(NSTACK)
00256      CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00257
00258      CALL setmessagesource("IndexxInt8")
00259
00260      IF (PRESENT(status)) status = 0
00261
00262      narr = SIZE(arr1d, 1)
00263      nidx = SIZE(idx1d, 1)
00264
00265      IF (narr /= nidx) THEN
00266          WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00267          WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00268          WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00269              trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00270
00271          CALL allmessage(error, scratchmessage)
00272          CALL unsetmessagesource()
00273
00274      IF (PRESENT(status)) status = 1
00275
00276      RETURN
00277  END IF
00278
00279      idx1d = arth(1, 1, narr)
00280
00281      ist = 0
00282      l = 1
00283      r = narr
00284
00285      DO
00286          IF (r - l < nn) THEN
00287              DO j = l + 1, r
00288                  tmpidx = idx1d(j)
00289                  a = arr1d(tmpidx)
00290                  DO i = j - 1, l, -1
00291                      IF (arr1d(idx1d(i)) <= a) EXIT
00292                      idx1d(i + 1) = idx1d(i)
00293                  END DO
00294                  idx1d(i + 1) = tmpidx
00295              END DO
00296          IF (ist == 0) THEN

```

```

00315         CALL unsetmessagesource()
00316
00317         RETURN
00318     END IF
00319
00320     r = stack(ist)
00321     l = stack(ist - 1)
00322     ist = ist - 2
00323 ELSE
00324     k = (l + r) / 2
00325
00326     CALL swap(idx1d(k), idx1d(l + 1))
00327     CALL icompchg(idx1d(l), idx1d(r))
00328     CALL icompchg(idx1d(l + 1), idx1d(r))
00329     CALL icompchg(idx1d(l), idx1d(l + 1))
00330
00331     i = l + 1
00332     j = r
00333     tmpidx = idx1d(l + 1)
00334     a = arr1d(tmpidx)
00335
00336     DO
00337         DO
00338             i = i + 1
00339             IF (arr1d(idx1d(i)) > a) EXIT
00340         END DO
00341
00342         DO
00343             j = j - 1
00344             IF (arr1d(idx1d(j)) < a) EXIT
00345         END DO
00346
00347         IF (j < i) EXIT
00348         CALL swap(idx1d(i), idx1d(j))
00349     END DO
00350
00351     idx1d(l + 1) = idx1d(j)
00352     idx1d(j) = tmpidx
00353     ist = ist + 2
00354
00355     IF (ist > nstack) THEN
00356         WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00357         WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00358             trim(adjustl(tmpstr1))
00359
00360         CALL logmessage(error, scratchmessage)
00361         CALL unsetmessagesource()
00362
00363         IF (PRESENT(status)) status = 2
00364
00365         RETURN
00366
00367     END IF
00368
00369     IF (r - i + 1 >= j - 1) THEN
00370         stack(ist) = r
00371         stack(ist - 1) = i
00372         r = j - 1
00373     ELSE
00374         stack(ist) = j - 1
00375         stack(ist - 1) = l
00376         l = i
00377     END IF
00378 END IF
00379 END DO
00380
00381 CALL unsetmessagesource()
00382
00383 CONTAINS
00384
00385 SUBROUTINE icompchg(i, j)
00386
00387     IMPLICIT NONE
00388
00389     ! Global variables
00390     INTEGER, INTENT(INOUT) :: i, j
00391
00392     ! Local variables
00393     INTEGER :: swp
00394
00395

```

```

00396         IF (arrld(j) < arrld(i)) THEN
00397             swp = i
00398             i = j
00399             j = swp
00400         END IF
00401
00402     END SUBROUTINE icompchg
00403
00404 END SUBROUTINE indexxint8
00405
00406 !=====
00407
00408 !-----
00409 ! SUBROUTINE INDEXX STRING
00410 !-----
00429 !-----
00430 SUBROUTINE indexxstring(arrld, idxld, status, caseSens)
00431
00432     IMPLICIT NONE
00433
00434     ! Global variables
00435     CHARACTER(LEN=*) , DIMENSION(:), INTENT(IN) :: arrld
00436     LOGICAL, OPTIONAL, INTENT(IN) :: caseSens
00437     INTEGER, DIMENSION(:), INTENT(OUT) :: idxld
00438     INTEGER, OPTIONAL, INTENT(OUT) :: status
00439
00440     ! Local variables
00441     INTEGER, PARAMETER :: NN = 15, nstack = 50
00442     CHARACTER(LEN=LEN(arrld(1))) :: a
00443     INTEGER :: nARR, nIDX, tmpIDX
00444     INTEGER :: k, i, j, l, r
00445     INTEGER :: ist, stack(NSTACK)
00446     CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00447     LOGICAL :: sFlag
00448
00449
00450     CALL setmessagesource("IndexxString")
00451
00452     sflag = .true.
00453     IF (PRESENT(caseSens)) sflag = caseSens
00454
00455     IF (PRESENT(status)) status = 0
00456
00457     narr = SIZE(arrld, 1)
00458     nidx = SIZE(idxld, 1)
00459
00460     IF (narr /= nidx) THEN
00461         WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00462         WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00463         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arrld and idxld is not the same: ' // &
00464             trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00465
00466         CALL allmessage(error, scratchmessage)
00467         CALL unsetmessagesource()
00468
00469         IF (PRESENT(status)) status = 1
00470
00471         RETURN
00472     END IF
00473
00474     idxld = arth(1, 1, narr)
00475
00476     ist = 0
00477     l = 1
00478     r = narr
00479
00480     DO
00481         IF (r - l < nn) THEN
00482             DO j = l + 1, r
00483                 tmpidx = idxld(j)
00484                 a = arrld(tmpidx)
00485                 DO i = j - 1, l, -1
00486                     IF (stringlexcomp(arrld(idxld(i)), a, sflag) <= 0) EXIT
00487                     idxld(i + 1) = idxld(i)
00488                 END DO
00489                 idxld(i + 1) = tmpidx
00490             END DO
00491
00492             IF (ist == 0) THEN
00493                 CALL unsetmessagesource()
00494             
```

```

00495         RETURN
00496     END IF
00497
00498     r = stack(ist)
00499     l = stack(ist - 1)
00500     ist = ist - 2
00501 ELSE
00502     k = (l + r) / 2
00503
00504     CALL swap(idx1d(k), idx1d(l + 1))
00505     CALL icompchg(idx1d(l), idx1d(r))
00506     CALL icompchg(idx1d(l + 1), idx1d(r))
00507     CALL icompchg(idx1d(l), idx1d(l + 1))
00508
00509     i = l + 1
00510     j = r
00511     tmpidx = idx1d(l + 1)
00512     a = arr1d(tmpidx)
00513
00514     DO
00515         DO
00516             i = i + 1
00517             IF (stringlexcomp(arr1d(idx1d(i)), a, sflag) > 0) EXIT
00518         END DO
00519
00520         DO
00521             j = j - 1
00522             IF (stringlexcomp(arr1d(idx1d(j)), a, sflag) < 0) EXIT
00523         END DO
00524
00525         IF (j < i) EXIT
00526         CALL swap(idx1d(i), idx1d(j))
00527     END DO
00528
00529     idx1d(l + 1) = idx1d(j)
00530     idx1d(j) = tmpidx
00531     ist = ist + 2
00532
00533     IF (ist > nstack) THEN
00534         WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00535         WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00536                                     trim(adjustl(tmpstr1))
00537
00538         CALL logmessage(error, scratchmessage)
00539         CALL unsetmessagesource()
00540
00541         IF (PRESENT(status)) status = 2
00542
00543         RETURN
00544     END IF
00545
00546     IF (r - i + 1 >= j - 1) THEN
00547         stack(ist) = r
00548         stack(ist - 1) = i
00549         r = j - 1
00550     ELSE
00551         stack(ist) = j - 1
00552         stack(ist - 1) = l
00553         l = i
00554     END IF
00555 END IF
00556 END DO
00557
00558 CALL unsetmessagesource()
00559
00560 CONTAINS
00561
00562 SUBROUTINE icompchg(i, j)
00563
00564     IMPLICIT NONE
00565
00566     ! Global variables
00567     INTEGER, INTENT(INOUT) :: i, j
00568
00569     ! Local variables
00570     INTEGER :: swp
00571
00572     IF (stringlexcomp(arr1d(j), arr1d(i), sflag) < 0) THEN
00573         swp = i

```

```

00576         i   = j
00577         j   = swp
00578     END IF
00579
00580     END SUBROUTINE icompchg
00581
00582 END SUBROUTINE indexxstring
00583
00584 !=====
00585 !-----
00586 ! SUBROUTINE INDEXX SINGLE
00587 !-----
00588 !-----
00606 !-----
00607 SUBROUTINE indexxsingle(arr1D, idx1D, status)
00608
00609     IMPLICIT NONE
00610
00611     ! Global variables
00612     REAL(SP), DIMENSION(:), INTENT(IN) :: arr1D
00613     INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00614     INTEGER, OPTIONAL, INTENT(OUT)    :: status
00615
00616     ! Local variables
00617     INTEGER, PARAMETER                :: NN = 15, nstack = 50
00618     REAL(SP)                          :: a
00619     INTEGER                           :: nARR, nIDX, tmpIDX
00620     INTEGER                           :: k, i, j, l, r
00621     INTEGER                           :: ist, stack(NSTACK)
00622     CHARACTER(LEN=64)                 :: tmpStr1, tmpStr2
00623
00624
00625     CALL setmessagesource("IndexxSingle")
00626
00627     IF (PRESENT(status)) status = 0
00628
00629     narr = SIZE(arr1D, 1)
00630     nidx = SIZE(idx1D, 1)
00631
00632     IF (narr /= nidx) THEN
00633         WRITE(tmpStr1, '(a, i0)') 'nARR = ', narr
00634         WRITE(tmpStr2, '(a, i0)') 'nIDX = ', nidx
00635         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00636             trim(adjustl(tmpStr1)) // ', ' // trim(adjustl(tmpStr2))
00637
00638         CALL logmessage(error, scratchmessage)
00639         CALL unsetmessagesource()
00640
00641         IF (PRESENT(status)) status = 1
00642
00643         RETURN
00644     END IF
00645
00646     idx1D = arth(1, 1, narr)
00647
00648     ist = 0
00649     l   = 1
00650     r   = narr
00651
00652     DO
00653         IF (r - l < nn) THEN
00654             DO j = l + 1, r
00655                 tmpidx = idx1D(j)
00656                 a = arr1D(tmpidx)
00657                 DO i = j - 1, l, -1
00658                     IF (arr1D(idx1D(i)) <= a) EXIT
00659                     idx1D(i + 1) = idx1D(i)
00660                 END DO
00661                 idx1D(i + 1) = tmpidx
00662             END DO
00663
00664             IF (ist == 0) THEN
00665                 CALL unsetmessagesource()
00666
00667                 RETURN
00668             END IF
00669
00670             r = stack(ist)
00671             l = stack(ist - 1)
00672             ist = ist - 2
00673         ELSE

```

```

00674         k = (l + r) / 2
00675
00676         CALL swap(idx1d(k), idx1d(l + 1))
00677         CALL icompchg(idx1d(l), idx1d(r))
00678         CALL icompchg(idx1d(l + 1), idx1d(r))
00679         CALL icompchg(idx1d(l), idx1d(l + 1))
00680
00681         i = l + 1
00682         j = r
00683         tmpidx = idx1d(l + 1)
00684         a = arr1d(tmpidx)
00685
00686         DO
00687             DO
00688                 i = i + 1
00689                 IF (arr1d(idx1d(i)) > a) EXIT
00690             END DO
00691
00692             DO
00693                 j = j - 1
00694                 IF (arr1d(idx1d(j)) < a) EXIT
00695             END DO
00696
00697             IF (j < i) EXIT
00698             CALL swap(idx1d(i), idx1d(j))
00699         END DO
00700
00701         idx1d(l + 1) = idx1d(j)
00702         idx1d(j) = tmpidx
00703         ist = ist + 2
00704
00705         IF (ist > nstack) THEN
00706             WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00707             WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00708                                     trim(adjustl(tmpstr1))
00709
00710             CALL logmessage(error, scratchmessage)
00711             CALL unsetmessagesource()
00712
00713             IF (PRESENT(status)) status = 2
00714
00715             RETURN
00716
00717         END IF
00718
00719         IF (r - i + 1 >= j - 1) THEN
00720             stack(ist) = r
00721             stack(ist - 1) = i
00722             r = j - 1
00723         ELSE
00724             stack(ist) = j - 1
00725             stack(ist - 1) = l
00726             l = i
00727         END IF
00728     END IF
00729 END DO
00730
00731 CALL unsetmessagesource()
00732
00733 CONTAINS
00734
00735 SUBROUTINE icompchg(i, j)
00736
00737     IMPLICIT NONE
00738
00739     ! Global variables
00740     INTEGER, INTENT(INOUT) :: i, j
00741
00742     ! Local variables
00743     INTEGER :: swp
00744
00745     IF (arr1d(j) < arr1d(i)) THEN
00746         swp = i
00747         i = j
00748         j = swp
00749     END IF
00750
00751 END SUBROUTINE icompchg
00752
00753 END SUBROUTINE indexssingle
00754

```

```

00755
00756 !=====
00757
00758 !-----
00759 ! SUBROUTINE INDEXX DOUBLE
00760 !-----
00778 !-----
00779 SUBROUTINE indexxdouble(arr1D, idx1D, status)
00780
00781   IMPLICIT NONE
00782
00783   ! Global variables
00784   REAL(HP), DIMENSION(:), INTENT(IN) :: arr1D
00785   INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00786   INTEGER, OPTIONAL, INTENT(OUT)    :: status
00787
00788   ! Local variables
00789   INTEGER, PARAMETER                :: NN = 15, nstack = 50
00790   REAL(HP)                          :: a
00791   INTEGER                           :: nARR, nIDX, tmpIDX
00792   INTEGER                           :: k, i, j, l, r
00793   INTEGER                           :: ist, stack(NSTACK)
00794   CHARACTER(LEN=64)                 :: tmpStr1, tmpStr2
00795
00796
00797   CALL setmessagesource("IndexxDouble")
00798
00799   IF (PRESENT(status)) status = 0
00800
00801   narr = SIZE(arr1D, 1)
00802   nidx = SIZE(idx1D, 1)
00803
00804   IF (narr /= nidx) THEN
00805     WRITE(tmpStr1, '(a, i0)') 'nARR = ', narr
00806     WRITE(tmpStr2, '(a, i0)') 'nIDX = ', nidx
00807     WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00808       trim(adjustl(tmpStr1)) // ', ' // trim(adjustl(tmpStr2))
00809
00810     CALL logmessage(error, scratchmessage)
00811     CALL unsetmessagesource()
00812
00813     IF (PRESENT(status)) status = 1
00814
00815     RETURN
00816   END IF
00817
00818   idx1D = arth(1, 1, narr)
00819
00820   ist = 0
00821   l = 1
00822   r = narr
00823
00824   DO
00825     IF (r - l < nn) THEN
00826       DO j = 1 + 1, r
00827         tmpidx = idx1D(j)
00828         a = arr1D(tmpidx)
00829         DO i = j - 1, l, -1
00830           IF (arr1D(idx1D(i)) <= a) EXIT
00831           idx1D(i + 1) = idx1D(i)
00832         END DO
00833         idx1D(i + 1) = tmpidx
00834       END DO
00835
00836       IF (ist == 0) THEN
00837         CALL unsetmessagesource()
00838
00839         RETURN
00840       END IF
00841
00842       r = stack(ist)
00843       l = stack(ist - 1)
00844       ist = ist - 2
00845     ELSE
00846       k = (l + r) / 2
00847
00848       CALL swap(idx1D(k), idx1D(l + 1))
00849       CALL icompchg(idx1D(l), idx1D(r))
00850       CALL icompchg(idx1D(l + 1), idx1D(r))
00851       CALL icompchg(idx1D(l), idx1D(l + 1))
00852

```



```

00853      i = l + 1
00854      j = r
00855      tmpidx = idxld(l + 1)
00856      a = arrld(tmpidx)
00857
00858      DO
00859          DO
00860              i = i + 1
00861              IF (arrld(idxld(i)) > a) EXIT
00862          END DO
00863
00864          DO
00865              j = j - 1
00866              IF (arrld(idxld(j)) < a) EXIT
00867          END DO
00868
00869          IF (j < i) EXIT
00870          CALL swap(idxld(i), idxld(j))
00871      END DO
00872
00873      idxld(l + 1) = idxld(j)
00874      idxld(j) = tmpidx
00875      ist = ist + 2
00876
00877      IF (ist > nstack) THEN
00878          WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00879          WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00880              trim(adjustl(tmpstr1))
00881
00882          CALL logmessage(error, scratchmessage)
00883          CALL unsetmessagesource()
00884
00885          IF (PRESENT(status)) status = 2
00886
00887          RETURN
00888
00889      END IF
00890
00891      IF (r - i + 1 >= j - 1) THEN
00892          stack(ist) = r
00893          stack(ist - 1) = i
00894          r = j - 1
00895      ELSE
00896          stack(ist) = j - 1
00897          stack(ist - 1) = l
00898          l = i
00899      END IF
00900  END IF
00901  END DO
00902
00903  CALL unsetmessagesource()
00904
00905  CONTAINS
00906
00907  SUBROUTINE icompchg(i, j)
00908
00909      IMPLICIT NONE
00910
00911      ! Global variables
00912      INTEGER, INTENT(INOUT) :: i, j
00913
00914      ! Local variables
00915      INTEGER :: swp
00916
00917      IF (arrld(j) < arrld(i)) THEN
00918          swp = i
00919          i = j
00920          j = swp
00921      END IF
00922
00923  END SUBROUTINE icompchg
00924
00925  END SUBROUTINE indexdouble
00926
00927  !=====
00928  !-----
00929  ! SUBROUTINE QUICK SORT
00930  !-----
00931  !=====
00932
00950

```

```

00951 SUBROUTINE quicksort(arrld, status)
00952
00953     IMPLICIT NONE
00954
00955     ! Global variables
00956     REAL(SZ), DIMENSION(:), INTENT(INOUT) :: arrld
00957     INTEGER, OPTIONAL, INTENT(OUT)       :: status
00958
00959     ! Local variables
00960     INTEGER, PARAMETER                  :: NN = 15, nstack = 50
00961     REAL(SZ)                           :: a
00962     INTEGER                             :: nARR
00963     INTEGER                             :: k, i, j, l, r
00964     INTEGER                             :: ist, stack(NSTACK)
00965     CHARACTER(LEN=64)                   :: tmpStr1
00966
00967
00968     CALL setmessagesource("QuickSort")
00969
00970     IF (PRESENT(status)) status = 0
00971
00972     narr = size(arrld, 1)
00973
00974     ist = 0
00975     l   = 1
00976     r   = narr
00977
00978     DO
00979         ! Insertion sort when subarray small enough
00980         IF (r - l < nn) THEN
00981             DO j = l + 1, r
00982                 a = arrld(j)
00983                 DO i = j - 1, l, -1
00984                     IF (arrld(i) <= a) EXIT
00985                     arrld(i + 1) = arrld(i)
00986                 END DO
00987                 arrld(i + 1) = a
00988             END DO
00989
00990             IF (ist == 0) THEN
00991                 CALL unsetmessagesource()
00992
00993                 RETURN
00994             END IF
00995
00996             ! Pop stack and begin a new round of partitioning
00997             r = stack(ist)
00998             l = stack(ist - 1)
00999             ist = ist - 2
01000
01001             ! Choose median of left, center, and right elements as partitioning
01002             ! element a. Also rearrange so that a(l) <= a(l + 1) <= a(r)
01003             ELSE
01004                 k = (l + r) / 2
01005
01006                 CALL swap(arrld(k), arrld(l + 1))
01007                 CALL swap(arrld(l), arrld(r), arrld(l) > arrld(r))
01008                 CALL swap(arrld(l + 1), arrld(r), arrld(l + 1) > arrld(r))
01009                 CALL swap(arrld(l), arrld(l + 1), arrld(l) > arrld(l + 1))
01010
01011                 ! Initialize pointers for partitioning
01012                 i = l + 1
01013                 j = r
01014                 a = arrld(l + 1) ! Partitioning element.
01015
01016                 DO ! Here is the meat.
01017                     ! Scan up to find element >= a
01018                     DO
01019                         i = i + 1
01020                         IF (arrld(i) > a) EXIT
01021                     END DO
01022
01023                     ! Scan down to find element <= a
01024                     DO
01025                         j = j - 1
01026                         IF (arrld(j) < a) EXIT
01027                     END DO
01028
01029                     ! Pointers crossed. Exit with partitioning complete.
01030                     IF (j < i) EXIT
01031

```

```

01032         CALL swap(arr1d(i), arr1d(j)) !Exchange elements.
01033     END DO
01034
01035     ! Insert partitioning element
01036     arr1d(l + 1) = arr1d(j)
01037     arr1d(j) = a
01038     ist = ist + 2
01039
01040     ! Push pointers to larger subarray on stack; process smaller subarray immediately.
01041     IF (ist > nstack) THEN
01042         WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
01043         WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
01044             trim(adjustl(tmpstr1))
01045
01046         CALL logmessage(error, scratchmessage)
01047         CALL unsetmessagesource()
01048
01049         IF (PRESENT(status)) status = 2
01050
01051         RETURN
01052     END IF
01053
01054     IF (r - i + 1 >= j - 1) THEN
01055         stack(ist) = r
01056         stack(ist - 1) = i
01057         r = j - 1
01058     ELSE
01059         stack(ist) = j - 1
01060         stack(ist - 1) = l
01061         l = i
01062     END IF
01063 END IF
01064 END DO
01065
01066 CALL unsetmessagesource()
01067
01068 END SUBROUTINE quicksort
01069
01070
01071 !=====
01072 !-----
01073 ! SUBROUTINE SORT 2
01074 !-----
01075 !-----
01076
01077 SUBROUTINE sort2(arr1D, slv1D, status)
01078
01079     IMPLICIT NONE
01080
01081     ! Global variables
01082     REAL(S2), DIMENSION(:), INTENT(INOUT) :: arr1D, slv1D
01083     INTEGER, OPTIONAL, INTENT(OUT) :: status
01084
01085     ! Local variables
01086     INTEGER :: nARR, nSLV
01087     INTEGER, DIMENSION(SIZE(arr1D)) :: idx1D
01088     CHARACTER(LEN=64) :: tmpStr1, tmpStr2
01089
01090     CALL setmessagesource("Sort2")
01091
01092     narr = SIZE(arr1D, 1)
01093     nslv = SIZE(slv1D, 1)
01094
01095     IF (narr /= nslv) THEN
01096         WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
01097         WRITE(tmpstr2, '(a, i0)') 'nSLV = ', nslv
01098         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and slv1D is not the same: ' // &
01099             trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
01100
01101         CALL logmessage(error, scratchmessage)
01102         CALL unsetmessagesource()
01103
01104         IF (PRESENT(status)) status = 1
01105
01106         RETURN
01107     END IF
01108
01109     ! Make the index array
01110     CALL indexx(arr1D, idx1D, status)
01111
01112

```

```

01134      ! Sort the array
01135      arrld = arrld(idxl)
01136
01137      ! Rearrange slave
01138      slvld = slvld(idxl)
01139
01140      CALL unsetmessagesource()
01141
01142  END SUBROUTINE sort2
01143
01144  !=====
01145
01146  !-----
01147  ! SUBROUTINE ARRAY COPY INT
01148  !-----
01149  !-----
01168  !-----
01169  SUBROUTINE arraycopyint(src, dest, nCP, nNCP)
01170
01171      IMPLICIT NONE
01172
01173      ! Global variables
01174      INTEGER, DIMENSION(:), INTENT(IN) :: src
01175      INTEGER, DIMENSION(:), INTENT(OUT) :: dest
01176      INTEGER, INTENT(OUT) :: nCP, nNCP
01177
01178      ncp = min(SIZE(src), SIZE(dest))
01179      nncp = SIZE(src) - ncp
01180      dest(1:ncp) = src(1:ncp)
01181
01182  END SUBROUTINE arraycopyint
01183
01184  !=====
01185
01186  !-----
01187  ! SUBROUTINE ARRAY COPY SINGLE
01188  !-----
01207  !-----
01208  SUBROUTINE arraycopysingle(src, dest, nCP, nNCP)
01209
01210      IMPLICIT NONE
01211
01212      ! Global variables
01213      REAL(SP), DIMENSION(:), INTENT(IN) :: src
01214      REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
01215      INTEGER, INTENT(OUT) :: nCP, nNCP
01216
01217      ncp = min(SIZE(src), SIZE(dest))
01218      nncp = SIZE(src) - ncp
01219      dest(1:ncp) = src(1:ncp)
01220
01221  END SUBROUTINE arraycopysingle
01222
01223  !=====
01224
01225  !-----
01226  ! SUBROUTINE ARRAY COPY DOUBLE
01227  !-----
01246  !-----
01247  SUBROUTINE arraycopydouble(src, dest, nCP, nNCP)
01248
01249      IMPLICIT NONE
01250
01251      ! Global variables
01252      REAL(HP), DIMENSION(:), INTENT(IN) :: src
01253      REAL(HP), DIMENSION(:), INTENT(OUT) :: dest
01254      INTEGER, INTENT(OUT) :: nCP, nNCP
01255
01256      ncp = min(SIZE(src), SIZE(dest))
01257      nncp = SIZE(src) - ncp
01258      dest(1:ncp) = src(1:ncp)
01259
01260  END SUBROUTINE arraycopydouble
01261
01262  !=====
01263
01264  !-----
01265  ! SUBROUTINE ARRAY EQUAL INT
01266  !-----
01283  !-----
01284  LOGICAL FUNCTION arrayequalint(arr1, arr2) RESULT(myValOut)

```

```

01285
01286     IMPLICIT NONE
01287
01288     ! Global variables
01289     INTEGER, DIMENSION(:), INTENT(IN) :: arr1, arr2
01290
01291
01292     IF (SIZE(arr1) /= SIZE(arr2)) THEN
01293         myvalout = .false.
01294
01295         RETURN
01296     END IF
01297
01298     myvalout = .true.
01299     IF (any(arr1 - arr2 /= 0)) myvalout = .false.
01300
01301     RETURN
01302
01303 END FUNCTION arrayequalint
01304
01305 !=====
01306 !-----
01307 ! SUBROUTINE  ARRAY EQUAL SINGLE
01308 !-----
01309 !-----
01310
01329 LOGICAL FUNCTION arrayequalsingle(arr1, arr2) RESULT(myValOut)
01330
01331     IMPLICIT NONE
01332
01333     ! Global variables
01334     REAL(sp), DIMENSION(:), INTENT(IN) :: arr1, arr2
01335
01336     ! Local variables
01337     INTEGER :: i
01338
01339
01340     IF (SIZE(arr1) /= SIZE(arr2)) THEN
01341         myvalout = .false.
01342
01343         RETURN
01344     END IF
01345
01346     myvalout = .true.
01347
01348     DO i = 1, SIZE(arr1, 1)
01349         IF (comparereals(arr1(i), arr2(i), 0.00000001_sp) /= 0) THEN
01350             myvalout = .false.
01351
01352             EXIT
01353         END IF
01354     END DO
01355
01356     RETURN
01357
01358 END FUNCTION arrayequalsingle
01359
01360 !=====
01361 !-----
01362 ! SUBROUTINE  ARRAY EQUAL SINGLE
01363 !-----
01364 !-----
01365
01384 LOGICAL FUNCTION arrayequaldouble(arr1, arr2) RESULT(myValOut)
01385
01386     IMPLICIT NONE
01387
01388     ! Global variables
01389     REAL(hp), DIMENSION(:), INTENT(IN) :: arr1, arr2
01390
01391     ! Local variables
01392     INTEGER :: i
01393
01394
01395     IF (SIZE(arr1) /= SIZE(arr2)) THEN
01396         myvalout = .false.
01397
01398         RETURN
01399     END IF
01400
01401     myvalout = .true.

```

```

01402
01403     DO i = 1, SIZE(arr1, 1)
01404         IF (compare_reals(arr1(i), arr2(i), 0.0000000000001_hp) /= 0) THEN
01405             myvalout = .false.
01406
01407             EXIT
01408         END IF
01409     END DO
01410
01411     RETURN
01412
01413 END FUNCTION arrayequaldouble
01414
01415 !=====
01416
01417 !-----
01418 ! FUNCTION STRING LEX COMP
01419 !-----
01420 !-----
01443 INTEGER FUNCTION stringlexcomp(str1, str2, mSensitive) RESULT(myValOut)
01444
01445     USE utilities, ONLY : touppercase
01446
01447     IMPLICIT NONE
01448
01449     ! Global variables
01450     CHARACTER(LEN=*) , INTENT(IN) :: str1, str2
01451     LOGICAL, OPTIONAL, INTENT(IN) :: msensitive
01452
01453     ! Local variables
01454     LOGICAL :: sflag
01455
01456     sflag = .true.
01457     IF (PRESENT(msensitive)) sflag = msensitive
01458
01459     IF (sflag) THEN
01460         IF (trim(str1) == trim(str2)) THEN
01461             myvalout = 0
01462         ELSE IF (trim(str1) < trim(str2)) THEN
01463             myvalout = -1
01464         ELSE
01465             myvalout = 1
01466         END IF
01467     ELSE
01468         IF (touppercase(trim(str1)) == touppercase(trim(str2))) THEN
01469             myvalout = 0
01470         ELSE IF (touppercase(trim(str1)) < touppercase(trim(str2))) THEN
01471             myvalout = -1
01472         ELSE
01473             myvalout = 1
01474         END IF
01475     END IF
01476
01477     RETURN
01478
01479 END FUNCTION stringlexcomp
01480
01481 !=====
01482
01483 !-----
01484 ! SUBROUTINE SWAP INT
01485 !-----
01508 !-----
01509 SUBROUTINE swapint(a, b, mask)
01510
01511     IMPLICIT NONE
01512
01513     ! Global variables
01514     INTEGER, INTENT(INOUT) :: a, b
01515     LOGICAL, OPTIONAL, INTENT(IN) :: mask
01516
01517     ! Local variables
01518     INTEGER :: dum
01519     LOGICAL :: mflag
01520
01521     mflag = .true.
01522     IF (PRESENT(mask)) mflag = mask
01523
01524     IF (mflag) THEN
01525         dum = a

```

```

01527         a   = b
01528         b   = dum
01529     END IF
01530
01531 END SUBROUTINE swapint
01532
01533 !=====
01534 !-----
01535 ! SUBROUTINE SWAP SINGLE
01536 !-----
01537 !-----
01560 !-----
01561 SUBROUTINE swapsingle(a, b, mask)
01562
01563     IMPLICIT NONE
01564
01565     ! Global variables
01566     REAL(SP), INTENT(INOUT)      :: a, b
01567     LOGICAL, OPTIONAL, INTENT(IN) :: mask
01568
01569     ! Local variables
01570     REAL(SP) :: dum
01571     LOGICAL   :: mFlag
01572
01573
01574     mflag = .true.
01575     IF (PRESENT(mask)) mflag = mask
01576
01577     IF (mflag) THEN
01578         dum = a
01579         a   = b
01580         b   = dum
01581     END IF
01582
01583 END SUBROUTINE swapsingle
01584
01585 !=====
01586 !-----
01587 ! SUBROUTINE SWAP DOUBLE
01588 !-----
01589 !-----
01612 !-----
01613 SUBROUTINE swapdouble(a, b, mask)
01614
01615     IMPLICIT NONE
01616
01617     ! Global variables
01618     REAL(HP), INTENT(INOUT)      :: a, b
01619     LOGICAL, OPTIONAL, INTENT(IN) :: mask
01620
01621     ! Local variables
01622     REAL(HP) :: dum
01623     LOGICAL   :: mFlag
01624
01625
01626     mflag = .true.
01627     IF (PRESENT(mask)) mflag = mask
01628
01629     IF (mflag) THEN
01630         dum = a
01631         a   = b
01632         b   = dum
01633     END IF
01634
01635 END SUBROUTINE swapdouble
01636
01637 !=====
01638 !-----
01639 ! SUBROUTINE SWAP INT VEC
01640 !-----
01641 !-----
01664 !-----
01665 SUBROUTINE swapintvec(a, b, mask)
01666
01667     IMPLICIT NONE
01668
01669     ! Global variables
01670     INTEGER, DIMENSION(:), INTENT(INOUT) :: a, b
01671     LOGICAL, OPTIONAL, INTENT(IN)         :: mask
01672
01673     ! Local variables

```

```

01674      INTEGER, DIMENSION(SIZE(a)) :: dum
01675      LOGICAL                        :: mFlag
01676
01677
01678      mflag = .true.
01679      IF (PRESENT(mask)) mflag = mask
01680
01681      IF (mflag) THEN
01682          dum = a
01683          a   = b
01684          b   = dum
01685      END IF
01686
01687  END SUBROUTINE swapintvec
01688
01689  !=====
01690
01691  !-----
01692  ! SUBROUTINE SWAP SINGLE VEC
01693  !-----
01694  !-----
01716  !-----
01717  SUBROUTINE swapsinglevec(a, b, mask)
01718
01719      IMPLICIT NONE
01720
01721      ! Global variables
01722      REAL(SP), DIMENSION(:), INTENT(INOUT) :: a, b
01723      LOGICAL, OPTIONAL, INTENT(IN)         :: mask
01724
01725      ! Local variables
01726      REAL(SP), DIMENSION(SIZE(a)) :: dum
01727      LOGICAL                        :: mFlag
01728
01729
01730      mflag = .true.
01731      IF (PRESENT(mask)) mflag = mask
01732
01733      IF (mflag) THEN
01734          dum = a
01735          a   = b
01736          b   = dum
01737      END IF
01738
01739  END SUBROUTINE swapsinglevec
01740
01741  !=====
01742
01743  !-----
01744  ! SUBROUTINE SWAP DOUBLE VEC
01745  !-----
01746  !-----
01768  !-----
01769  SUBROUTINE swapdoublevec(a, b, mask)
01770
01771      IMPLICIT NONE
01772
01773      ! Global variables
01774      REAL(HP), DIMENSION(:), INTENT(INOUT) :: a, b
01775      LOGICAL, OPTIONAL, INTENT(IN)         :: mask
01776
01777      ! Local variables
01778      REAL(HP), DIMENSION(SIZE(a)) :: dum
01779      LOGICAL                        :: mFlag
01780
01781
01782      mflag = .true.
01783      IF (PRESENT(mask)) mflag = mask
01784
01785      IF (mflag) THEN
01786          dum = a
01787          a   = b
01788          b   = dum
01789      END IF
01790
01791  END SUBROUTINE swapdoublevec
01792
01793  !=====
01794
01795  !-----
01796  ! SUBROUTINE ARTH INT
01797  !-----
01798  !-----
01817  !-----

```



```

01818 pure FUNCTION arthint(first, increment, n) RESULT(arthOut)
01819
01820     IMPLICIT NONE
01821
01822     ! Global variables
01823     INTEGER, INTENT(IN)    :: first, increment
01824     INTEGER, INTENT(IN)    :: n
01825     INTEGER, DIMENSION(n) :: arthout
01826
01827     ! Local variables
01828     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01829     INTEGER :: k, k2
01830     INTEGER :: temp
01831
01832
01833     IF (n > 0) arthout(1) = first
01834
01835     IF (n <= nparth) THEN
01836         DO k = 2, n
01837             arthout(k) = arthout(k - 1) + increment
01838         END DO
01839     ELSE
01840         DO k = 2, nparth2
01841             arthout(k) = arthout(k - 1) + increment
01842         END DO
01843
01844         temp = increment * nparth2
01845         k = nparth2
01846
01847         DO
01848             IF (k >= n) EXIT
01849             k2 = k + k
01850             arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01851             temp = temp + temp
01852             k = k2
01853         END DO
01854     END IF
01855
01856     RETURN
01857
01858 END FUNCTION arthint
01859
01860 !=====
01861
01862 !-----
01863 ! SUBROUTINE ARTH SINGLE
01864 !-----
01865 !-----
01885 pure FUNCTION arthsingle(first, increment, n) RESULT(arthOut)
01886
01887     IMPLICIT NONE
01888
01889     ! Global variables
01890     REAL(sp), INTENT(IN)    :: first, increment
01891     INTEGER, INTENT(IN)    :: n
01892     REAL(sp), DIMENSION(n) :: arthout
01893
01894     ! Local variables
01895     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01896     INTEGER :: k, k2
01897     REAL(sp) :: temp
01898
01899
01900     IF (n > 0) arthout(1) = first
01901
01902     IF (n <= nparth) THEN
01903         DO k = 2, n
01904             arthout(k) = arthout(k - 1) + increment
01905         END DO
01906     ELSE
01907         DO k = 2, nparth2
01908             arthout(k) = arthout(k - 1) + increment
01909         END DO
01910
01911         temp = increment * nparth2
01912         k = nparth2
01913
01914         DO
01915             IF (k >= n) EXIT
01916             k2 = k + k
01917             arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))

```

```

01918         temp = temp + temp
01919         k = k2
01920     END DO
01921 END IF
01922
01923     RETURN
01924
01925 END FUNCTION arthsingle
01926
01927 !=====
01928 !-----
01929 ! SUBROUTINE ARTH DOUBLE
01930 !-----
01931 !-----
01951 !-----
01952 pure FUNCTION arthdouble(first, increment, n) RESULT(arthOut)
01953
01954     IMPLICIT NONE
01955
01956     ! Global variables
01957     REAL(hp), INTENT(IN) :: first, increment
01958     INTEGER, INTENT(IN) :: n
01959     REAL(hp), DIMENSION(n) :: arthout
01960
01961     ! Local variables
01962     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01963     INTEGER :: k, k2
01964     REAL(hp) :: temp
01965
01966
01967     IF (n > 0) arthout(1) = first
01968
01969     IF (n <= nparth) THEN
01970         DO k = 2, n
01971             arthout(k) = arthout(k - 1) + increment
01972         END DO
01973     ELSE
01974         DO k = 2, nparth2
01975             arthout(k) = arthout(k - 1) + increment
01976         END DO
01977
01978         temp = increment * nparth2
01979         k = nparth2
01980
01981         DO
01982             IF (k >= n) EXIT
01983             k2 = k + k
01984             arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01985             temp = temp + temp
01986             k = k2
01987         END DO
01988     END IF
01989
01990     RETURN
01991
01992 END FUNCTION arthdouble
01993
01994 !=====
01995
01996 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::timeconvsec` (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.](#)

```

00001 !-----
00002 !           M O D U L E   T I M E   D A T E   U T I L S
00003 !-----
00014 !-----
00015
00016 MODULE timedateutils
00017
00018     USE pahm_sizes
00019     USE pahm_messages
00020
00021     PRIVATE :: upp
00022
00023 !-----
00024 !   I N T E R F A C E S
00025 !-----
00026 INTERFACE timeconv
00027     MODULE PROCEDURE timeconvsec
00028     MODULE PROCEDURE timeconvrsec
00029 END INTERFACE timeconv
00030
00031 INTERFACE gregtojulday
00032     MODULE PROCEDURE gregtojuldayisec
00033     MODULE PROCEDURE gregtojuldayrsec
00034     MODULE PROCEDURE gregtojulday2
00035 END INTERFACE gregtojulday
00036
00037 INTERFACE splitdatetimestring
00038     MODULE PROCEDURE splitdatetimestring
00039     MODULE PROCEDURE splitdatetimestring2

```

```

00040 END INTERFACE splitdatetimestring
00041 !-----
00042
00043 ! Julian day number for the first date of the Gregorian calendar (10/05/1582).
00044 INTEGER, PARAMETER :: firstgregdate = 1582 * 10000 + 10 * 100 + 05
00045 INTEGER, PARAMETER :: firstgregtime = 0 * 10000 + 0 * 100 + 0
00046 REAL(hp), PARAMETER :: offfirstgregday = 2299150.5_hp
00047
00048 ! A modified version of the Julian date denoted MJD obtained by subtracting
00049 ! 2,400,000.5 days from the Julian date JD, The MJD therefore gives the number
00050 ! of days since midnight of November 17, 1858. This date corresponds to
00051 ! 2400000.5 days after day 0 of the Julian calendar
00052 ! (https://scienceworld.wolfram.com/astronomy/ModifiedJulianDate.html).
00053 INTEGER, PARAMETER :: modjuldate = 1858 * 10000 + 11 * 100 + 17
00054 INTEGER, PARAMETER :: modjultime = 0 * 10000 + 0 * 100 + 0
00055 REAL(hp), PARAMETER :: offmodjulday = 2400000.5_hp
00056
00057 ! Julian day number for the first date of Unix time. This MJD gives the number
00058 ! of days since midnight of January 1, 1970.
00059 INTEGER, PARAMETER :: unixdate = 1970 * 10000 + 1 * 100 + 1
00060 INTEGER, PARAMETER :: unixtime = 0 * 10000 + 0 * 100 + 0
00061 REAL(hp), PARAMETER :: offunixjulday = 2440587.5_hp
00062
00063 ! Julian day number for the first date of Model time. This MJD gives the number
00064 ! of days since midnight of January 1, 1990.
00065 INTEGER, PARAMETER :: modeldate = 1990 * 10000 + 1 * 100 + 1
00066 INTEGER, PARAMETER :: modeltime = 0 * 10000 + 0 * 100 + 0
00067 REAL(hp), PARAMETER :: offmodeljulday = 2447892.5_hp
00068
00069 !----- MOD JUL DAY
00070 ! Definitions to use or not modified julian day calculations
00071 ! If USEMODJULDAY >= 1 use MJD calculation
00072 INTEGER, PARAMETER :: usemodjulday = 0
00073 !--- First option for a modified julian day
00074 !INTEGER, PARAMETER :: MDJDATE = MODJULDATE
00075 !INTEGER, PARAMETER :: MDJTIME = MODJULDATE
00076 !REAL(hp), PARAMETER :: MDJOFFSET = OFFMODJULDAY
00077 !---
00078
00079 !--- Second option for a modified julian day
00080 INTEGER, PARAMETER :: mdjdate = unixdate
00081 INTEGER, PARAMETER :: mdjtime = unixtime
00082 REAL(hp), PARAMETER :: mdjoffset = offunixjulday
00083
00084 !--- Third option for a modified julian day
00085 !INTEGER, PARAMETER :: MDJDATE = MODELDATE
00086 !INTEGER, PARAMETER :: MDJTIME = MODELTIME
00087 !REAL(hp), PARAMETER :: MDJOFFSET = OFFMODELJULDAY
00088 !---
00089 !-----
00090
00091
00092 CONTAINS
00093
00094
00095 !-----
00096 ! SUBROUTINE TIME CONV I SEC
00097 !-----
00124
00125 SUBROUTINE timeconvsec(iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)
00126
00127 USE pahm_global, ONLY : refyear, refmonth, refday, refhour, refmin, refsec
00128
00129 IMPLICIT NONE
00130
00131 ! Global variables
00132 INTEGER, INTENT(IN) :: iYear, iMonth, iDay, iHour, iMin, iSec
00133 REAL(SZ), INTENT(OUT) :: timeSec
00134
00135 ! Local variables
00136 REAL(SZ) :: jd0, jd1
00137 CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00138
00139 !----- START CALCULATIONS -----
00140
00141 CALL setmessagesource("TimeConv")
00142
00143 jd0 = gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
00144 jd1 = gregtojulday(iyear, imonth, iday, ihour, imin, isec)
00145
00146 IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN

```

```

00147         timesec = rmissv
00148
00149         WRITE(tmpstr1, '(f20.3)') jd0
00150         WRITE(tmpstr2, '(f20.3)') jd1
00151         WRITE(scratchmessage, '(a)') 'Invalid julian dates calculated: refJD = ' // &
00152             trim(adjustl(tmpstr1)) // ', inpJD = ' // trim(adjustl(tmpstr2))
00153
00154         CALL allmessage(error, scratchmessage)
00155         CALL unsetmessagesource()
00156
00157         CALL terminate()
00158     END IF
00159
00160     timesec = elapsedsecs(jd0, jd1, 'days')
00161
00162     CALL unsetmessagesource()
00163
00164     RETURN
00165
00166 END SUBROUTINE timeconvsec
00167
00168 !=====
00169
00170 !-----
00171 ! SUBROUTINE TIME CONVR SEC
00172 !-----
00201 !-----
00202 SUBROUTINE timeconvrsec(iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)
00203
00204     USE pahm_global, ONLY : refyear, refmonth, refday, refhour, refmin, refsec
00205
00206     IMPLICIT NONE
00207
00208     ! Global variables
00209     INTEGER, INTENT(IN) :: iYear, iMonth, iDay, iHour, iMin
00210     REAL(SZ), INTENT(IN) :: rSec
00211     REAL(SZ), INTENT(OUT) :: timeSec
00212
00213     ! Local variables
00214     REAL(SZ) :: jd0, jd1
00215     CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00216
00217     !----- START CALCULATIONS -----
00218
00219     CALL setmessagesource("TimeConv")
00220
00221     jd0 = gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
00222     jd1 = gregtojulday(iyear, imonth, iday, ihour, imin, rsec)
00223
00224     IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
00225         timesec = rmissv
00226
00227         WRITE(tmpstr1, '(f20.3)') jd0
00228         WRITE(tmpstr2, '(f20.3)') jd1
00229         WRITE(scratchmessage, '(a)') 'Invalid julian dates calculated: refJD = ' // &
00230             trim(adjustl(tmpstr1)) // ', inpJD = ' // trim(adjustl(tmpstr2))
00231
00232         CALL allmessage(error, scratchmessage)
00233         CALL unsetmessagesource()
00234
00235         CALL terminate()
00236     END IF
00237
00238     timesec = elapsedsecs(jd0, jd1, 'days')
00239
00240     CALL unsetmessagesource()
00241
00242     RETURN
00243
00244 END SUBROUTINE timeconvrsec
00245
00246 !=====
00247
00248 !DEL !-----
00249 !DEL ! SUBROUTINE TIME CONV ADCIRC <- TO BE DELETED
00250 !DEL !-----
00251 !DEL !-----
00252 !DEL SUBROUTINE TimeConvADCIRC(year, month, day, hour, minute, sec, timeSec)
00253
00254 !DEL IMPLICIT NONE
00255

```

```

00256 !DEL    INTEGER :: year, month, day, hour, minute, leap
00257 !DEL    REAL(SZ) :: timeSec, sec, secPerDay, secPerHour, secPerMin
00258
00259 !DEL    !----- START CALCULATIONS -----
00260
00261 !DEL    secPerDay = 86400_SZ
00262 !DEL    secPerHour = 3600.0_SZ
00263 !DEL    secPerMin = 60.0_SZ
00264
00265 !DEL    CALL SetMessageSource("TimeConv")
00266
00267 !DEL    timeSec = (day - 1) * secPerDay + hour * secPerHour + minute * secPerMin + sec
00268 !DEL    IF (month >= 2) timeSec = timeSec + 31 * secPerDay
00269
00270 !DEL    leap = (year / 4) * 4
00271 !DEL    IF ((leap == year) .AND. (month >= 3)) timeSec = timeSec + 29 * secPerDay
00272 !DEL    IF ((leap /= year) .AND. (month >= 3)) timeSec = timeSec + 28 * secPerDay
00273
00274 !DEL    IF (month >= 4) timeSec = timeSec + 31 * secPerDay
00275 !DEL    IF (month >= 5) timeSec = timeSec + 30 * secPerDay
00276 !DEL    IF (month >= 6) timeSec = timeSec + 31 * secPerDay
00277 !DEL    IF (month >= 7) timeSec = timeSec + 30 * secPerDay
00278 !DEL    IF (month >= 8) timeSec = timeSec + 31 * secPerDay
00279 !DEL    IF (month >= 9) timeSec = timeSec + 31 * secPerDay
00280 !DEL    IF (month >= 10) timeSec = timeSec + 30 * secPerDay
00281 !DEL    IF (month >= 11) timeSec = timeSec + 31 * secPerDay
00282 !DEL    IF (month == 12) timeSec = timeSec + 30 * secPerDay
00283
00284 !DEL    IF (month > 12) THEN
00285 !DEL        CALL AllMessage(ERROR, 'Fatal error in subroutine TimeConv: month > 12.')
00286 !DEL        CALL Terminate()
00287 !DEL    END IF
00288
00289 !DEL    CALL UnsetMessageSource()
00290
00291 !DEL    RETURN
00292
00293 !DEL END SUBROUTINE TimeConvADCIRC
00294
00295 !DEL=====
00296
00297 !-----
00298 ! FUNCTION LEAP YEAR
00299 !-----
00300 !-----
00301
00315 LOGICAL FUNCTION leapyear(iYear) RESULT(myVal)
00316
00317     IMPLICIT NONE
00318
00319     INTEGER, INTENT(IN) :: iyear
00320
00321     !----- START CALCULATIONS -----
00322
00323     IF (iyear < 1582) Then
00324         myval = .false.
00325
00326         RETURN
00327     END IF
00328
00329     ! ADCIRC uses the construct leap = (iYear / 4) * 4 == iYear
00330     ! to determine if a year is a leap year. This produces wrong
00331     ! results, example while 1700, 1900, 2100 are not leap years,
00332     ! the above construct determines that these years are leap years.
00333     ! Needs to be fixed.
00334
00335     IF ((mod(iyear, 100) /= 0) .AND. (mod(iyear, 4) == 0)) THEN
00336         myval = .true.
00337     ELSE IF (mod(iyear, 400) == 0) THEN
00338         myval = .true.
00339     ELSE
00340         myval = .false.
00341     END IF
00342
00343     RETURN
00344 END FUNCTION leapyear
00345
00346 !=====
00347
00348 !-----
00349 ! FUNCTION YEAR DAYS
00350 !-----

```

```

00365 !-----
00366 INTEGER FUNCTION yeardays(iYear) RESULT(myVal)
00367
00368     IMPLICIT NONE
00369
00370     INTEGER, INTENT(IN) :: iyear
00371
00372     !----- START CALCULATIONS -----
00373
00374     myval = 365
00375     IF (leapyear(iyear)) myval = 366
00376
00377     RETURN
00378 END FUNCTION yeardays
00379
00380 !=====
00381
00382 !-----
00383 ! F U N C T I O N   M O N T H   D A Y S
00384 !-----
00402 !-----
00403 INTEGER FUNCTION monthdays(iYear, iMonth) RESULT(myVal)
00404
00405     IMPLICIT NONE
00406
00407     ! Global variables
00408     INTEGER, INTENT(IN) :: iyear, imonth
00409
00410     ! Local variables
00411     INTEGER :: leap, monlen(12, 2)
00412
00413     !----- START CALCULATIONS -----
00414
00415     IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12)) THEN
00416         myval = imissv
00417
00418         RETURN
00419     END IF
00420
00421     ! Initialize lenghts of months:
00422     monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,    &
00423                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /), &
00424                     (/ 12, 2 /))
00425
00426     leap = 1
00427     IF (leapyear(iyear)) leap = 2
00428
00429     myval = monlen(imonth, leap)
00430
00431     RETURN
00432 END FUNCTION monthdays
00433
00434 !=====
00435
00436 !-----
00437 ! F U N C T I O N   D A Y   O F   Y E A R
00438 !-----
00459 !-----
00460 INTEGER FUNCTION dayofyear(iYear, iMonth, iDay) RESULT(myVal)
00461
00462     IMPLICIT NONE
00463
00464     ! Global variables
00465     INTEGER, INTENT(IN) :: iyear, imonth, iday
00466
00467     ! Local variables
00468     REAL(sz) :: jd0, jd1
00469
00470     !----- START CALCULATIONS -----
00471
00472     jd0 = gregtojulday(iyear, 1, 1, 0, 0, 0)
00473     jd1 = gregtojulday(iyear, imonth, iday, 0, 0, 0)
00474
00475     IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
00476         myval = imissv
00477
00478         RETURN
00479     END IF
00480
00481     myval = int(jd1 - jd0 + 1.0_sz)
00482

```



```

00483     RETURN
00484 END FUNCTION dayofyear
00485
00486 !=====
00487
00488 !-----
00489 ! FUNCTION GREG TO JUL DAY ISEC
00490 !-----
00535 !-----
00536 REAL(sz) function gregtojuldayisec(iyear, imonth, iday, ihour, imin, isec, mjd) result(myval)
00537
00538     IMPLICIT NONE
00539
00540     ! Global variables
00541     INTEGER, INTENT(IN)          :: iyear, imonth, iday, ihour, imin, isec
00542     INTEGER, OPTIONAL, INTENT(IN) :: mjd
00543
00544     ! Local variables
00545     INTEGER :: leap, monlen(12, 2)
00546     LOGICAL :: modjul
00547     REAL(hp) :: temp1, temp2
00548
00549     !----- START CALCULATIONS -----
00550
00551     modjul = .false.
00552     IF (PRESENT(mjd)) THEN
00553         modjul = (mjd > 0)
00554     ELSE
00555         modjul = (usemodjulday > 0)
00556     END IF
00557
00558     ! Initialize lengths of months:
00559     monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,    &
00560                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /), &
00561                     (/ 12, 2 /))
00562
00563     ! This function intentionally works on Gregorian dates only. For modeling
00564     ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00565     ! it is not necessary to go beyond that date.
00566
00567     ! Is this a LEAP year?
00568     leap = 1
00569     IF (leapyear(iyear)) leap = 2
00570
00571     IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
00572         myval = rmissv
00573     ELSE
00574         RETURN
00575     ELSE IF ((imonth < 1) .OR. (imonth > 12) .OR. &
00576             (iday < 1) .OR. (iday > monlen(imonth, leap)) .OR. &
00577             (ihour < 0) .OR. (ihour > 23) .OR. &
00578             (imin < 0) .OR. (imin > 59) .OR. &
00579             (isec < 0) .OR. (isec > 60)) THEN
00580         myval = rmissv
00581     ELSE
00582         RETURN
00583     ELSE
00584         temp1 = int((imonth - 14.0_hp) / 12.0_hp)
00585         temp2 = iday - 32075.0_hp &
00586             + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp) &
00587             + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp) &
00588             - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
00589         temp1 = real(ihour, hp) * 3600.0_hp &
00590             + real(imin, hp) * 60.0_hp &
00591             + real(isec, hp) - 43200.0_hp
00592
00593         IF (modjul) THEN
00594             print *, 'we are using mod jul with MDJOFFSET = ', mdjoffset
00595             myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
00596         ELSE
00597             myval = temp2 + (temp1 / 86400.0_hp)
00598         END IF
00599     END IF
00600
00601     RETURN
00602 END FUNCTION gregtojuldayisec
00603
00604 !=====
00605
00606 !-----
00607 ! FUNCTION GREG TO JUL DAY RSEC

```

```

00608 !-----
00654 !-----
00655 REAL(sz) function gregtojuldayrsec(iyear, imonth, iday, ihour, imin, rsec, mjd) result(myval)
00656
00657     IMPLICIT NONE
00658
00659     ! Global variables
00660     INTEGER, INTENT(IN)          :: iyear, imonth, iday, ihour, imin
00661     REAL(sz), INTENT(IN)         :: rsec
00662     INTEGER, OPTIONAL, INTENT(IN) :: mjd
00663
00664     ! Local variables
00665     INTEGER :: leap, monlen(12, 2)
00666     LOGICAL :: modjul
00667     REAL(hp) :: temp1, temp2
00668
00669     !----- START CALCULATIONS -----
00670
00671     modjul = .false.
00672     IF (PRESENT(mjd)) THEN
00673         modjul = (mjd > 0)
00674     ELSE
00675         modjul = (usemodjulday > 0)
00676     END IF
00677
00678     ! Initialize lengths of months:
00679     monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31,    &
00680                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /), &
00681                     (/ 12, 2 /))
00682
00683     ! This function intentionally works on Gregorian dates only. For modeling
00684     ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00685     ! it is not necessary to go beyond that date.
00686
00687     ! Is this a LEAP year?
00688     leap = 1
00689     IF (leapyear(iyear)) leap = 2
00690
00691     IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
00692         myval = rmissv
00693
00694         RETURN
00695     ELSE IF ((imonth < 1) .OR. (imonth > 12) .OR. &
00696             (iday < 1) .OR. (iday > monlen(imonth, leap)) .OR. &
00697             (ihour < 0) .OR. (ihour > 23) .OR. &
00698             (imin < 0) .OR. (imin > 59) .OR. &
00699             (rsec < 0) .OR. (rsec > 60)) THEN
00700         myval = rmissv
00701
00702         RETURN
00703     ELSE
00704         temp1 = int((imonth - 14.0_hp) / 12.0_hp)
00705         temp2 = iday - 32075.0_hp &
00706                + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp) &
00707                + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp) &
00708                - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
00709         temp1 = real(ihour, hp) * 3600.0_hp &
00710                + real(imin, hp) * 60.0_hp &
00711                + real(rsec, hp) - 43200.0_hp
00712
00713         IF (modjul) THEN
00714             myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
00715         ELSE
00716             myval = temp2 + (temp1 / 86400.0_hp)
00717         END IF
00718     END IF
00719
00720     RETURN
00721 END FUNCTION gregtojuldayrsec
00722
00723 !=====
00724
00725 !-----
00726 ! FUNCTION GREG TO JUL DAY I SEC 2
00727 !-----
00728 !-----
00776 REAL(sz) function gregtojulday2(idate, itime, mjd) result(myval)
00777
00778     IMPLICIT NONE
00779
00780     ! Global variables

```

```

00781     INTEGER, INTENT(IN)           :: idate, itime
00782     INTEGER, OPTIONAL, INTENT(IN) :: mjd
00783
00784     ! Local variables
00785     INTEGER :: iyear, imonth, iday, ihour, imin, isec
00786     INTEGER :: leap, monlen(12, 2)
00787     LOGICAL  :: modjul
00788     REAL(hp) :: temp1, temp2
00789
00790     !----- START CALCULATIONS -----
00791
00792     modjul = .false.
00793     IF (PRESENT(mjd)) THEN
00794         modjul = (mjd > 0)
00795     ELSE
00796         modjul = (usemodjulday > 0)
00797     END IF
00798
00799     ! Initialize lengths of months:
00800     monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, &
00801                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /), &
00802                     (/ 12, 2 /))
00803
00804     ! This function intentionally works on Gregorian dates only. For modeling
00805     ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00806     ! it is not necessary to go beyond that date.
00807
00808     ! Is this a LEAP year?
00809     leap = 1
00810     IF (leapyear(iyear)) leap = 2
00811
00812     CALL splitdate(idate, iyear, imonth, iday)
00813     CALL splitdate(itime, ihour, imin, isec)
00814
00815     IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12) &
00816         .OR. (iday < 1) .OR. (iday > monlen(imonth, leap)) &
00817         .OR. (ihour < 0) .OR. (ihour > 23) &
00818         .OR. (imin < 0) .OR. (imin > 59) &
00819         .OR. (isec < 0) .OR. (isec > 60)) THEN
00820         myval = rmissv
00821     ELSE
00822         RETURN
00823     ELSE
00824         IF (idate < firstgregdate) THEN
00825             myval = rmissv
00826         ELSE
00827             RETURN
00828         ELSE
00829             temp1 = int((imonth - 14.0_hp) / 12.0_hp)
00830             temp2 = iday - 32075.0_hp &
00831                   + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp) &
00832                   + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp) &
00833                   - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
00834             temp1 = real(ihour, hp) * 3600.0_hp &
00835                   + real(imin, hp) * 60.0_hp &
00836                   + real(isec, hp) - 43200.0_hp
00837
00838             IF (modjul) THEN
00839                 myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
00840             ELSE
00841                 myval = temp2 + (temp1 / 86400.0_hp)
00842             END IF
00843         END IF
00844     END IF
00845
00846     RETURN
00847 END FUNCTION gregtojulday2
00848
00849 !=====
00850
00851 !-----
00852 ! SUBROUTINE JUL DAY TO GREG
00853 !-----
00854 !-----
00855 SUBROUTINE juldaytogreg(julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
00856
00857     IMPLICIT NONE
00858
00859     ! Global Variables
00860     REAL(SZ), INTENT(IN) :: julDay
00861     INTEGER, OPTIONAL, INTENT(IN) :: mJD

```

```

00906     INTEGER, INTENT(OUT)          :: iYear, iMonth, iDay, iHour, iMin, iSec
00907
00908     ! Local Variables
00909     REAL(HP) :: temp1 , temp2 , temp3 , temp4 , temp5
00910     REAL(HP) :: thisJulDay, myJulDay, delta
00911     INTEGER  :: nTry
00912     LOGICAL  :: modJul
00913
00914     !----- START CALCULATIONS -----
00915
00916     modjul = .false.
00917     IF (PRESENT(mjd)) THEN
00918         modjul = (mjd > 0)
00919     ELSE
00920         modjul = (usemodjulday > 0)
00921     END IF
00922
00923     IF (modjul) THEN
00924         thisjulday = julday + mdjoffset
00925     ELSE
00926         thisjulday = julday
00927     END IF
00928
00929     ! Check for valid Julian day (Gregorian calendar only)
00930     IF (thisjulday < offfirstgregday) THEN
00931         iyear = imissv
00932         imonth = imissv
00933         iday = imissv
00934         ihour = imissv
00935         imin = imissv
00936         isec = imissv
00937
00938         RETURN
00939     END IF
00940
00941     delta = 0.0_hp
00942     ntry = 1
00943     DO WHILE (ntry <= 2)
00944         myjulday= thisjulday + delta
00945         temp4 = myjulday
00946         temp5 = dmod(myjulday, 1.0_hp)
00947
00948         IF (temp5 < 0.5) THEN
00949             temp3 = 0.5_hp + temp5
00950             temp4 = aint(temp4)
00951         ELSE
00952             temp3 = temp5 - 0.5_hp
00953             temp4 = aint(temp4) + 1.0_hp
00954         END IF
00955
00956         temp1 = temp4 + 68569.0
00957         temp2 = aint(4.0_hp * temp1 / 146097.0_hp)
00958         temp1 = temp1 - aint((146097.0_hp * temp2 + 3.0_hp) / 4.0_hp)
00959         iyear = int(4000.0_hp * (temp1 + 1.0_hp) / 1461001.0_hp)
00960         temp1 = temp1 - aint((1461.0_hp * iyear) / 4.0_hp) + 31.0_hp
00961         imonth = int(80.0_hp * temp1 / 2447.0_hp)
00962         iday = int(temp1 - aint(2447.0_hp * imonth / 80.0_hp))
00963         temp1 = aint(imonth / 11.0_hp)
00964         imonth = int(imonth + 2.0 - 12.0_hp * temp1)
00965         iyear = int(100.0_hp * (temp2 - 49.0_hp) + iyear + temp1)
00966         ihour = int(temp3 * 24.0_hp)
00967         imin = int(temp3 * 1440.0_hp - 60.0_hp * ihour)
00968         isec = nint(temp3 * 86400.0_hp - 3600.0_hp * ihour - 60.0_hp * imin)
00969
00970         IF (isec >= 60) THEN
00971             IF (ntry < 2) THEN
00972                 delta = 0.49999_hp / 86400.0_hp
00973                 ntry = ntry + 1
00974             ELSE
00975                 iyear = imissv
00976                 EXIT
00977             END IF
00978         ELSE
00979             EXIT
00980         END IF
00981     END DO
00982
00983     END SUBROUTINE juldaytogreg
00984
00985     !=====
00986

```

```

00987 !-----
00988 ! SUBROUTINE DAY OF YEAR TO GREG
00989 !-----
01010 !
01011 SUBROUTINE dayofyeartogreg(inYR, inDY, iYear, iMonth, iDay)
01012
01013     IMPLICIT NONE
01014
01015     ! Global Variables
01016     INTEGER, INTENT(IN) :: inYR, inDY
01017     INTEGER, INTENT(OUT) :: iYear, iMonth, iDay
01018
01019     ! Local Variables
01020     REAL(S2) :: julDay
01021     INTEGER :: yr, mo, da, hh, mm, ss
01022
01023     !----- START CALCULATIONS -----
01024
01025     ! Check for valid day of year (Gregorian calendar only)
01026     IF ((inYR < 1582) .OR. (inDY < 1) .OR. (inDY > 366) ) THEN
01027         iyear = imissv
01028         imonth = imissv
01029         iday = imissv
01030
01031         RETURN
01032     END IF
01033
01034     julday = gregtojulday(inYR, 1, 1, 0, 0, 0) + (inDY - 1) * 1.0_hp
01035
01036     CALL juldaytogreg(julday, yr, mo, da, hh, mm, ss)
01037
01038     iyear = yr
01039     imonth = mo
01040     iday = da
01041
01042 END SUBROUTINE dayofyeartogreg
01043
01044 !=====
01045
01046 !-----
01047 ! SUBROUTINE SPLIT DATE TIME STRING
01048 !-----
01072 !
01073 SUBROUTINE splitdatetimestring(inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
01074
01075     IMPLICIT NONE
01076
01077     ! Global Variables
01078     CHARACTER(LEN=*) , INTENT(IN) :: inDateTime
01079     INTEGER, INTENT(OUT) :: iYear, iMonth, iDay, iHour, iMin, iSec
01080
01081     ! Local Variables
01082     CHARACTER(LEN=LEN(inDateTime)) :: tmpDateStr
01083     INTEGER :: errIO
01084
01085     !----- START CALCULATIONS -----
01086
01087     tmpdatestr = preprocessdatetimestring(indatetime)
01088
01089     IF (trim(tmpdatestr) == "") THEN
01090         iyear = imissv
01091         imonth = 0
01092         iday = 0
01093         ihour = 0
01094         imin = 0
01095         isec = 0
01096
01097         RETURN
01098     END IF
01099
01100     READ(tmpdatestr(1:4), '(I4.4)', iostat=errio) iyear
01101     IF ((errio /= 0) .OR. (iyear < 1582)) iyear = imissv
01102
01103     READ(tmpdatestr(5:6), '(I2.2)', iostat=errio) imonth
01104     IF ((errio /= 0) .OR. (imonth < 1) .OR. (imonth > 12)) imonth = 0
01105
01106     READ(tmpdatestr(7:8), '(I2.2)', iostat=errio) iday
01107     IF ((errio /= 0) .OR. (iday < 0) .OR. (iday > monthdays(iyear, imonth))) iday = 0
01108
01109     READ(tmpdatestr(9:10), '(I2.2)', iostat=errio) ihour
01110     IF ((errio /= 0) .OR. (ihour < 0) .OR. (ihour >= 23)) ihour = 0

```

```

01111
01112     READ(tmpdatestr(11:12), '(I2.2)', iostat=errio) imin
01113     IF ((errio /= 0) .OR. (imin < 0) .OR. (imin >= 60)) imin = 0
01114
01115     READ(tmpdatestr(13:14), '(I2.2)', iostat=errio) isec
01116     IF ((errio /= 0) .OR. (isec < 0) .OR. (isec >= 60)) isec = 0
01117
01118 END SUBROUTINE splitdatetimestring
01119
01120 !=====
01121
01122 !-----
01123 ! SUBROUTINE SPLIT DATE TIME STRING 2
01124 !-----
01125 !-----
01126
01127 SUBROUTINE splitdatetimestring2(inDateTime, iDate, iTime)
01128
01129     IMPLICIT NONE
01130
01131     ! Global Variables
01132     CHARACTER(LEN=*) , INTENT(IN)      :: inDateTime
01133     INTEGER, INTENT(OUT)                :: iDate, iTime
01134
01135     ! Local Variables
01136     INTEGER                          :: iYear, iMonth, iDay, iHour, iMin, iSec
01137
01138     !----- START CALCULATIONS -----
01139
01140     CALL splitdatetimestring(indatetime, iyear, imonth, iday, ihour, imin, isec)
01141
01142     IF ((iyear == imissv) .OR. (imonth <= 0) .OR. (iday <= 0)) THEN
01143         idate = imissv
01144     ELSE
01145         idate = joindate(iyear, imonth, iday)
01146     END IF
01147
01148     itime = joindate(ihour, imin, isec)
01149
01150 END SUBROUTINE splitdatetimestring2
01151
01152 !=====
01153
01154 !-----
01155 ! FUNCTION PRE PROCESS DATE TIME STRING
01156 !-----
01157 !-----
01158
01159 FUNCTION preprocessdatetimestring(inDateTime) Result(myValOut)
01160
01161     IMPLICIT NONE
01162
01163     ! Global Variables
01164     CHARACTER(LEN=*) , INTENT(IN)      :: indatetime
01165     CHARACTER(LEN=LEN(inDateTime)) :: myvalout
01166
01167     ! Local Variables
01168     CHARACTER(LEN=1)                :: c
01169     INTEGER                          :: i, ipos
01170
01171     !----- START CALCULATIONS -----
01172
01173     myvalout = blank
01174     ipos = 1
01175
01176     DO i = 1, len(indatetime)
01177         c = indatetime(i:i)
01178         IF ((48 <= ichar(c)) .AND. (ichar(c) <= 57)) THEN
01179             myvalout(ipos:ipos) = c
01180             ipos = ipos + 1
01181         ENDIF
01182     END DO
01183
01184     RETURN
01185
01186 END FUNCTION preprocessdatetimestring
01187
01188 !=====
01189
01190 !-----
01191 ! FUNCTION JOIN DATE
01192 !-----
01193 !-----
01194

```

```

01241 INTEGER FUNCTION joindate(iYear, iMonth, iDay) RESULT(myVal)
01242
01243     IMPLICIT NONE
01244
01245     ! Global Variables
01246     INTEGER, INTENT(IN) :: iyear, imonth, iday
01247
01248     !----- START CALCULATIONS -----
01249
01250     myval = iyear * 10000 + imonth * 100 + iday
01251
01252 END FUNCTION joindate
01253
01254 !=====
01255
01256 !-----
01257 ! SUBROUTINE SPLIT DATE
01258 !-----
01280 !-----
01281 SUBROUTINE splitdate(inDate, iYear, iMonth, iDay)
01282
01283     IMPLICIT NONE
01284
01285     ! Global Variables
01286     INTEGER, INTENT(IN) :: inDate
01287     INTEGER, INTENT(OUT) :: iYear, iMonth, iDay
01288
01289     !----- START CALCULATIONS -----
01290
01291     iyear = indate / 10000
01292     imonth = indate / 100 - iyear * 100
01293     iday = indate - imonth * 100 - iyear * 10000
01294
01295 END SUBROUTINE splitdate
01296
01297 !=====
01298
01299 !-----
01300 ! FUNCTION DATE TIME 2 STRING
01301 !-----
01323 !
01324 !         (optional - for sep <= 0 use ' ', for sep > 0 use 'T')
01326 !         (optional - units = [S(seconds), M(minutes), H(hours), D(days), W(weeks)])
01335 !-----
01336 FUNCTION datetime2string(year, month, day, hour, min, sec, sep, units, zone, err) result(myValOut)
01337
01338     IMPLICIT NONE
01339
01340     INTEGER, INTENT(IN) :: year, month, day
01341     INTEGER, OPTIONAL, INTENT(IN) :: sep, hour, min, sec
01342     CHARACTER(LEN=*) , OPTIONAL, INTENT(IN) :: units, zone
01343     INTEGER, OPTIONAL, INTENT(OUT) :: err ! Error status, 0 if success, nonzero in case of format
01344     error.
01345
01346     ! The resulting date time string. Considering using trim() on it.
01347     CHARACTER(LEN=64) :: myvalout
01348     CHARACTER(LEN=20) :: myunits, myzone
01349     CHARACTER(LEN=1) :: mytimesep
01350     INTEGER :: myhour, mymin, mysec, myerr
01351
01352     myhour = 0
01353     IF (PRESENT(hour)) myhour = hour
01354     mymin = 0
01355     IF (PRESENT(min)) mymin = min
01356     mysec = 0
01357     IF (PRESENT(sec)) mysec = sec
01358
01359     mytimesep = ' '
01360     IF (PRESENT(sep)) THEN
01361         IF (sep > 0) mytimesep = 'T'
01362         IF (sep <= 0) mytimesep = ' '
01363     END IF
01364
01365     IF (PRESENT(units)) THEN
01366         SELECT CASE(trim(adjustl(upp(units))))
01367             CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01368                 myunits = 'seconds since'
01369             CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01370                 myunits = 'minutes since'
01371             CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01372                 myunits = 'hours since'
01373             CASE('DAYS', 'DAY', 'DA', 'D')

```

```

01373         myunits = 'days since'
01374         CASE ('WEEKS', 'WEEK', 'WE', 'W')
01375             myunits = 'weeks since'
01376         CASE DEFAULT
01377             myvalout = ' '
01378         END SELECT
01379     ELSE
01380         myunits = ' '
01381     END IF
01382
01383     IF (PRESENT(zone)) THEN
01384         myzone = adjustl(zone)
01385     ELSE
01386         myzone = ' '
01387     END IF
01388
01389     !WRITE(myValOut, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":", i2.2, ":", i2.2, "Z")', IOSTAT=myErr) &
01390     !      year, month, day, myTimeSep, myHour, myMin, mySec
01391     WRITE(myvalout, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":", i2.2, ":", i2.2)', iostat=myerr) &
01392     year, month, day, mytimesep, myhour, mymin, mysec
01393
01394     IF (len_trim(myunits) /= 0) THEN
01395         myvalout = trim(myunits) // " " // trim(myvalout)
01396     END IF
01397
01398     IF (len_trim(myzone) /= 0) THEN
01399         myvalout = trim(myvalout) // " " // trim(myzone)
01400     END IF
01401
01402     IF (PRESENT(err)) err = myerr
01403
01404     RETURN
01405
01406 END FUNCTION datetime2string
01407
01408 !=====
01409 !-----
01410 ! FUNCTION GET TIME CONV SEC
01411 !-----
01412 !-----
01413 !-----
01431 REAL(sz) function gettimeconvsec(units, invert) result(myvalout)
01432
01433     IMPLICIT NONE
01434
01435     CHARACTER(LEN=*) , INTENT(IN) :: units
01436     INTEGER, OPTIONAL, INTENT(IN) :: invert
01437
01438     INTEGER :: myinvert
01439     CHARACTER(LEN=LEN(units)) :: myunits
01440     REAL(sz), PARAMETER :: minsecs = 60.0_sz
01441     REAL(sz), PARAMETER :: hoursecs = 3600.0_sz
01442     REAL(sz), PARAMETER :: daysecs = 86400.0_sz
01443     REAL(sz), PARAMETER :: weeksecs = 604800.0_sz
01444
01445
01446     myinvert = 0
01447     IF (PRESENT(invert)) THEN
01448         IF (invert > 0) myinvert = 1
01449         IF (invert <= 0) myinvert = 0
01450     END IF
01451
01452     myunits = adjustl(units)
01453     IF (myinvert == 0) THEN
01454         SELECT CASE(trim(upp(myunits)))
01455             CASE ('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01456                 myvalout = 1.0_sz
01457             CASE ('MINUTES', 'MINUTE', 'MIN', 'M')
01458                 myvalout = minsecs
01459             CASE ('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01460                 myvalout = hoursecs
01461             CASE ('DAYS', 'DAY', 'DA', 'D')
01462                 myvalout = daysecs
01463             CASE ('WEEKS', 'WEEK', 'WE', 'W')
01464                 myvalout = weeksecs
01465             CASE DEFAULT
01466                 myvalout = 1.0_sz
01467         END SELECT
01468     ELSE
01469         SELECT CASE(trim(upp(myunits)))
01470             CASE ('SECONDS', 'SECOND', 'SE', 'SC', 'S')

```



```

01471         myvalout = 1.0_sz
01472         CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01473             myvalout = 1.0_sz / minsecs
01474         CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01475             myvalout = 1.0_sz / hoursecs
01476         CASE('DAYS', 'DAY', 'DA', 'D')
01477             myvalout = 1.0_sz / daysecs
01478         CASE('WEEKS', 'WEEK', 'WE', 'W')
01479             myvalout = 1.0_sz / weeksecs
01480         CASE DEFAULT
01481             myvalout = 1.0_sz
01482     END SELECT
01483 END IF
01484
01485     RETURN
01486
01487 END FUNCTION gettimeconvsec
01488
01489 !=====
01490
01491 !-----
01492 ! FUNCTION ELAPSED SECS
01493 !-----
01494 !-----
01516
01517 REAL(sz) function elapsedsecs(intime1, intime2, inunits) result(myval)
01518
01519     IMPLICIT NONE
01520
01521     ! Global Variables
01522     REAL(sz), INTENT(IN) :: intime1, intime2
01523     CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: inunits
01524
01525     ! Local Variables
01526     REAL(sz) :: uconfac
01527     CHARACTER(LEN=:), ALLOCATABLE :: unitsval
01528
01529     !----- START CALCULATIONS -----
01530
01531     IF (PRESENT(inunits)) THEN
01532         ALLOCATE (CHARACTER(LEN=LEN(inunits)) :: unitsval)
01533         unitsval = inunits
01534     ELSE
01535         ALLOCATE (CHARACTER(LEN=1) :: unitsval)
01536         unitsval = 'S'
01537     END IF
01538
01539     uconfac = gettimeconvsec(unitsval)
01540
01541     myval = (intime2 - intime1) * uconfac
01542     myval = fixnearwholereal(myval, 0.001_sz)
01543
01544     DEALLOCATE(unitsval)
01545
01546     RETURN
01547
01548 END FUNCTION elapsedsecs
01549
01550 !=====
01551
01552 !-----
01553 ! FUNCTION UPP
01554 !-----
01555 !-----
01566
01567 FUNCTION upp(inpString) RESULT(outString)
01568
01569     CHARACTER(*), INTENT(IN) :: inpstring
01570
01571     INTEGER, PARAMETER :: duc = ichar('A') - ichar('a')
01572     CHARACTER(LEN(inpString)) :: outstring
01573     CHARACTER :: ch
01574     INTEGER :: i
01575
01576     DO i = 1, len(inpstring)
01577         ch = inpstring(i:i)
01578         IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
01579         outstring(i:i) = ch
01580     END DO
01581
01582     RETURN
01583
01584 END FUNCTION upp

```

```
01585
01586 !=====
01587
01588 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.](#)

```

00001 !-----
00002 !           M O D U L E   U T I L I T I E S
00003 !-----
00014 !-----
00015
00016 MODULE utilities
00017
00018   USE pahm_sizes
00019   USE pahm_messages
00020
00021   IMPLICIT NONE
00022
00023   INTEGER, PRIVATE      :: numBTFiles = 0
00024   REAL(sz), PARAMETER  :: closetol = 0.001_sz
00025
00026 !-----
00027 ! I N T E R F A C E S
00028 !-----
00029 INTERFACE geotocpp
00030   MODULE PROCEDURE geotocpp_scalar
00031   MODULE PROCEDURE geotocpp_ld
00032 END INTERFACE geotocpp
00033
00034 INTERFACE cpptogeo
00035   MODULE PROCEDURE cpptogeo_scalar
00036   MODULE PROCEDURE cpptogeo_ld
00037 END INTERFACE cpptogeo
00038
00039 INTERFACE sphericaldistance
00040   MODULE PROCEDURE sphericaldistance_scalar
00041   MODULE PROCEDURE sphericaldistance_ld
00042   MODULE PROCEDURE sphericaldistance_2d
00043 END INTERFACE sphericaldistance
00044 !-----
00045
00046 CONTAINS
00047
00048 !-----
00049 !           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
00050 !-----
00051 !-----
00052 !-----
00067 !-----
00068 SUBROUTINE openfileforread(lun, fileName, errorIO)
00069
00070   USE pahm_global
00071
00072   IMPLICIT NONE
00073
00074   ! Global variables
00075   INTEGER, INTENT(IN)      :: lun           ! fortran logical unit number
00076   CHARACTER(LEN=*), INTENT(IN) :: fileName  ! full pathname of file
00077   INTEGER, INTENT(OUT)     :: errorIO      ! zero if the file opened successfully
00078
00079   ! Local variables
00080   LOGICAL                  :: fileFound    ! .TRUE. if the file is present
00081   CHARACTER(LEN=LEN(fileName)) :: tmpFileName ! full pathname of file
00082
00083   CALL setmessagesource("OpenFileForRead")
00084
00085   errorio = 0
00086
00087   tmpfilename = adjustl(filename)
00088
00089   ! Check to see if file exists
00090   WRITE(scratchmessage, '("Searching for file to open on unit ", i0, "...")') lun
00091   CALL logmessage(info, trim(scratchmessage))
00092
00093   INQUIRE(file=trim(filename), exist=filefound)
00094   IF (.NOT. filefound) THEN
00095     WRITE(scratchmessage, '("The file : ", a, " was not found.")') trim(tmpfilename)
00096     CALL allmessage(info, scratchmessage)
00097
00098     errorio = 1
00099
00100     CALL unsetmessagesource()

```

```

00101
00102     RETURN ! file not found
00103 ELSE
00104     WRITE(scratchmessage, '( "The file : ", a, " was found. The file will be opened." )' )
trim(tmpfilename)
00105
00106     CALL logmessage(info, trim(scratchmessage))
00107 END IF
00108
00109 ! Open existing file
00110 OPEN(unit=lun, file=trim(tmpfilename), status='OLD', action='READ', iostat=errorio)
00111 IF (errorio /= 0) THEN
00112     WRITE(scratchmessage, '( "Could not open the file: ", a, "." )' ) trim(tmpfilename)
00113
00114     CALL allmessage(error, trim(scratchmessage))
00115
00116     CALL unsetmessagesource()
00117
00118     RETURN ! file found but could not be opened
00119 ELSE
00120     WRITE(scratchmessage, '( "The file ", a, " was opened successfully." )' ) trim(tmpfilename)
00121
00122     CALL logmessage(info, trim(scratchmessage))
00123 END IF
00124
00125 CALL unsetmessagesource()
00126
00127 RETURN
00128
00129 END SUBROUTINE openfileforread
00130
00131 !=====
00132
00133 !-----
00134 !   SUBROUTINE READ CONTROL FILE
00135 !-----
00152 !-----
00153 SUBROUTINE readcontrolfile(inpFile)
00154
00155     USE pahm_global
00156     USE pahm_messages
00157     USE timedateutils, ONLY : timeconv, splitdatetimestring, joindate, gregtojulday, juldaytogreg, &
00158                             gettimeconvsec, datetime2string
00159
00160     IMPLICIT NONE
00161
00162     ! Global variables
00163     CHARACTER(LEN=*) , INTENT(IN) :: inpFile
00164
00165     ! Local variables
00166     LOGICAL :: fileFound ! .TRUE. if the file is present
00167     CHARACTER(LEN=LEN(inpFile)) :: tmpFileName
00168     CHARACTER(LEN=512) :: inpLine, outLine
00169     CHARACTER(LEN=40) :: keyWord
00170
00171     INTEGER :: iUnit, errIO, status
00172
00173     INTEGER :: nPnts
00174     INTEGER :: nVal, i
00175     REAL(SZ), DIMENSION(200) :: realVal
00176     CHARACTER(LEN=512), DIMENSION(200) :: charVal
00177     CHARACTER(LEN=512) :: tmpCharVal
00178
00179     INTEGER :: iValOut(1)
00180     REAL(SZ) :: rValOut(1)
00181
00182     LOGICAL :: wrtPARS, gotNBTRFILES = .false.
00183
00184     CHARACTER(LEN=512) :: cntlFmtStr, fmtDimParInvalid, fmtParNotFound
00185     CHARACTER(LEN=FNAMELEN) :: tmpStr
00186     REAL(SZ) :: jday
00187
00188
00189     !----- Initialize variables
00190     ! Global variables
00191     numbtfiles = 0
00192
00193     ! Local variables
00194     inline = blank
00195     outline = blank
00196     keyword = blank

```

```

00197      charval                      = blank
00198      cntlfmtstr                   = blank
00199      fmtdimparinvalid             = blank
00200      fmtparnotfound               = blank
00201      tmpstr                       = blank
00202
00203      iunit = lun_ctrl
00204      errio = 0
00205      !-----
00206
00207
00208      CALL setmessagesource("ReadControlFile")
00209
00210      !----- Establish the format variables
00211      cntlfmtstr = ' "in control file ' // "<" // trim(adjustl(inpfile)) // ">" // '''
00212
00213      fmtdimparinvalid = ' (" Invalid dimension parameter: ", a, 1x, i0, 2x, '
00214      fmtdimparinvalid = trim(fmtdimparinvalid) // trim(cntlfmtstr) // ', 1x, a)'
00215
00216      fmtparnotfound = ' (" Could not find input parameter: ", a, 1x, '
00217      fmtparnotfound = trim(fmtparnotfound) // trim(cntlfmtstr) // ', 1x, a)'
00218      !-----
00219
00220      tmpfilename = adjustl(inpfile)
00221
00222      INQUIRE(file=trim(tmpfilename), exist=filefound)
00223      IF (.NOT. filefound) THEN
00224          WRITE(lun_screen, ' ("The control file : ", a, " was not found, cannot continue.")')
00225          trim(tmpfilename)
00226          stop ! file not found
00227      ELSE
00228          WRITE(lun_screen, ' ("The control file : ", a, " was found and will be opened for reading.")')
00229          trim(tmpfilename)
00230          END IF
00231
00232      ! Open existing file
00233      OPEN(unit=iunit, file=trim(tmpfilename), status='OLD', action='READ', iostat=errio)
00234      IF (errio /= 0) THEN
00235          WRITE(lun_screen, ' ("Could not open the control file: ", a, ".")') trim(tmpfilename)
00236          stop ! file found but could not be opened
00237      END IF
00238
00239      DO WHILE (.true.)
00240          READ(unit=iunit, fmt='(a)', err=10, END=20) inline
00241          status = parseline(inline, outline, keyword, nval, charval, realval)
00242
00243          IF (status > 0) THEN
00244              SELECT CASE (toupper(trim(keyword)))
00245              !----- CASE
00246              CASE ('TITLE')
00247                  IF (nval == 1) THEN
00248                      title = trim(adjustl(charval(nval)))
00249                  ELSE
00250                      WRITE(title, '(a, 1x, a)') trim(adjustl(title)), trim(adjustl(charval(nval)))
00251                  END IF
00252
00253              !----- CASE
00254              CASE ('LOGFILENAME')
00255                  IF (nval == 1) THEN
00256                      logfilename = trim(adjustl(charval(nval)))
00257                  ELSE
00258                      IF (trim(adjustl(logfilename)) == "") THEN
00259                          logfilename = trim(adjustl(charval(nval)))
00260                      END IF
00261                  END IF
00262
00263              !----- CASE
00264              CASE ('WRITEPARAMS')
00265                  npnts = loadintvar(nval, realval, 1, ivalout)
00266                  IF (ivalout(1) > 0) THEN
00267                      writeparams = .true.
00268                  ELSE
00269                      writeparams = .false.
00270                  END IF
00271
00272              !----- CASE
00273              CASE ('NBTRFILES')
00274                  npnts = loadintvar(nval, realval, 1, ivalout)
00275                  nbtrfiles = ivalout(1)

```

```

00276         IF (nbtrfiles > 0) THEN
00277             ALLOCATE(besttrackfilename(nbtrfiles))
00278             besttrackfilename = blank
00279         END IF
00280         gotnbtrfiles = .true.
00281
00282         !----- CASE
00283         CASE ('BESTTRACKFILENAME')
00284             IF (.NOT. gotnbtrfiles) THEN
00285                 WRITE(scratchmessage, fmtparnotfound) 'nBTrFiles', '(add the "nBTrFiles" keyword before
"bestTrackFileName").'
00286                 CALL allmessage(error, scratchmessage)
00287             ELSE
00288                 IF (ALLOCATED(besttrackfilename)) THEN
00289                     tmpstr = adjustl(charval(nval))
00290                     IF (trim(tmpstr) == "") THEN
00291                         nval = nval - 1
00292                     ELSE
00293                         IF (nval <= nbtrfiles) THEN ! because bestTrackFileName has been allocated this way
above
00294                             numbtfiles = numbtfiles + 1
00295                             besttrackfilename(numbtfiles) = trim(tmpstr)
00296                             besttrackfilenamespecified = .true.
00297                         END IF
00298                     END IF
00299                 END IF
00300             END IF
00301
00302         !----- CASE
00303         CASE ('MESHFILETYPE')
00304             IF (nval == 1) THEN
00305                 meshfiletype = trim(adjustl(charval(nval)))
00306             ELSE
00307                 IF (trim(adjustl(meshfiletype)) == "") THEN
00308                     meshfiletype = trim(adjustl(charval(nval)))
00309                 END IF
00310             END IF
00311
00312         !----- CASE
00313         CASE ('MESHFILENAME')
00314             IF (nval == 1) THEN
00315                 meshfilename = trim(adjustl(charval(nval)))
00316             ELSE
00317                 IF (trim(adjustl(meshfilename)) == "") THEN
00318                     meshfilename = trim(adjustl(charval(nval)))
00319                 END IF
00320             END IF
00321             IF (trim(adjustl(meshfilename)) /= "") meshfilenamespecified = .true.
00322
00323         !----- CASE
00324         CASE ('MESHFILEFORM')
00325             IF (nval == 1) THEN
00326                 meshfileform = trim(adjustl(charval(nval)))
00327             ELSE
00328                 IF (trim(adjustl(meshfileform)) == "") THEN
00329                     meshfileform = trim(adjustl(charval(nval)))
00330                 END IF
00331             END IF
00332
00333         !----- CASE
00334         CASE ('GRAVITY')
00335             npnts = loadrealvar(nval, realval, 1, rvalout)
00336             gravity = rvalout(1)
00337
00338         !----- CASE
00339         CASE ('RHOWATER')
00340             npnts = loadrealvar(nval, realval, 1, rvalout)
00341             rhowater = rvalout(1)
00342
00343         !----- CASE
00344         CASE ('RHOAIR')
00345             npnts = loadrealvar(nval, realval, 1, rvalout)
00346             rhoair = rvalout(1)
00347
00348         !----- CASE
00349         CASE ('BACKGROUNDATMPRESS')
00350             npnts = loadrealvar(nval, realval, 1, rvalout)
00351             backgroundatmpress = rvalout(1)
00352
00353         !----- CASE
00354         CASE ('BLADJUSTFAC')

```

```

00355         npnts = loadrealvar(nval, realval, 1, rvalout)
00356         bladjustfac = rvalout(1)
00357
00358     !----- CASE
00359     CASE ('REFDATETIME')
00360         IF (nval == 1) THEN
00361             refdatetime = trim(adjustl(charval(nval)))
00362         ELSE
00363             IF (trim(adjustl(refdatetime)) == "") THEN
00364                 refdatetime = trim(adjustl(charval(nval)))
00365             END IF
00366         END IF
00367
00368         CALL splitdatetimestring(refdatetime, refyear, refmonth, reftday, refhour, refmin, refsec)
00369
00370         IF ((refyear == imissv) .OR. (refmonth <= 0) .OR. (reftday <= 0)) THEN
00371             reftime = imissv
00372         ELSE
00373             reftime = joindate(refyear, refmonth, reftday)
00374         END IF
00375         reftime = joindate(refhour, refmin, refsec)
00376         refdatespecified = .true.
00377
00378     !----- CASE
00379     CASE ('UNITTIME')
00380         IF (begdatespecified .OR. begsimspecified .OR. &
00381             enddatespecified .OR. endsimspecified) THEN
00382             scratchmessage = 'add the "unitTime" keyword before the ' // &
00383                 '"begDateTime"/"begTime" and "endDateTime"/"endTime" keywords'
00384             CALL allmessage(error, scratchmessage)
00385             CALL terminate()
00386         ELSE
00387             IF (nval == 1) THEN
00388                 tmpcharval = touppercase(adjustl(charval(nval)))
00389                 unittime = tmpcharval(1:1)
00390             ELSE
00391                 IF (trim(adjustl(unittime)) == "") THEN
00392                     tmpcharval = touppercase(adjustl(charval(nval)))
00393                     unittime = tmpcharval(1:1)
00394                 END IF
00395             END IF
00396         END IF
00397
00398     !----- CASE
00399     CASE ('BEGDATETIME')
00400         IF (begdatespecified .OR. begsimspecified) THEN
00401             scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
00402             CALL allmessage(error, scratchmessage)
00403
00404             begdatespecified = .false.
00405         ELSE
00406             IF (.NOT. refdatespecified) THEN
00407                 scratchmessage = 'Add the "refDateTime" keyword before "begDateTime".'
00408                 CALL allmessage(error, scratchmessage)
00409             ELSE
00410                 IF (nval == 1) THEN
00411                     begdatetime = trim(adjustl(charval(nval)))
00412                 ELSE
00413                     IF (trim(adjustl(begdatetime)) == "") THEN
00414                         begdatetime = trim(adjustl(charval(nval)))
00415                     END IF
00416                 END IF
00417
00418                 CALL splitdatetimestring(begdatetime, begyear, begmonth, begday, beghour, begmin, begsec)
00419
00420                 IF ((begyear == imissv) .OR. (begmonth <= 0) .OR. (begday <= 0)) THEN
00421                     begdate = imissv
00422                 ELSE
00423                     begdate = joindate(begyear, begmonth, begday)
00424                 END IF
00425                 begtime = joindate(beghour, begmin, begsec)
00426
00427                 CALL timeconv(begyear, begmonth, begday, beghour, begmin, begsec, mdbegsimtime)
00428                 begsimtime = mdbegsimtime * gettimeconvsec(unittime, 1)
00429
00430                 begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)
00431
00432                 begdatespecified = .true.
00433                 begsimspecified = .true.
00434             END IF
00435         END IF

```



```

00436
00437 !----- CASE
00438 CASE ('ENDDATETIME')
00439 IF (enddatespecified .OR. endsimspecified) THEN
00440   scratchmessage = 'Only one of "endDateTime" or "endSimTime" can be specified'
00441   CALL allmessage(error, scratchmessage)
00442
00443   enddatespecified = .false.
00444 ELSE
00445   IF (.NOT. refdatespecified) THEN
00446     scratchmessage = 'Add the "refDateTime" keyword before "endDateTime".'
00447     CALL allmessage(error, scratchmessage)
00448   ELSE
00449     IF (nval == 1) THEN
00450       enddatetime = trim(adjustl(charval(nval)))
00451     ELSE
00452       IF (trim(adjustl(enddatetime)) == "") THEN
00453         enddatetime = trim(adjustl(charval(nval)))
00454       END IF
00455     END IF
00456
00457     CALL splitdatetimestring(enddatetime, endyear, endmonth, endday, endhour, endmin, endsec)
00458
00459     IF ((endyear == imissv) .OR. (endmonth <= 0) .OR. (endday <= 0)) THEN
00460       enddate = imissv
00461     ELSE
00462       enddate = joindate(endyear, endmonth, endday)
00463     END IF
00464     endtime = joindate(endhour, endmin, endsec)
00465
00466     CALL timeconv(endyear, endmonth, endday, endhour, endmin, endsec, mdendsimtime)
00467     endsimtime = mdendsimtime * gettimeconvsec(unittime, 1)
00468
00469     enddatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)
00470
00471     enddatespecified = .true.
00472     endsimspecified = .true.
00473   END IF
00474 END IF
00475
00476 !----- CASE
00477 CASE ('OUTDT')
00478   npnts = loadrealvar(nval, realval, 1, rvalout)
00479   outdt = rvalout(1)
00480   mdoutdt = fixnearwholereal(outdt * gettimeconvsec(unittime), closetol)
00481
00482 !----- CASE
00483 CASE ('BEGSIMTIME')
00484 IF (begdatespecified .OR. begsimspecified) THEN
00485   scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
00486   CALL allmessage(error, scratchmessage)
00487
00488   begsimspecified = .false.
00489 ELSE
00490   IF (.NOT. refdatespecified) THEN
00491     scratchmessage = 'Add the "refDateTime" keyword before "begSimTime".'
00492     CALL allmessage(error, scratchmessage)
00493   ELSE
00494     npnts = loadrealvar(nval, realval, 1, rvalout)
00495     begsimtime = rvalout(1)
00496
00497     mdbegsimtime = begsimtime * gettimeconvsec(unittime)
00498
00499     jday = (mdbegsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
00500 refhour, refmin, refsec)
00501     CALL juldaytogreg(jday, begyear, begmonth, begday, beghour, begmin, begsec)
00502     begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)
00503
00504     begdatespecified = .true.
00505     begsimspecified = .true.
00506   END IF
00507 END IF
00508
00509 !----- CASE
00510 CASE ('ENDSIMTIME')
00511 IF (enddatespecified .OR. endsimspecified) THEN
00512   scratchmessage = 'Only one of "endDateTime" and "endSimTime" can be specified'
00513   CALL allmessage(error, scratchmessage)
00514
00515   endsimspecified = .false.
00516 ELSE

```

```

00516         IF (.NOT. refdatespecified) THEN
00517             scratchmessage = 'Add the "refDateTime" keyword before "endSimTime").'
00518             CALL allmessage(error, scratchmessage)
00519         ELSE
00520             npnts = loadrealvar(nval, realval, 1, rvalout)
00521             endsimtime = rvalout(1)
00522
00523             mdendsimtime = endsimtime * gettimeconvsec(unittime)
00524
00525             jday = (mdendsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
refhour, refmin, refsec)
00526             CALL juldaytogreg(jday, endyear, endmonth, endday, endhour, endmin, endsec)
00527             begdatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)
00528
00529             enddatespecified = .true.
00530             endsimspecified = .true.
00531         END IF
00532     END IF
00533
00534     !----- CASE
00535     CASE ('OUTFILENAME')
00536         IF (nval == 1) THEN
00537             outfile = trim(adjustl(charval(nval)))
00538         ELSE
00539             IF (trim(adjustl(outfile)) == "") THEN
00540                 outfile = trim(adjustl(charval(nval)))
00541             END IF
00542         END IF
00543         IF (trim(adjustl(outfile)) /= "") outfilenamespecified = .true.
00544
00545     !----- CASE
00546     CASE ('NCVARNAM_PRES')
00547         IF (nval == 1) THEN
00548             ncvarnam_pres = trim(adjustl(charval(nval)))
00549         ELSE
00550             IF (trim(adjustl(ncvarnam_pres)) == "") THEN
00551                 ncvarnam_pres = trim(adjustl(charval(nval)))
00552             END IF
00553         END IF
00554
00555     !----- CASE
00556     CASE ('NCVARNAM_WNDX')
00557         IF (nval == 1) THEN
00558             ncvarnam_wndx = trim(adjustl(charval(nval)))
00559         ELSE
00560             IF (trim(adjustl(ncvarnam_wndx)) == "") THEN
00561                 ncvarnam_wndx = trim(adjustl(charval(nval)))
00562             END IF
00563         END IF
00564
00565     !----- CASE
00566     CASE ('NCVARNAM_WNDY')
00567         IF (nval == 1) THEN
00568             ncvarnam_wndy = trim(adjustl(charval(nval)))
00569         ELSE
00570             IF (trim(adjustl(ncvarnam_wndy)) == "") THEN
00571                 ncvarnam_wndy = trim(adjustl(charval(nval)))
00572             END IF
00573         END IF
00574
00575     !----- CASE
00576     CASE ('NCSHUFFLE')
00577         npnts = loadintvar(nval, realval, 1, ivalout)
00578         ncshuffle = ivalout(1)
00579
00580     !----- CASE
00581     CASE ('NCDEFLATE')
00582         npnts = loadintvar(nval, realval, 1, ivalout)
00583         ncdeflate = ivalout(1)
00584
00585     !----- CASE
00586     CASE ('NCDLEVEL')
00587         npnts = loadintvar(nval, realval, 1, ivalout)
00588         ncdlevel = ivalout(1)
00589
00590     !----- CASE
00591     CASE ('MODELTYPE')
00592         npnts = loadintvar(nval, realval, 1, ivalout)
00593         modeltype = ivalout(1)
00594
00595     !----- CASE

```

```

00596         CASE DEFAULT
00597             ! Do nothing
00598         END SELECT
00599     END IF
00600 END DO
00601
00602 10 WRITE(lun_screen, '( "Error while processing line: ", a, " in file: ", a )' ) &
00603     trim(adjustl(inpline)), trim(tmpfilename)
00604
00605 CLOSE(iunit)
00606 stop
00607
00608 20 CLOSE(iunit)
00609
00610 WRITE(lun_screen, '(a)') 'Finished processing the input fields from the control file ...'
00611
00612 !-----
00613 !--- CHECK INPUT VARIABLES AND SET DEFAULTS
00614 !-----
00615 CALL initlogging()
00616
00617 IF (checkcontrolfileinputs() /= 0) THEN
00618     WRITE(scratchmessage, '(a)') &
00619         'Errors found while processing the input variables. Check the log file for details.'
00620     CALL screenmessage(error, scratchmessage)
00621     CALL unsetmessagesource()
00622     CALL terminate()
00623 END IF
00624
00625 CALL printmodelparams()
00626
00627 CALL unsetmessagesource()
00628
00629 END SUBROUTINE readcontrolfile
00630
00631 !=====
00632 !-----
00633 ! SUBROUTINE PRINT MODEL PARAMS
00634 !-----
00635 !-----
00636 !-----
00644 SUBROUTINE printmodelparams()
00645
00646     USE pahm_global
00647
00648     IMPLICIT NONE
00649
00650     CHARACTER(LEN=128) :: tmpStr
00651     INTEGER             :: i
00652
00653     IF (writeparams) THEN
00654         print *, "
00655         WRITE(*, '(a)') '----- MODEL PARAMETERS -----'
00656
00657         WRITE(*, '(a, a)') ' title = ', trim(adjustl(title))
00658
00659         DO i = 1, nbtrfiles
00660             WRITE(*, '(a, "(, i1, ") ", a)') ' bestTrackFileName', i, " = " //
00661             trim(adjustl(besttrackfilename(i)))
00662             END DO
00663             WRITE(*, '(a, a)') ' meshFileType = ', trim(adjustl(meshfiletype))
00664             WRITE(*, '(a, a)') ' meshFileName = ', trim(adjustl(meshfilename))
00665             WRITE(*, '(a, a)') ' meshFileForm = ', trim(adjustl(meshfileform))
00666
00667             print *, "
00668             WRITE(tmpstr, '(f20.5)') gravity
00669             WRITE(*, '(a, a)') ' gravity = ', trim(adjustl(tmpstr)) // " m/s^2"
00670             WRITE(tmpstr, '(f20.5)') rhowater
00671             WRITE(*, '(a, a)') ' rhoWater = ', trim(adjustl(tmpstr)) // " kg/m^3"
00672             WRITE(tmpstr, '(f20.5)') rhoair
00673             WRITE(*, '(a, a)') ' rhoAir = ', trim(adjustl(tmpstr)) // " kg/m^3"
00674             WRITE(tmpstr, '(f20.5)') backgroundatmpress
00675             WRITE(*, '(a, a)') ' backgroundAtmPress = ', trim(adjustl(tmpstr)) // " mbar"
00676             WRITE(tmpstr, '(f20.2)') bladjustfac
00677             WRITE(*, '(a, a)') ' blAdjustFac = ', trim(adjustl(tmpstr))
00678
00679             print *, "
00680             WRITE(*, '(a, a)') ' refDateTime = ', trim(adjustl(refdatettime))
00681             WRITE(*, '(a, i4.4)') ' refYear = ', refyear
00682             WRITE(*, '(a, i2.2)') ' refMonth = ', refmonth
00683             WRITE(*, '(a, i2.2)') ' refDay = ', refday

```

```

00683      WRITE(*, '(a, i2.2)') '    refHour          = ', refhour
00684      WRITE(*, '(a, i2.2)') '    refMin           = ', refmin
00685      WRITE(*, '(a, i2.2)') '    refSec           = ', refsec
00686      WRITE(*, '(a, i1)') '    refDateSpecified   = ', refdatespecified
00687
00688      print *, "
00689      WRITE(*, '(a, a)') '    begDateTime       = ', trim(adjustl(begdatetime))
00690      WRITE(*, '(a, i4.4)') '    begYear          = ', begyear
00691      WRITE(*, '(a, i2.2)') '    begMonth         = ', begmonth
00692      WRITE(*, '(a, i2.2)') '    begDay           = ', begday
00693      WRITE(*, '(a, i2.2)') '    begHour          = ', beghour
00694      WRITE(*, '(a, i2.2)') '    begMin           = ', begmin
00695      WRITE(*, '(a, i2.2)') '    begSec           = ', begsec
00696      WRITE(*, '(a, i1)') '    begDateSpecified   = ', begdatespecified
00697
00698      print *, "
00699      WRITE(*, '(a, a)') '    endDateTime      = ', trim(adjustl(enddatetime))
00700      WRITE(*, '(a, i4.4)') '    endYear          = ', endyear
00701      WRITE(*, '(a, i2.2)') '    endMonth         = ', endmonth
00702      WRITE(*, '(a, i2.2)') '    endDay           = ', endday
00703      WRITE(*, '(a, i2.2)') '    endHour          = ', endhour
00704      WRITE(*, '(a, i2.2)') '    endMin           = ', endmin
00705      WRITE(*, '(a, i2.2)') '    endSec           = ', endsec
00706      WRITE(*, '(a, i1)') '    endDateSpecified   = ', enddatespecified
00707
00708      print *, "
00709      WRITE(*, '(a, a1)') '    unitTime          = ', unittime
00710      WRITE(tmpstr, '(f20.5)') outdt
00711      WRITE(*, '(a, a)') '    outDT              = ', trim(adjustl(tmpstr)) // " " //
tolowercase(trim(unittime))
00712      WRITE(tmpstr, '(f20.5)') moutdt
00713      WRITE(*, '(a, a)') '    mdOutDT           = ', trim(adjustl(tmpstr)) // " s"
00714      WRITE(tmpstr, '(f20.5)') begsimtime
00715      WRITE(*, '(a, a)') '    begSimTime        = ', trim(adjustl(tmpstr)) // " " //
tolowercase(trim(unittime))
00716      WRITE(tmpstr, '(f20.5)') mdbegsimtime
00717      WRITE(*, '(a, a)') '    mdBegSimTime      = ', trim(adjustl(tmpstr)) // " s"
00718      WRITE(*, '(a, i1)') '    begSimSpecified   = ', begsimspecified
00719      WRITE(tmpstr, '(f20.5)') endsimtime
00720      WRITE(*, '(a, a)') '    endSimTime        = ', trim(adjustl(tmpstr)) // " " //
tolowercase(trim(unittime))
00721      WRITE(tmpstr, '(f20.5)') mdendsimtime
00722      WRITE(*, '(a, a)') '    mdEndSimTime      = ', trim(adjustl(tmpstr)) // " s"
00723      WRITE(*, '(a, i1)') '    endSimSpecified   = ', endsimspecified
00724      WRITE(tmpstr, '(i10)') noutdt
00725      WRITE(*, '(a, a)') '    nOutDT              = ', trim(adjustl(tmpstr))
00726
00727      print *, "
00728      WRITE(*, '(a, a)') '    outFileNames     = ', trim(adjustl(outfilename))
00729      WRITE(*, '(a, i1)') '    ncShuffle         = ', ncshuffle
00730      WRITE(*, '(a, i1)') '    ncDeflate         = ', ncdeflate
00731      WRITE(*, '(a, i1)') '    ncDLevel          = ', nclevel
00732      WRITE(*, '(a, a)') '    ncVarNam_Pres     = ', trim(ncvarnam_pres)
00733      WRITE(*, '(a, a)') '    ncVarNam_WndX     = ', trim(ncvarnam_wndx)
00734      WRITE(*, '(a, a)') '    ncVarNam_WndY     = ', trim(ncvarnam_wndy)
00735
00736      print *, "
00737      WRITE(*, '(a, i1)') '    modelType         = ', modeltype
00738
00739      WRITE(*, '(a)') '----- MODEL PARAMETERS -----'
00740      print *, "
00741      END IF
00742
00743      END SUBROUTINE printmodelparams
00744
00745      !=====
00746
00747      !-----
00748      ! FUNCTION GET LINE RECORD
00749      !-----
00750      !-----
00773      INTEGER FUNCTION getlinerecord(inpLine, outLine, lastCommFlag) RESULT(myLen)
00774
00775      IMPLICIT NONE
00776
00777      ! Imported variable declarations.
00778      CHARACTER(LEN=*) , INTENT(IN) :: inpline
00779      CHARACTER(LEN=LEN(inpLine)) , INTENT(OUT) :: outline
00780      INTEGER, OPTIONAL :: lastcommflag
00781
00782      ! Local variable declarations.

```

```

00783 CHARACTER(LEN=LEN(inpLine)) :: line
00784 CHARACTER :: tmpinpline(len(inpLine))
00785 INTEGER :: lenline, commflag, icomm
00786
00787 ! Table of some ASCII character symbols
00788 !   CHAR   ASCII
00789 !   TAB     9
00790 !   SPC    32
00791 !   !      33
00792 !   #      35
00793 !   *      42
00794 !   +      43
00795 !   -      45
00796 !   /      47
00797 !   :      58
00798 !   =      61
00799 !   \      92
00800 !   |     124
00801
00802 mylen      = 0
00803 outline    = blank
00804 tmpinpline = blank
00805
00806 commflag = 0
00807 IF (PRESENT(lastcommflag)) THEN
00808   IF (lastcommflag <= 0) commflag = 0
00809   IF (lastcommflag > 0) commflag = 1
00810 END IF
00811
00812 tmpinpline = transfer(inpLine, tmpinpline)
00813 tmpinpline = pack(tmpinpline, tmpinpline /= achar(9), spread(' ', 1, len(inpLine)))
00814
00815 line      = trim(adjustl(transfer(tmpinpline, line)))
00816 lenline   = len_trim(line)
00817
00818 IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
00819   IF (commflag > 0) THEN
00820     icomm = index(line, char(33), back = .false.)
00821     IF (icomm == 0) icomm = index(line, char(35), back = .false.)
00822     IF (icomm > 0) lenline = icomm - 1
00823
00824     line = trim(adjustl(line(1:lenline)))
00825     lenline = len_trim(line)
00826   END IF
00827
00828   outline = line
00829   mylen   = lenline
00830 END IF
00831
00832 RETURN
00833
00834 END FUNCTION getlinerecord
00835
00836 !=====
00837 !-----
00838 ! FUNCTION PARSE LINE
00839 !-----
00840 !-----
00841
00869 INTEGER FUNCTION parseline(inpLine, outLine, keyWord, nVal, cVal, rVal) RESULT(myStatus)
00870
00871 IMPLICIT NONE
00872
00873 ! Imported variable declarations.
00874 CHARACTER(LEN=*) , INTENT(IN) :: inpline
00875 CHARACTER(LEN=LEN(inpLine)) , INTENT(OUT) :: outline
00876 CHARACTER(LEN=40) , INTENT(INOUT) :: keyword
00877 INTEGER , INTENT(INOUT) :: nval
00878 CHARACTER(LEN=512) , DIMENSION(200) , INTENT(INOUT) :: cval
00879 REAL(siz) , DIMENSION(200) , INTENT(INOUT) :: rval
00880
00881 ! Local variable declarations
00882 LOGICAL :: isstring, kextract, decflag, nested
00883 INTEGER :: iblank, icont, ipipe, kstr, kend
00884 INTEGER :: lend, lens, lstr, lval, nmul, schar
00885 INTEGER :: copies, i, ic, ie, ierr, is, j, status
00886 INTEGER, DIMENSION(20) :: imul
00887 CHARACTER(LEN=256) :: vstring, string
00888 CHARACTER(LEN=LEN(inpLine)) :: line
00889 INTEGER :: lenline
00890

```

```

00891      ! Table of some ASCII character symbols
00892      !   CHAR      ASCII
00893      !   TAB        9
00894      !   SPC       32
00895      !   !         33
00896      !   #         35
00897      !   *         42
00898      !   +         43
00899      !   -         45
00900      !   /         47
00901      !   :         58
00902      !   =         61
00903      !   \         92
00904      !   |        124
00905
00906      ! Initialize.
00907      line           = blank
00908      vstring        = blank
00909      string         = blank
00910
00911      lenline = getlinerecord(inpline, line, 1)
00912      outline = line
00913
00914      ! If not a blank or comment line [CHAR(33)=!], decode and extract input
00915      ! values. Find equal sign [CHAR(61)].
00916      status = -1
00917      nested = .false.
00918      IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
00919          status = 1
00920          kstr = 1
00921          kend = index(line, char(61), back = .false.) - 1
00922          lstr = index(line, char(61), back = .true.) + 1
00923
00924          ! Determine if KEYWORD is followed by double equal sign (==) indicating
00925          ! nested parameter.
00926          IF ((lstr - kend) == 3) nested = .true.
00927
00928          ! Extract KEYWORD, trim leading and trailing blanks.
00929          kextract = .false.
00930          IF (kend > 0) THEN
00931              lend = lenline
00932              keyword = line(kstr:kend)
00933              nval = 0
00934              kextract = .true.
00935          ELSE
00936              lstr = 1
00937              lend = lenline
00938              kextract = .true.
00939          END IF
00940
00941          ! Extract parameter values string. Remove continuation symbol
00942          ! [CHAR(92)=\] or multi-line value [CHAR(124)=|], if any. Trim
00943          ! leading trailing blanks.
00944          IF (kextract) THEN
00945              icon = index(line, char(92), back = .false.)
00946              ipipe = index(line, char(124), back = .false.)
00947              IF (icon > 0) lend = icon - 1
00948              IF (ipipe > 0) lend = ipipe - 1
00949              vstring = adjustl(line(lstr:lend))
00950              lval = len_trim(vstring)
00951
00952              ! The PROGRAM, VERSION and TITLE KEYWORDS are special ones because
00953              ! they can include strings, numbers, spaces, and continuation symbol.
00954              isstring = .false.
00955              SELECT CASE (toupper(trim(keyword)))
00956              CASE ('TITLE')
00957                  nval = nval + 1
00958                  cval(nval) = vstring(1:lval)
00959                  isstring = .true.
00960
00961              CASE ('LOGFILENAME')
00962                  nval = nval + 1
00963                  cval(nval) = vstring(1:lval)
00964                  isstring = .true.
00965
00966              CASE ('BESTTRACKFILENAME')
00967                  nval = nval + 1
00968                  cval(nval) = vstring(1:lval)
00969                  isstring = .true.
00970
00971              CASE ('MESHFILENAME')

```

```

00972         nval = nval + 1
00973         cval(nval) = vstring(1:lval)
00974         isstring = .true.
00975
00976         CASE ('MESHFILETYPE')
00977             nval = nval + 1
00978             cval(nval) = vstring(1:lval)
00979             isstring = .true.
00980
00981         CASE ('MESHFILEFORM')
00982             nval = nval + 1
00983             cval(nval) = vstring(1:lval)
00984             isstring = .true.
00985
00986         CASE ('REFDATETIME')
00987             nval = nval + 1
00988             cval(nval) = vstring(1:lval)
00989             isstring = .true.
00990
00991         CASE ('UNITTIME')
00992             nval = nval + 1
00993             cval(nval) = vstring(1:lval)
00994             isstring = .true.
00995
00996         CASE ('BEGDATETIME')
00997             nval = nval + 1
00998             cval(nval) = vstring(1:lval)
00999             isstring = .true.
01000
01001         CASE ('ENDDATETIME')
01002             nval = nval + 1
01003             cval(nval) = vstring(1:lval)
01004             isstring = .true.
01005
01006         CASE ('OUTFILENAME')
01007             nval = nval + 1
01008             cval(nval) = vstring(1:lval)
01009             isstring = .true.
01010
01011         CASE ('NCVARNAM_PRES')
01012             nval = nval + 1
01013             cval(nval) = vstring(1:lval)
01014             isstring = .true.
01015
01016         CASE ('NCVARNAM_WNDX')
01017             nval = nval + 1
01018             cval(nval) = vstring(1:lval)
01019             isstring = .true.
01020
01021         CASE ('NCVARNAM_WNDY')
01022             nval = nval + 1
01023             cval(nval) = vstring(1:lval)
01024             isstring = .true.
01025
01026         CASE DEFAULT
01027             ! For every other KEYWORD except the above.
01028             ! Check if there is a multiplication symbol [CHAR(42)=*] in the variable
01029             ! string indicating repetition of input values.
01030             nmul = 0
01031             DO i = 1, lval
01032                 IF (vstring(i:i) == char(42)) THEN
01033                     nmul = nmul + 1
01034                     imul(nmul) = i
01035                 END IF
01036             END DO
01037             ic = 1
01038
01039             ! Check for blank spaces [CHAR(32)=' '] between entries and decode.
01040             is = 1
01041             ie = lval
01042             iblank = 0
01043             decflag = .false.
01044             DO i = 1, lval
01045                 IF (vstring(i:i) == char(32)) THEN
01046                     IF (vstring(i + 1:i + 1) /= char(32)) decflag = .true.
01047                     iblank = i
01048                 ELSE
01049                     ie = i
01050                 END IF
01051                 IF (decflag .OR. (i == lval)) THEN
01052                     nval = nval + 1

```

```

01053
01054      ! Processing numeric values. Check starting character to determine
01055      ! if numeric or character values. It is possible to have both when
01056      ! processing repetitions via the multiplication symbol.
01057      schar = ichar(vstring(is:is))
01058      IF ((48 <= schar) .AND. (schar <= 57)) .OR. (schar == 43) .OR. (schar == 45)) THEN
01059      IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
01060          READ(vstring(is:imul(ic) - 1), *) copies
01061          schar = ichar(vstring(imul(ic) + 1:imul(ic) + 1))
01062          IF ((43 <= schar) .AND. (schar <= 57)) THEN
01063              READ(vstring(imul(ic) + 1:ie), *) rval(nval)
01064              DO j = 1, copies - 1
01065                  rval(nval + j) = rval(nval)
01066              END DO
01067          ELSE
01068              string = vstring(imul(ic) + 1:ie)
01069              lens = len_trim(string)
01070              cval(nval) = string(1:lens)
01071              DO j = 1, copies - 1
01072                  cval(nval + j) = cval(nval)
01073              END DO
01074          END IF
01075          nval = nval + copies - 1
01076          ic = ic + 1
01077      ELSE
01078          string = vstring(is:ie)
01079          lens = len_trim(string)
01080          !READ(string(1:lens), *) rVal(nVal)
01081          READ(string(1:lens), *, iostat=ierr) rval(nval)
01082          IF (ierr /= 0) THEN
01083              WRITE(*, *) '#### ERROR :: Cannot interpret string ', string(1:lens), &
01084                  ' as a REAL number.'
01085          END IF
01086      END IF
01087      ELSE
01088
01089      ! Processing character values (logicals and strings).
01090      IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
01091          READ(vstring(is:imul(ic) - 1), *) copies
01092          cval(nval) = vstring(imul(ic) + 1:ie)
01093          DO j = 1, copies - 1
01094              cval(nval + j) = cval(nval)
01095          END DO
01096          nval = nval + copies - 1
01097          ic = ic + 1
01098      ELSE
01099          string = vstring(is:ie)
01100          cval(nval) = trim(adjustl(string))
01101      END IF
01102      isstring = .true.
01103      END IF
01104      is = iblank + 1
01105      ie = lval
01106      decflag = .false.
01107      END IF
01108      END DO
01109      END SELECT ! keyWord
01110      END IF ! kExtract
01111      status = nval
01112      END IF
01113
01114      mystatus = status
01115
01116      RETURN
01117
01118      END FUNCTION parseline
01119
01120      !=====
01121
01122      !-----
01123      !      SUBROUTINE CHECK CONTROL FILE INPUTS
01124      !-----
01125      !-----
01126
01140      INTEGER FUNCTION checkcontrolfileinputs() RESULT(errStatus)
01141
01142      USE pahm_global
01143      USE timedateutils, ONLY : firstgregdate, firstgregtime, &
01144                              gregtojulday, joindate, gettimeconvsec
01145
01146      IMPLICIT NONE
01147

```



```

01148      ! Local variables
01149      INTEGER                                :: errio, errnum
01150      INTEGER                                :: icnt
01151      LOGICAL                                :: filefound
01152      REAL(sz)                               :: gregjd, refjd, jd0, jd1
01153      REAL(sz)                               :: timesec, timeconvfac
01154      CHARACTER(LEN=64)                     :: tmpstr, tmpstr1, tmpstr2
01155
01156
01157      !----- Initialize variables
01158      errio      = 0
01159      errnum     = 0
01160      errstatus  = 0
01161      scratchmessage = blank
01162
01163
01164      CALL setmessagesource("CheckControlFileInputs")
01165
01166      !----- 1) Best track files (mandatory variables) -----
01167      IF (nbtrfiles <= 0) THEN
01168          errnum = 1
01169
01170          WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01171              ' . Invalid value supplied for dimension parameter: nBTrFiles = ', &
01172              nbtrfiles, ' (should be greater than zero).'
01173          CALL logmessage(error, scratchmessage)
01174      ELSE IF (besttrackfilenamespecified) THEN
01175          IF (numbtfles /= nbtrfiles) THEN
01176              errnum = 2
01177
01178              WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01179              ' . The number of files for <bestTrackFileName> should be equal to nBTrFiles:
01180              ', &
01181              nbtrfiles, ' .'
01182          CALL logmessage(error, scratchmessage)
01183      END IF
01184
01185      DO icnt = 1, numbtfles
01186          INQUIRE(file=trim(adjustl(besttrackfilename(icnt))), iostat=errio, exist=filefound)
01187          IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01188              errnum = 3
01189
01190              WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01191              ' . Could not access the best track file for read: ' // &
01192              trim(adjustl(besttrackfilename(icnt)))
01193          CALL logmessage(error, scratchmessage)
01194      END DO
01195      ELSE
01196          errnum = 4
01197
01198          WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01199          ' . bestTrackFileName(s) not specified. This is a mandatory variable. '
01200          CALL logmessage(error, scratchmessage)
01201      END IF
01202
01203      !----- 2) Mesh file (mandatory variables) -----
01204      IF (.NOT. meshfilenamespecified) THEN
01205          errnum = 5
01206
01207          WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01208          ' . Parameter <meshFileName> is not specified (mandatory variable) '
01209          CALL logmessage(error, scratchmessage)
01210      ELSE
01211          meshfilename = adjustl(meshfilename)
01212          INQUIRE(file=trim(meshfilename), iostat=errio, exist=filefound)
01213          IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01214              errnum = 6
01215
01216              WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01217              ' . Could not access the mesh file for read: ' // trim(meshfilename)
01218          CALL logmessage(error, scratchmessage)
01219      END IF
01220      END IF
01221
01222      ! Check for meshFileType
01223      meshfiletype = toupper(trim(adjustl(meshfiletype)))
01224      SELECT CASE (trim(meshfiletype))
01225          !CASE ('ADCIRC', 'SCHISM', 'FVCOM', 'ROMS', 'GENERIC')
01226          CASE ('ADCIRC', 'SCHISM')
01227              ! These are the valid values

```

```

01228
01229     CASE DEFAULT
01230         WRITE(scratchmessage, '(a)') 'This file type is not supported: meshFileType = ' //
trim(meshfiletype)
01231         CALL logmessage(info, scratchmessage)
01232
01233         meshfiletype = 'ADCIRC'
01234
01235         WRITE(scratchmessage, '(a)') 'This value of meshFileType is adjusted to: meshFileType = ' //
trim(meshfiletype)
01236         CALL logmessage(info, scratchmessage)
01237     END SELECT
01238
01239     ! Check for meshFileForm
01240     meshfileform = toupper(trim(adjustl(meshfileform)))
01241     SELECT CASE (trim(meshfileform))
01242     !CASE ('ASCII', 'NETCDF')
01243     CASE ('ASCII')
01244         ! These are valid values
01245
01246     CASE DEFAULT
01247         WRITE(scratchmessage, '(a)') 'This file format is not supported: meshFileForm = ' //
trim(meshfileform)
01248         CALL logmessage(info, scratchmessage)
01249
01250         meshfileform = 'ASCII'
01251
01252         WRITE(scratchmessage, '(a)') 'This value of meshFileForm is adjusted to: meshFileForm = ' //
trim(meshfileform)
01253         CALL logmessage(info, scratchmessage)
01254     END SELECT
01255
01256     !----- 3) Reference date and time (mandatory variables) -----
01257     gregjd = gregtojulday(firstgregdate, firstgregtime)
01258     refjd = gregtojulday(refdate, reftime)
01259     IF (refjd < gregjd) THEN
01260         errnum = 7
01261         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01262             '. Invalid DateTime string was supplied: refDateTime = ' // trim(refdatetime)
01263         CALL logmessage(error, scratchmessage)
01264     END IF
01265
01266     !----- 4) Stepping parameters (mandatory variables) -----
01267     ! check for valid start time
01268     IF (begsimspecified) THEN
01269         IF (refjd + (mdbegsimtime * gettimeconvsec('D', 1)) < gregjd) THEN
01270             errnum = 8
01271             WRITE(tmpstr, '(f20.5)') begsimtime
01272             WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01273                 '. Invalid start time in reference to refDateTime was supplied: begSimTime =
' // &
01274                 trim(adjustl(tmpstr))
01275             CALL logmessage(error, scratchmessage)
01276         END IF
01277     ELSE
01278         errnum = 81
01279         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01280             '. Neither "begDateTime" or "begSimTime" are defined properly'
01281         CALL logmessage(error, scratchmessage)
01282     END IF
01283
01284     ! check for valid stop time
01285     IF (endsimspecified) THEN
01286         IF (compare_reals(endsimtime, begsimtime, closetol) <= 0) THEN
01287             errnum = 9
01288             WRITE(tmpstr1, '(f20.5)') begsimtime
01289             WRITE(tmpstr2, '(f20.5)') endsimtime
01290             WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01291                 '. Stop time should be greater than start time: begSimTime = ' // &
01292                 trim(adjustl(tmpstr1)) // ', endSimTime = ' // trim(adjustl(tmpstr2))
01293             CALL logmessage(error, scratchmessage)
01294         END IF
01295     ELSE
01296         errnum = 91
01297         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01298             '. Neither "endDateTime" or "endSimTime" are defined properly'
01299         CALL logmessage(error, scratchmessage)
01300     END IF
01301
01302     ! check for valid outDT; (endSimTime - begSimTime) should be an integral multiple of outDT
01303     IF (outdt <= 0) THEN

```

```

01304     WRITE(tmpstr, '(f20.5)') outdt
01305     WRITE(scratchmessage, '(a)') 'Frequency of output data should be greater than zero: outDT = ' // &
01306           trim(adjustl(tmpstr))
01307     CALL logmessage(info, scratchmessage)
01308
01309     mdoutdt = 3600.0
01310     outdt = fixnearwholereal(mdoutdt * gettimeconvsec(unittime, 1), closetol)
01311
01312     WRITE(tmpstr, '(f20.5)') outdt
01313     WRITE(scratchmessage, '(a)') 'The outDT value is adjusted to: outDT = ' // trim(adjustl(tmpstr))
01314     CALL logmessage(info, scratchmessage)
01315   END IF
01316
01317   jd0 = refjd + (mdbegsimtime * gettimeconvsec('D', 1))
01318   jd1 = refjd + (mdendsimtime * gettimeconvsec('D', 1))
01319   timesec = fixnearwholereal((jd1 - jd0) * gettimeconvsec('D'), closetol)
01320   IF ((timesec < mdoutdt) .OR. comparereals(modulo(timesec, mdoutdt), 0.0_sz) /= 0) THEN
01321     errnum = 10
01322
01323     WRITE(tmpstr1, '(f20.5)') timesec
01324     WRITE(tmpstr2, '(f20.5)') outdt
01325     WRITE(scratchmessage, '(a)') errnum, &
01326           'The value of (endSimTime - begSimTime) = ' // trim(adjustl(tmpstr1)) // &
01327           ' should be an integral multiple of outDT = ' // trim(adjustl(tmpstr2))
01328     CALL logmessage(error, scratchmessage)
01329   ELSE
01330     noutdt = int(timesec / mdoutdt) + 1
01331   END IF
01332
01333   !----- 4) outFileName (mandatory variable) -----
01334   outfilename = adjustl(outfilename)
01335   IF (.NOT. outfilenamespecified) THEN
01336     errnum = 11
01337
01338     WRITE(scratchmessage, '(a)') errnum, &
01339           'Output filename is not specified: outFileName = ' // trim(outfilename)
01340     CALL logmessage(error, scratchmessage)
01341   END IF
01342
01343   !----- 5) NetCDF variables ncShuffle, ncDeflate, ncDLevel and others -----
01344   IF (ncshuffle <= 0) THEN
01345     ncshuffle = 0
01346   ELSE
01347     ncshuffle = 1
01348   END IF
01349
01350   IF (ncdeflate <= 0) THEN
01351     ncdeflate = 0
01352   ELSE
01353     ncdeflate = 1
01354   END IF
01355
01356   IF (ncdlevel <= 0) THEN
01357     ncdlevel = 0
01358   ELSE
01359     IF (ncdlevel > 9) ncdlevel = 9
01360   END IF
01361
01362   ncvarnam_pres = trim(adjustl(ncvarnam_pres))
01363   IF (len_trim(ncvarnam_pres) == 0) ncvarnam_pres = trim(adjustl(def_ncnam_pres))
01364   ncvarnam_wndx = trim(adjustl(ncvarnam_wndx))
01365   IF (len_trim(ncvarnam_wndx) == 0) ncvarnam_wndx = trim(adjustl(def_ncnam_wndx))
01366   ncvarnam_wndy = trim(adjustl(ncvarnam_wndy))
01367   IF (len_trim(ncvarnam_wndy) == 0) ncvarnam_wndy = trim(adjustl(def_ncnam_wndy))
01368
01369   !----- 5) modelType (mandatory variable) -----
01370   SELECT CASE (modeltype)
01371   !CASE (1, 2, 3, 4)
01372   CASE (1)
01373     ! These are all valid values
01374
01375   CASE DEFAULT
01376     errnum = 12
01377
01378     WRITE(scratchmessage, '(a)') errnum, &
01379           'This model type is not supported: modelType = ', modeltype
01380     CALL logmessage(error, scratchmessage)
01381   END SELECT
01382
01383   !----- 6) various physical parameters -----
01384   IF ((gravity < 9.76) .OR. (gravity > 9.83)) THEN

```

```

01385     WRITE(tmpstr1, '(f20.5, a)') gravity
01386     tmpstr1 = trim(tmpstr1) // ' m/s^2'
01387     WRITE(tmpstr2, '(f20.5, a)') defv_gravity
01388     tmpstr2 = trim(tmpstr2) // ' m/s^2'
01389     WRITE(scratchmessage, '(a)') 'The value of gravity = ' // trim(adjustl(tmpstr1)) // &
01390     ' is adjusted to: gravity = ' // trim(adjustl(tmpstr2))
01391
01392     CALL logmessage(info, scratchmessage)
01393
01394     gravity = defv_gravity
01395 END IF
01396
01397 IF ((rhowater < 992.0) .OR. (rhowater > 1029.0)) THEN
01398     WRITE(tmpstr1, '(f20.5, a)') rhowater
01399     tmpstr1 = trim(tmpstr1) // ' kg/m^3'
01400     WRITE(tmpstr2, '(f20.5, a)') defv_rhowater
01401     tmpstr2 = trim(tmpstr2) // ' kg/m^3'
01402     WRITE(scratchmessage, '(a)') 'The value of rhoWater = ' // trim(adjustl(tmpstr1)) // &
01403     ' is adjusted to: rhoWater = ' // trim(adjustl(tmpstr2))
01404
01405     CALL logmessage(info, scratchmessage)
01406
01407     rhowater = defv_rhowater
01408 END IF
01409
01410 IF ((rhoair < 1.0) .OR. (rhoair > 1.3)) THEN
01411     WRITE(tmpstr1, '(f20.5, a)') rhoair
01412     tmpstr1 = trim(tmpstr1) // ' kg/m^3'
01413     WRITE(tmpstr2, '(f20.5, a)') defv_rhoair
01414     tmpstr2 = trim(tmpstr2) // ' kg/m^3'
01415     WRITE(scratchmessage, '(a)') 'The value of rhoAir = ' // trim(adjustl(tmpstr1)) // &
01416     ' is adjusted to: rhoAir = ' // trim(adjustl(tmpstr2))
01417
01418     CALL logmessage(info, scratchmessage)
01419
01420     rhoair = defv_rhoair
01421 END IF
01422
01423 IF ((backgroundatmpress < 900.0) .OR. (backgroundatmpress > 1025.0)) THEN
01424     WRITE(tmpstr1, '(f20.5, a)') backgroundatmpress
01425     tmpstr1 = trim(tmpstr1) // ' mb'
01426     WRITE(tmpstr2, '(f20.5, a)') defv_atmpress
01427     tmpstr2 = trim(tmpstr2) // ' mb'
01428     WRITE(scratchmessage, '(a)') 'The value of backgroundAtmPress = ' // trim(adjustl(tmpstr1)) // &
01429     ' is adjusted to: backgroundAtmPress = ' // trim(adjustl(tmpstr2))
01430
01431     CALL logmessage(info, scratchmessage)
01432
01433     backgroundatmpress = defv_atmpress
01434 END IF
01435
01436 IF ((bladjustfac < 0.65) .OR. (bladjustfac > 1.0)) THEN
01437     WRITE(tmpstr1, '(f20.5)') bladjustfac
01438     WRITE(tmpstr2, '(f20.5)') defv_bladjustfac
01439     WRITE(scratchmessage, '(a)') 'The value of blAdjustFac = ' // trim(adjustl(tmpstr1)) // &
01440     ' is adjusted to: blAdjustFac = ' // trim(adjustl(tmpstr2))
01441
01442     CALL logmessage(info, scratchmessage)
01443
01444     bladjustfac = defv_bladjustfac
01445 END IF
01446
01447     errstatus = errnum
01448
01449 END FUNCTION checkcontrolfileinputs
01450
01451 !=====
01452
01453 !-----
01454 ! FUNCTION LOAD INT VAR
01455 !-----
01456 !-----
01457
01478 INTEGER FUNCTION loadintvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01479
01480     IMPLICIT NONE
01481
01482     INTEGER, INTENT(IN) :: ninp, nout
01483     REAL(s2), INTENT(IN) :: vinp(ninp)
01484     INTEGER, INTENT(OUT) :: vout(nout)
01485
01486     INTEGER :: i, ic

```

```

01487
01488 !-----
01489 ! Load INTEGER variable with input values.
01490 !-----
01491
01492 ! If not all values are provided for variable, assume the last value
01493 ! for the rest of the array.
01494 ic = 0
01495 IF (ninp <= nout) THEN
01496   DO i = 1, ninp
01497     ic = ic + 1
01498     vout(i) = int(vinp(i))
01499   END DO
01500   DO i = ninp + 1, nout
01501     ic = ic + 1
01502     vout(i) = int(vinp(ninp))
01503   END DO
01504 ELSE
01505   DO i = 1, nout
01506     ic = ic + 1
01507     vout(i) = int(vinp(i))
01508   END DO
01509 END IF
01510
01511 nvalsout = ic
01512
01513 RETURN
01514
01515 END FUNCTION loadintvar
01516
01517 !-----
01518
01519 !-----
01520 ! FUNCTION LOAD LOG VAR
01521 !-----
01543 !-----
01544 INTEGER FUNCTION loadlogvar (nInp, vInp, nOut, vOut) RESULT(nValsOut)
01545
01546 IMPLICIT NONE
01547
01548 INTEGER, INTENT(IN) :: ninp, nout
01549 CHARACTER(LEN=*), INTENT(IN) :: vinp(ninp)
01550 LOGICAL, INTENT(OUT) :: vout(nout)
01551
01552 INTEGER :: i, ic
01553
01554 !-----
01555 ! Load INTEGER variable with input values.
01556 !-----
01557
01558 ! If not all values are provided for variable, assume the last value
01559 ! for the rest of the array.
01560 ic = 0
01561 IF (ninp <= nout) THEN
01562   DO i = 1, ninp
01563     ic = ic + 1
01564     IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01565       vout(i) = .true.
01566     ELSE
01567       vout(i) = .false.
01568     END IF
01569   END DO
01570   DO i = ninp + 1, nout
01571     ic = ic + 1
01572     IF ((vinp(ninp)(1:1) == 'T') .OR. (vinp(ninp)(1:1) == 't')) THEN
01573       vout(i) = .true.
01574     ELSE
01575       vout(i) = .false.
01576     END IF
01577   END DO
01578 ELSE
01579   DO i = 1, nout
01580     ic = ic + 1
01581     IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01582       vout(i) = .true.
01583     ELSE
01584       vout(i) = .false.
01585     END IF
01586   END DO
01587 END IF
01588

```

```

01589     nvalsout = ic
01590
01591     RETURN
01592
01593 END FUNCTION loadlogvar
01594
01595 !=====
01596
01597 !-----
01598 ! FUNCTION LOAD REAL VAR
01599 !-----
01600 !-----
01601
01602 INTEGER FUNCTION loadrealvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01603
01604     IMPLICIT NONE
01605
01606     INTEGER, INTENT(IN)    :: ninp, nout
01607     REAL(sz), INTENT(IN)  :: vinp(ninp)
01608     REAL(sz), INTENT(OUT) :: vout(nout)
01609
01610     INTEGER                :: i, ic
01611
01612     !-----
01613     ! Load INTEGER variable with input values.
01614     !-----
01615
01616     ! If not all values are provided for variable, assume the last value
01617     ! for the rest of the array.
01618     ic = 0
01619     IF (ninp <= nout) THEN
01620         DO i = 1, ninp
01621             ic = ic + 1
01622             vout(i) = vinp(i)
01623         END DO
01624         DO i = ninp + 1, nout
01625             ic = ic + 1
01626             vout(i) = vinp(ninp)
01627         END DO
01628     ELSE
01629         DO i = 1, nout
01630             ic = ic + 1
01631             vout(i) = vinp(i)
01632         END DO
01633     END IF
01634
01635     nvalsout = ic
01636
01637     RETURN
01638
01639 END FUNCTION loadrealvar
01640
01641 !=====
01642
01643 !-----
01644 ! FUNCTION TO LOWER CASE
01645 !-----
01646 !-----
01647
01648 PURE FUNCTION tolowercase(inpString) RESULT(outString)
01649
01650     IMPLICIT NONE
01651
01652     CHARACTER(*), INTENT(IN) :: inpstring
01653
01654     INTEGER, PARAMETER      :: duc = ichar('A') - ichar('a')
01655     CHARACTER(LEN(inpString)) :: outstring
01656     CHARACTER               :: ch
01657     INTEGER                 :: i
01658
01659     DO i = 1, len(inpstring)
01660         ch = inpstring(i:i)
01661         IF ((ch >= 'A') .AND. (ch <= 'Z')) ch = char(ichar(ch) - duc)
01662         outstring(i:i) = ch
01663     END DO
01664
01665     RETURN
01666
01667 END FUNCTION tolowercase
01668
01669 !=====
01670
01671 !-----
01672
01673 !-----

```

```

01704 ! FUNCTION TO UPPER CASE
01705 !-----
01719 !-----
01720 PURE FUNCTION touppercase(inpString) RESULT(outString)
01721
01722     IMPLICIT NONE
01723
01724     CHARACTER(*), INTENT(IN) :: inpstring
01725
01726     INTEGER, PARAMETER      :: duc = ichar('A') - ichar('a')
01727     CHARACTER(LEN(inpString)) :: outstring
01728     CHARACTER               :: ch
01729     INTEGER                 :: i
01730
01731     DO i = 1, len(inpstring)
01732         ch = inpstring(i:i)
01733         IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
01734         outstring(i:i) = ch
01735     END DO
01736
01737     RETURN
01738
01739 END FUNCTION touppercase
01740
01741 !-----
01742
01743 !-----
01744 ! FUNCTION CONV LON
01745 !-----
01759 !-----
01760 REAL(sz) function convlon(inplon) result (myvalout)
01761
01762     IMPLICIT NONE
01763
01764     REAL(sz) :: inplon
01765
01766     myvalout = mod(inplon + 180.0_sz, 360.0_sz) - 180.0_sz
01767
01768     RETURN
01769
01770 END FUNCTION convlon
01771
01772 !-----
01773
01774 !-----
01775 ! SUBROUTINE GEO TO CPP SCALAR
01776 !-----
01799 !-----
01800 SUBROUTINE geotocpp_scalar(lat, lon, lat0, lon0, x, y)
01801
01802     USE pahm_global, ONLY : rearth, deg2rad
01803
01804     IMPLICIT NONE
01805
01806     REAL(SZ), INTENT(IN) :: lat
01807     REAL(SZ), INTENT(IN) :: lon
01808     REAL(SZ), INTENT(IN) :: lat0
01809     REAL(SZ), INTENT(IN) :: lon0
01810     REAL(SZ), INTENT(OUT) :: x
01811     REAL(SZ), INTENT(OUT) :: y
01812
01813     x = deg2rad * rearth * (lon - lon0) * cos(lat0)
01814     y = deg2rad * rearth * lat
01815
01816 END SUBROUTINE geotocpp_scalar
01817
01818 !-----
01819
01820 !-----
01821 ! SUBROUTINE GEO TO CPP 1D
01822 !-----
01846 !-----
01847 SUBROUTINE geotocpp_1d(lat, lon, lat0, lon0, x, y)
01848
01849     USE pahm_global, ONLY : rearth, deg2rad
01850
01851     IMPLICIT NONE
01852
01853     REAL(SZ), INTENT(IN) :: lat(:)
01854     REAL(SZ), INTENT(IN) :: lon(:)
01855     REAL(SZ), INTENT(IN) :: lat0

```

```

01856     REAL(SZ), INTENT(IN)  :: lon0
01857     REAL(SZ), INTENT(OUT) :: x(:)
01858     REAL(SZ), INTENT(OUT) :: y(:)
01859
01860     x = deg2rad * rearth * (lon - lon0) * cos(lat0)
01861     y = deg2rad * rearth * lat
01862
01863     END SUBROUTINE geotocpp_1d
01864
01865 !=====
01866
01867 !-----
01868 !  SUBROUTINE  CPP TO GEO SCALAR
01869 !-----
01892 !-----
01893 SUBROUTINE cpptgeo_scalar(x, y, lat0, lon0, lat, lon)
01894
01895     USE pahm_global, ONLY : rearth, deg2rad
01896
01897     IMPLICIT NONE
01898
01899     REAL(SZ), INTENT(IN)  :: x
01900     REAL(SZ), INTENT(IN)  :: y
01901     REAL(SZ), INTENT(IN)  :: lat0
01902     REAL(SZ), INTENT(IN)  :: lon0
01903     REAL(SZ), INTENT(OUT) :: lat
01904     REAL(SZ), INTENT(OUT) :: lon
01905
01906     lat = y / (deg2rad * rearth)
01907     lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01908
01909     END SUBROUTINE cpptgeo_scalar
01910
01911 !=====
01912
01913 !-----
01914 !  SUBROUTINE  CPP TO GEO 1D
01915 !-----
01939 !-----
01940 SUBROUTINE cpptgeo_1d(x, y, lat0, lon0, lat, lon)
01941
01942     USE pahm_global, ONLY : rearth, deg2rad
01943
01944     IMPLICIT NONE
01945
01946     REAL(SZ), INTENT(IN)  :: x(:)
01947     REAL(SZ), INTENT(IN)  :: y(:)
01948     REAL(SZ), INTENT(IN)  :: lat0
01949     REAL(SZ), INTENT(IN)  :: lon0
01950     REAL(SZ), INTENT(OUT) :: lat(:)
01951     REAL(SZ), INTENT(OUT) :: lon(:)
01952
01953     lat = y / (deg2rad * rearth)
01954     lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01955
01956     END SUBROUTINE cpptgeo_1d
01957
01958 !=====
01959
01960 !-----
01961 !  FUNCTION  SPHERICAL DISTANCE
01962 !-----
01992 !-----
01993 REAL(sz) function sphericaldistance_scalar(lat1, lon1, lat2, lon2) result(myvalout)
01994
01995     USE pahm_global, ONLY : rearth, deg2rad
01996
01997     IMPLICIT NONE
01998
01999     REAL(sz), INTENT(IN) :: lat1 ! latitude of point 1 on the sphere (degrees north)
02000     REAL(sz), INTENT(IN) :: lon1 ! longitude of point 1 on the sphere (degrees east)
02001     REAL(sz), INTENT(IN) :: lat2 ! latitude of point 2 on the sphere (degrees north)
02002     REAL(sz), INTENT(IN) :: lon2 ! longitude of point 2 on the sphere (degrees east)
02003
02004     REAL(sz)
02005         :: phi1, phi2, lamda1, lamda2, dphi, dlamda, dsigma
02006
02007     phi1 = deg2rad * lat1
02008     phi2 = deg2rad * lat2
02009     dphi = abs(phi2 - phi1)
02010
02011     lamda1 = deg2rad * lon1

```



```

02011     lamda2 = deg2rad * lon2
02012     dlamda = abs(lamda2 - lamda1)
02013
02014     ! Vincenty formula to calculate a distance along a sphere
02015     dsigma = atan(sqrt((cos(phi2) * sin(dlamda))**2 + &
02016                       (cos(phi1) * sin(phi2) - sin(phi1) * cos(phi2) * cos(dlamda))**2))
02017     dsigma = dsigma / (sin(phi1) * sin(phi2) + cos(phi1) * cos(phi2) * cos(dlamda))
02018
02019     ! This is the great-circle distance; REARTH in meters
02020     myvalout = rearth * dsigma
02021
02022     RETURN
02023
02024 END FUNCTION sphericaldistance_scalar
02025
02026 !=====
02027
02028 ! -----
02029 !  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
02030 ! -----
02060 !-----
02061 FUNCTION sphericaldistance_ld(lats, lons, lat0, lon0) RESULT(myValOut)
02062
02063     USE pahm_global, ONLY : rearth, deg2rad
02064
02065     IMPLICIT NONE
02066
02067     ! Global variables
02068     REAL(sz), INTENT(IN) :: lats(:) ! latitude of point 1 on the sphere (degrees north)
02069     REAL(sz), INTENT(IN) :: lons(:) ! longitude of point 1 on the sphere (degrees east)
02070     REAL(sz), INTENT(IN) :: lat0 ! latitude of point 2 on the sphere (degrees north)
02071     REAL(sz), INTENT(IN) :: lon0 ! longitude of point 2 on the sphere (degrees east)
02072
02073     REAL(sz), DIMENSION(:), ALLOCATABLE :: myvalout
02074
02075     ! Local variables
02076     REAL(sz), DIMENSION(:), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
02077     REAL(sz) :: phi0, lamda0
02078     INTEGER :: status, n1
02079
02080
02081     CALL setmessagessource("SphericalDistance_1D")
02082
02083     IF (SIZE(lats) /= SIZE(lons)) THEN
02084         WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
02085         CALL allmessage(error, scratchmessage)
02086
02087         CALL terminate()
02088     END IF
02089
02090     n1 = SIZE(lats, 1)
02091     ALLOCATE(myvalout(n1), stat = status)
02092     ALLOCATE(phis(n1), lamdas(n1), dphi(n1), dlamda(n1), dsigma(n1), stat = status)
02093
02094     IF (status /= 0) THEN
02095         WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02096         CALL allmessage(error, scratchmessage)
02097
02098         CALL terminate()
02099     END IF
02100
02101     phis = deg2rad * lats
02102     phi0 = deg2rad * lat0
02103     dphi = abs(phi0 - phis)
02104
02105     lamdas = deg2rad * lons
02106     lamda0 = deg2rad * lon0
02107     dlamda = abs(lamda0 - lamdas)
02108
02109     ! Vincenty formula to calculate a distance along a sphere
02110     dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
02111                       (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
02112     dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))
02113
02114     ! This is the great-circle distance; REARTH in meters
02115     myvalout = rearth * dsigma
02116
02117     DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)
02118
02119     CALL unsetmessagessource()
02120

```

```

02121     RETURN
02122
02123 END FUNCTION sphericaldistance_1d
02124
02125 !=====
02126
02127 ! -----
02128 !   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
02129 ! -----
02130
02131 FUNCTION sphericaldistance_2d(lats, lons, lat0, lon0) RESULT(myValOut)
02132
02133     USE pahm_global, ONLY : rearth, deg2rad
02134
02135     IMPLICIT NONE
02136
02137     ! Global variables
02138     REAL(sz), INTENT(IN) :: lats(:, :) ! latitude of point 1 on the sphere (degrees north)
02139     REAL(sz), INTENT(IN) :: lons(:, :) ! longitude of point 1 on the sphere (degrees east)
02140     REAL(sz), INTENT(IN) :: lat0      ! latitude of point 2 on the sphere (degrees north)
02141     REAL(sz), INTENT(IN) :: lon0      ! longitude of point 2 on the sphere (degrees east)
02142
02143     REAL(sz), DIMENSION(:, :), ALLOCATABLE :: myvalout
02144
02145     ! Local variables
02146     REAL(sz), DIMENSION(:, :), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
02147     REAL(sz)                                :: phi0, lamda0
02148     INTEGER                                :: status, n1, n2
02149
02150     CALL setmessagesource("SphericalDistance_2D")
02151
02152     IF (SIZE(lats) /= SIZE(lons)) THEN
02153         WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
02154         CALL allmessage(error, scratchmessage)
02155
02156         CALL unsetmessagesource()
02157
02158         CALL terminate()
02159     END IF
02160
02161     n1 = SIZE(lats, 1)
02162     n2 = SIZE(lats, 2)
02163     ALLOCATE(myvalout(n1, n2), stat = status)
02164     ALLOCATE(phis(n1, n2), lamdas(n1, n2), dphi(n1, n2), dlamda(n1, n2), dsigma(n1, n2), stat = status)
02165
02166     IF (status /= 0) THEN
02167         WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02168         CALL allmessage(error, scratchmessage)
02169
02170         CALL unsetmessagesource()
02171
02172         CALL terminate()
02173     END IF
02174
02175     phis = deg2rad * lats
02176     phi0 = deg2rad * lat0
02177     dphi = abs(phi0 - phis)
02178
02179     lamdas = deg2rad * lons
02180     lamda0 = deg2rad * lon0
02181     dlamda = abs(lamda0 - lamdas)
02182
02183     ! Vincenty formula to calculate a distance along a sphere
02184     dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
02185         (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
02186     dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))
02187
02188     ! This is the great-circle distance; REARTH in meters
02189     myvalout = rearth * dsigma
02190
02191     DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)
02192
02193     CALL unsetmessagesource()
02194
02195     RETURN
02196
02197 END FUNCTION sphericaldistance_2d
02198
02199 !=====
02200
02201

```

```

02231 ! -----
02232 !  FUNCTION  SPHERICAL  DISTANCE  HARV
02233 ! -----
02260 ! -----
02261 REAL(sZ) function sphericaldistanceharv(lat1, lon1, lat2, lon2) result(myvalout)
02262
02263     USE pahm_global, ONLY : rearth, deg2rad
02264
02265     IMPLICIT NONE
02266
02267     REAL(sZ), INTENT(IN) :: lat1    ! latitude of point 1 on the sphere (degrees north)
02268     REAL(sZ), INTENT(IN) :: lon1    ! longitude of point 1 on the sphere (degrees east)
02269     REAL(sZ), INTENT(IN) :: lat2    ! latitude of point 2 on the sphere (degrees north)
02270     REAL(sZ), INTENT(IN) :: lon2    ! longitude of point 2 on the sphere (degrees east)
02271
02272     REAL(sZ)
02273         :: phi1, phi2, lamda1, lamda2, dphi, dlamda, dsigma
02274
02275     phi1 = deg2rad * lat1
02276     phi2 = deg2rad * lat2
02277     dphi = abs(phi2 - phi1)
02278
02279     lamda1 = deg2rad * lon1
02280     lamda2 = deg2rad * lon2
02281     dlamda = abs(lamda2 - lamda1)
02282
02283     ! Haversine formula formula to calculate a distance along a sphere
02284     dsigma = sqrt(sin(dphi / 2.0_sZ)**2 + cos(phi1) * cos(phi2) * sin(dlamda / 2.0_sZ)**2)
02285     dsigma = 2.0_sZ * asin(dsigma)
02286
02287     ! This is the great-circle distance; REARTH in meters
02288     myvalout = rearth * dsigma
02289
02290     RETURN
02291
02292 END FUNCTION sphericaldistanceharv
02293 !=====
02294
02295 !DEL ! -----
02296 !DEL !  FUNCTION  SPHERICAL  DISTANCE  ADCIRC
02297 !DEL ! -----
02298 !DEL !   jgf49.1001 PV to be deleted
02299 !DEL !> Function to get the distance along the surface of
02300 !DEL !> a sphere (the earth's surface in this case).
02301 !DEL ! -----
02302 !DEL REAL(SZ) FUNCTION SphericalDistanceADCIRC(dx, dy, y1, y2) RESULT(myValOut)
02303
02304 !DEL     USE PaHM_Global, ONLY : REARTH, DEG2RAD
02305
02306 !DEL     IMPLICIT NONE
02307
02308 !DEL     REAL(SZ), INTENT(IN) :: dx    ! longitude distance in radians
02309 !DEL     REAL(SZ), INTENT(IN) :: dy    ! latitude distance in radians
02310 !DEL     REAL(SZ), INTENT(IN) :: y1    ! degrees latitude of starting point
02311 !DEL     REAL(SZ), INTENT(IN) :: y2    ! degrees latitude of ending point
02312
02313 !DEL     ! compute the distances based on haversine formula for
02314 !DEL     ! distance along a sphere
02315 !DEL     myValOut = SQRT(SIN(dy / 2.0_SZ)**2 +
02316 !DEL                  COS(y1 * DEG2RAD) * COS(y2 * DEG2RAD) * SIN(dx / 2.0_SZ)**2) &
02317
02318 !DEL     ! This is the great-circle distance; REARTH in meters
02319 !DEL     myValOut = REARTH * (2.0_SZ * ASIN(myValOut))
02320
02321 !DEL     RETURN
02322
02323 !DEL END FUNCTION SphericalDistanceADCIRC
02324
02325 !DEL=====
02326
02327 ! -----
02328 !  SUBROUTINE  SPHERICAL  FRAC  POINT
02329 ! -----
02363 ! -----
02364 SUBROUTINE sphericalfracpoint(lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, distl2)
02365
02366     USE pahm_global, ONLY : rearth, deg2rad, rad2deg
02367
02368     IMPLICIT NONE
02369
02370     ! Global variables

```

```

02371     REAL(SZ), INTENT(IN)           :: lat1      ! latitude of point 1 on the sphere (degrees north)
02372     REAL(SZ), INTENT(IN)           :: lon1      ! longitude of point 1 on the sphere (degrees east)
02373     REAL(SZ), INTENT(IN)           :: lat2      ! latitude of point 2 on the sphere (degrees north)
02374     REAL(SZ), INTENT(IN)           :: lon2      ! longitude of point 2 on the sphere (degrees east)
02375     REAL(SZ), INTENT(IN)           :: fraction   ! distance fraction of the intermediate point (0 <= f
<= 1)
02376     REAL(SZ), INTENT(OUT)           :: latf, lonf ! the calculated latitude and longitude of the
02377                                           ! intermediate point
02378     REAL(SZ), OPTIONAL, INTENT(OUT) :: distf      ! the distance between point 1 and the intermediate
point
02379     REAL(SZ), OPTIONAL, INTENT(OUT) :: dist12     ! the distance between point 1 and point 2
02380
02381     ! Local variables
02382     REAL(SZ)           :: myFrac
02383     REAL(SZ)           :: phi1, phi2, lamda1, lamda2, delta
02384     REAL(SZ)           :: aa, bb, xx, yy, zz
02385     REAL(SZ) :: myDist12, myDistF
02386
02387
02388     myfrac = fraction
02389     IF (myfrac < 0) myfrac = 0.0_sz
02390     IF (myfrac > 1) myfrac = 1.0_sz
02391
02392     ! Calculate the great circle distance between points 1 and 2
02393     mydist12 = sphericaldistance(lat1, lon1, lat2, lon2)
02394
02395     ! Distance is in meters (REARTH in meters). If myDist12 < 0.01_sz
02396     ! the two points are coincident
02397     IF (mydist12 < 0.01_sz) THEN
02398         latf = lat1
02399         lonf = lon1
02400         IF (PRESENT(distf)) distf = 0.0_sz
02401         IF (PRESENT(dist12)) dist12 = 0.0_sz
02402
02403         RETURN
02404     END IF
02405
02406     phi1 = deg2rad * lat1
02407     phi2 = deg2rad * lat2
02408     lamda1 = deg2rad * lon1
02409     lamda2 = deg2rad * lon2
02410
02411     delta = mydist12 / rearth
02412
02413     aa = sin((1.0_sz - myfrac) * delta) / sin(delta)
02414     bb = sin(myfrac * delta) / sin(delta)
02415
02416     xx = aa * cos(phi1) * cos(lamda1) + bb * cos(phi2) * cos(lamda2)
02417     yy = aa * cos(phi1) * sin(lamda1) + bb * cos(phi2) * sin(lamda2)
02418     zz = aa * sin(phi1) + bb * sin(phi2)
02419
02420     ! The (lat, lon) values of the intermediate point
02421     latf = rad2deg * atan2(zz, sqrt(xx * xx + yy * yy))
02422     lonf = rad2deg * atan2(yy, xx)
02423
02424     ! This is the great-circle distance; REARTH in meters
02425     mydistf = sphericaldistance(lat1, lon1, latf, lonf)
02426
02427     IF (PRESENT(distf)) distf = mydistf
02428     IF (PRESENT(dist12)) dist12 = mydist12
02429
02430     RETURN
02431
02432 END SUBROUTINE sphericalfracpoint
02433
02434 !=====
02435
02436 !-----
02437 ! SUBROUTINE GET LOC AND RATIO
02438 !-----
02439 !-----
02440
02441 SUBROUTINE getlocandratio(val, arrVal, idx1, idx2, wtRatio)
02442
02443     IMPLICIT NONE
02444
02445     ! Global variables
02446     REAL(SZ), INTENT(IN) :: val      ! value to search for
02447     REAL(SZ), INTENT(IN) :: arrVal(:) ! search array (1D)
02448     INTEGER, INTENT(OUT) :: idx1     ! the index of the lowest bound
02449     INTEGER, INTENT(OUT) :: idx2     ! the index of the highest bound
02450     REAL(SZ), INTENT(OUT) :: wtRatio ! the ratio factor that used in the linear interpolation

```

```

02472                                     ! calculations: F = F(idx1) + wtRatio * (F(idx2) - F(idx1))
02473                                     ! 0 <= wtRatio <= 1.0
02474
02475     ! Local variables
02476     INTEGER                :: nn, j1, j11, j12
02477     REAL(SZ)               :: diffVal
02478
02479
02480     idx1 = -1
02481     idx2 = -1
02482     wtratio = 0.0_sz
02483
02484     nn = SIZE(arrval, 1)
02485     j1 = minloc(abs(val - arrval), 1)
02486
02487     !----- Check if we got an exact bin value
02488     IF (comparereals(val - arrval(j1), 0.0_sz, 0.0001_sz) == 0) THEN
02489         idx1 = j1
02490         idx2 = j1
02491         wtratio = 0.0_sz
02492
02493         RETURN
02494     END IF
02495     !-----
02496
02497     !----- Checking the values at the two edges of the arrVal
02498     IF ((j1 == 1) .OR. (j1 == nn)) THEN
02499         IF (j1 == 1) THEN
02500             j11 = j1
02501             j12 = j1 + 1
02502         ELSE
02503             j11 = j1 - 1
02504             j12 = j1
02505         END IF
02506
02507         diffval = arrval(j12) - arrval(j11)
02508
02509         IF (comparereals(diffval, 0.0_sz, 0.0001_sz) == 0) THEN
02510             idx1 = j11
02511             idx2 = j11
02512             wtratio = 0.0_sz
02513         ELSE
02514             IF (comparereals(val - arrval(j11), 0.0_sz) * &
02515                 comparereals(val - arrval(j12), 0.0_sz) < 0) THEN
02516                 idx1 = j11
02517                 idx2 = j12
02518                 wtratio = (val - arrval(j11)) / diffval
02519             END IF
02520         END IF
02521
02522         RETURN
02523     END IF
02524     !-----
02525
02526     IF (comparereals(val - arrval(j1 - 1), 0.0_sz) * &
02527         comparereals(val - arrval(j1), 0.0_sz) < 0) THEN
02528         j11 = j1 - 1
02529         j12 = j1
02530
02531         diffval = arrval(j12) - arrval(j11)
02532
02533         idx1 = j11
02534         idx2 = j12
02535         wtratio = (val - arrval(j11)) / diffval
02536     ELSE IF (comparereals(val - arrval(j1), 0.0_sz) * &
02537         comparereals(val - arrval(j1 + 1), 0.0_sz) < 0) THEN
02538
02539         j11 = j1
02540         j12 = j1 + 1
02541
02542         diffval = arrval(j12) - arrval(j11)
02543
02544         idx1 = j11
02545         idx2 = j12
02546         wtratio = (val - arrval(j11)) / diffval
02547     END IF
02548
02549     RETURN
02550
02551 END SUBROUTINE getlocandratio
02552

```

```

02553 !=====
02554
02555 !-----
02556 ! FUNCTION CHAR UNIQUE
02557 !-----
02575 !-----
02576 INTEGER FUNCTION charunique(inpVec, outVec, idxVec) RESULT (myRec)
02577
02578     IMPLICIT NONE
02579
02580     CHARACTER(LEN=*) , INTENT(IN)                :: inpvec(:)
02581     CHARACTER(LEN=*) , INTENT(OUT)               :: outvec(:)
02582     INTEGER, ALLOCATABLE, INTENT(OUT)            :: idxvec(:)
02583
02584     CHARACTER(LEN=LEN(inpVec(1))), ALLOCATABLE :: chkstr(:)
02585     INTEGER, ALLOCATABLE                       :: chkint(:)
02586     INTEGER :: nels
02587     INTEGER :: icnt, jcnt    ! counters
02588
02589
02590     nels = SIZE(inpvec, 1)
02591
02592     ALLOCATE(chkstr(nels))
02593     ALLOCATE(chkint(nels))
02594
02595
02596     jcnt = 1
02597     DO icnt = 1, nels
02598         IF (trim(inpvec(icnt)) == "") cycle
02599         IF (any(chkstr == inpvec(icnt))) cycle
02600
02601         ! No match found so add it to the output
02602         chkstr(jcnt) = inpvec(icnt)
02603         chkint(jcnt) = icnt
02604         jcnt = jcnt + 1
02605     END DO
02606
02607     myrec = jcnt - 1
02608     outvec = chkstr
02609     idxvec = chkint
02610
02611     DEALLOCATE(chkstr)
02612     DEALLOCATE(chkint)
02613
02614     RETURN
02615
02616 END FUNCTION charunique
02617
02618 !=====
02619
02620 !-----
02621 ! FUNCTION VAL STR
02622 !-----
02641 !-----
02642 REAL(sp) function valstr(string) result(myval)
02643
02644     IMPLICIT NONE
02645
02646     ! Dummy arguments
02647     CHARACTER(LEN=*) , INTENT(IN) :: string
02648
02649     ! Local variables
02650     INTEGER :: i
02651     REAL(sp) :: v
02652
02653     i = realscan(string,1,v)
02654     myval = v
02655
02656     RETURN
02657
02658 END FUNCTION valstr
02659
02660 !=====
02661
02662 !-----
02663 ! FUNCTION D VAL STR
02664 !-----
02683 !-----
02684 REAL(hp) function dvalstr(string) result(myval)
02685
02686     IMPLICIT NONE

```

```

02687
02688     ! Dummy arguments
02689     CHARACTER(LEN=*), INTENT(IN) :: string
02690
02691     ! Local variables
02692     INTEGER :: i
02693     REAL(hp) :: v
02694
02695     i = drealscan(string,1,v)
02696     myval = v
02697
02698     RETURN
02699
02700 END FUNCTION dvalstr
02701
02702 !=====
02703
02704 !-----
02705 ! FUNCTION INT VAL STR
02706 !-----
02725 !-----
02726 INTEGER FUNCTION intvalstr(String) Result(myVal)
02727
02728     IMPLICIT NONE
02729
02730     ! Dummy arguments
02731     CHARACTER(LEN=*), INTENT(IN) :: string
02732
02733     ! Local variables
02734     INTEGER :: i
02735     INTEGER :: v
02736
02737     i = intscan(string,1,.true.,v)
02738     myval = v
02739
02740     RETURN
02741
02742 END FUNCTION intvalstr
02743
02744 !=====
02745
02746 !-----
02747 ! FUNCTION REAL SCAN
02748 !-----
02787 !-----
02788 INTEGER FUNCTION realscan(String, Pos, Value) Result(myVal)
02789
02790     IMPLICIT NONE
02791
02792     ! Dummy arguments
02793     INTEGER, INTENT(IN) :: pos
02794     CHARACTER(LEN=*), INTENT(IN) :: string
02795     REAL(sp), INTENT(OUT) :: value
02796
02797     ! Local variables
02798     INTEGER :: fract, intg, kfract, pmsign, power, ptr
02799
02800     ! CHECK POS.
02801     myval = pos
02802     Value = 0.0
02803     IF(pos < 1 .OR. len(string) < pos) RETURN
02804
02805     ! SET UP WORKING VARIABLES.
02806     intg = 0
02807     fract = 0
02808     kfract = 0
02809     power = 0
02810     DO WHILE (.true.)
02811         ! SKIP LEADING BLANKS.
02812         IF(string(myval:myval) == ' ') THEN
02813             myval = myval + 1
02814             IF(myval > len(string)) RETURN
02815             cycle
02816         END IF
02817
02818         ! LOOK FOR SIGN.
02819         ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
02820         pmsign = 0
02821         IF(string(myval:myval) == '+') THEN
02822             pmsign = +1
02823         ELSE IF(string(myval:myval) == '-') THEN

```

```

02824         pmsign = -1
02825     END IF
02826     IF (pmsign.NE.0) myval = myval + 1
02827
02828     ! LOOK FOR INTEGER PART.
02829     myval = intscan(string, myval, .false., intg)
02830
02831     ! LOOK FOR FRACTION PART.
02832     IF (myval.LE.len(string)) THEN
02833         IF (myval > pos+abs(pmsign)) THEN
02834             ! DETERMINE IF FIRST FORM OR SECOND FORM.
02835             ! HANDLE FIRST FORM: D+ ['. ' D*]
02836             IF (string(myval:myval) == ' ') THEN
02837                 myval = myval + 1
02838                 IF (myval.LE.len_trim(string)) THEN
02839                     IF (string(myval:myval).NE.' ') THEN
02840                         ptr = intscan(string, myval, .false., fract)
02841                         kffrac = ptr - myval
02842                         myval = ptr
02843                     END IF
02844                 END IF
02845             END IF
02846             ! HANDLE SECOND FORM: ' ' D+
02847             ELSE IF (string(myval:myval).NE.' ') THEN
02848                 ! IF ' ' MISSING, THEN WE HAVE NOTHING.
02849                 myval = pos
02850                 RETURN
02851             ELSE
02852                 myval = myval + 1
02853                 ptr = intscan(string, myval, .false., fract)
02854                 kffrac = ptr - myval
02855                 IF (kffrac == 0) THEN
02856                     ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
02857                     myval = pos
02858                     RETURN
02859                 ELSE
02860                     myval = ptr
02861                 END IF
02862             END IF
02863
02864             ! LOOK FOR EXPONENT PART.
02865             IF (myval.LE.len(string)) THEN
02866                 IF ((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e')) THEN
02867                     myval = myval + 1
02868                     ptr = intscan(string, myval, .true., power)
02869                     IF (ptr == myval) THEN
02870                         ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
02871                         ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
02872                         ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
02873                         myval = myval - 1
02874                         Value = intg + fract/10.0**kffrac
02875                         IF (pmsign == -1) Value = -Value
02876                         RETURN
02877                     ELSE
02878                         myval = ptr
02879                     END IF
02880                 END IF
02881             END IF
02882         END IF
02883
02884         ! COMPUTE REAL VALUE FROM ITS PARTS.
02885         IF (kffrac.NE.0) THEN
02886             Value = (intg+fract/10.0**kffrac)*10.0**power
02887         ELSE
02888             Value = intg*10.0**power
02889         END IF
02890         IF (pmsign == -1) Value = -Value
02891         EXIT
02892     END DO
02893
02894     RETURN
02895
02896 END FUNCTION realscan
02897
02898 !=====
02899
02900 !-----
02901 ! FUNCTION DREALSCAN
02902 !-----
02903 !-----
02942 INTEGER FUNCTION drealscan(String, Pos, Value) RESULT(myVal)

```



```

02943
02944     IMPLICIT NONE
02945
02946     ! Dummy arguments
02947     INTEGER, INTENT(IN)          :: pos
02948     CHARACTER(LEN=*), INTENT(IN) :: string
02949     REAL(hp), INTENT(OUT)        :: value
02950
02951     ! Local variables
02952     INTEGER :: fract, intg, kfract, pmsign, power, ptr
02953
02954     ! CHECK POS.
02955     myval = pos
02956     Value = 0.0
02957     IF (pos < 1 .OR. len(string) < pos) RETURN
02958
02959     ! SET UP WORKING VARIABLES.
02960     intg = 0
02961     fract = 0
02962     kfract = 0
02963     power = 0
02964     DO WHILE (.true.)
02965         ! SKIP LEADING BLANKS.
02966         IF (string(myval:myval) == ' ') THEN
02967             myval = myval + 1
02968             IF (myval > len(string)) RETURN
02969             cycle
02970         END IF
02971
02972         ! LOOK FOR SIGN.
02973         ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
02974         pmsign = 0
02975         IF (string(myval:myval) == '+') THEN
02976             pmsign = +1
02977         ELSE IF (string(myval:myval) == '-') THEN
02978             pmsign = -1
02979         END IF
02980         IF (pmsign.NE.0) myval = myval + 1
02981
02982         ! LOOK FOR INTEGER PART.
02983         myval = intscan(string, myval, .false., intg)
02984
02985         ! LOOK FOR FRACTION PART.
02986         IF (myval.LE.len(string)) THEN
02987             IF (myval > pos+abs(pmsign)) THEN
02988                 ! DETERMINE IF FIRST FORM OR SECOND FORM.
02989                 ! HANDLE FIRST FORM: D+ ['. ' D*]
02990                 IF (string(myval:myval) == '.') THEN
02991                     myval = myval + 1
02992                     IF (myval.LE.len_trim(string)) THEN
02993                         IF (string(myval:myval).NE.' ') THEN
02994                             ptr = intscan(string, myval, .false., fract)
02995                             kfract = ptr - myval
02996                             myval = ptr
02997                         END IF
02998                     END IF
02999                 ELSE
03000                     ! HANDLE SECOND FORM: '. ' D+
03001                     ELSE IF (string(myval:myval).NE.'.') THEN
03002                         ! IF '.' MISSING, THEN WE HAVE NOTHING.
03003                         myval = pos
03004                         RETURN
03005                     ELSE
03006                         myval = myval + 1
03007                         ptr = intscan(string, myval, .false., fract)
03008                         kfract = ptr - myval
03009                         IF (kfract == 0) THEN
03010                             ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
03011                             myval = pos
03012                             RETURN
03013                         ELSE
03014                             myval = ptr
03015                         END IF
03016                     END IF
03017                 END IF
03018
03019                 ! LOOK FOR EXPONENT PART.
03020                 IF (myval.LE.len(string)) THEN
03021                     IF ((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e') .OR. &
03022                         (string(myval:myval) == 'D') .OR. (string(myval:myval) == 'd')) THEN
03023                         myval = myval + 1
03024                         ptr = intscan(string, myval, .true., power)

```

```

03024         IF(ptr == myval) THEN
03025             ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
03026             ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
03027             ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
03028             myval = myval - 1
03029             Value = intg + fract/10.0**kfract
03030             IF(pmsign == -1)Value = -Value
03031             RETURN
03032         ELSE
03033             myval = ptr
03034         END IF
03035     END IF
03036 END IF
03037
03038
03039     ! COMPUTE REAL VALUE FROM ITS PARTS.
03040     IF(kfract.NE.0) THEN
03041         Value = (intg+fract/10.0**kfract)*10.0**power
03042     ELSE
03043         Value = intg*10.0**power
03044     END IF
03045     IF(pmsign == -1)Value = -Value
03046     EXIT
03047 END DO
03048
03049 RETURN
03050
03051 END FUNCTION drealscan
03052
03053 !=====
03054 !-----
03055 ! FUNCTION INTSCAN
03056 !-----
03057 !-----
03058 !-----
03095 INTEGER FUNCTION intscan(String, Pos, Signed, Value) Result(myVal)
03096
03097     IMPLICIT NONE
03098
03099     ! Dummy arguments
03100     INTEGER, INTENT(IN)          :: pos
03101     LOGICAL, INTENT(IN)         :: signed
03102     CHARACTER(LEN=*), INTENT(IN) :: string
03103     INTEGER, INTENT(OUT)        :: value
03104
03105     ! Local variables
03106     INTEGER(KIND=4) :: digit,pmsign
03107
03108     ! CHECK POS.
03109     myval = pos
03110     Value = 0
03111     IF(pos < 1 .OR. len(string) < pos)RETURN
03112     DO WHILE (.true.)
03113
03114         ! SKIP LEADING BLANKS.
03115         IF(string(myval:myval) == ' ') THEN
03116             myval = myval + 1
03117             IF(myval > len(string))RETURN
03118             cycle
03119         END IF
03120
03121         ! IF SIGNED, CHECK FOR SIGN.
03122         pmsign = 0
03123         IF(signed) THEN
03124             IF(string(myval:myval) == '+') THEN
03125                 pmsign = +1
03126             ELSE IF(string(myval:myval) == '-') THEN
03127                 pmsign = -1
03128             END IF
03129             IF(pmsign.NE.0)myval = myval + 1
03130
03131             ! IF sign is the last char in the field (with no integer following it)
03132             ! myVal value is left as POS or at the end of leading blanks.
03133             IF(myval > len_trim(string)) THEN
03134                 myval = myval - 1
03135                 RETURN
03136             END IF
03137         END IF
03138
03139         ! PROCESS DIGIT STRING.
03140         DO myval = myval, len(string)

```

```

03141         digit = ichar(string(myval:myval)) - ichar('0')
03142         IF(digit < 0 .OR. 9 < digit) GO TO 10
03143         Value = Value*10 + digit
03144     END DO
03145     ! Explicitly defined intscn to avoid possible compiler dependences (TWB. 930223)
03146     myval = len(string) + 1
03147     EXIT
03148 END DO
03149
03150 ! ADJUST SIGN.
03151 10 IF(signed.AND.pmsign == -1)Value = -Value
03152
03153 RETURN
03154
03155 END FUNCTION intscan
03156
03157 !=====
03158
03159 END MODULE utilities
03160

```

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 numerical 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 numerical 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 cyclostrophic 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(npoin, 4) [pahm_vortex::phis4](#)
- real(sz), dimension(npoin) [pahm_vortex::bs](#)
- real(sz), dimension(npoin, 4) [pahm_vortex::bs4](#)
- real(sz), dimension(npoin) [pahm_vortex::vmbl](#)
- real(sz), dimension(npoin, 4) [pahm_vortex::vmbl4](#)
- integer, dimension(npoin, 4) [pahm_vortex::quadflag4](#)
- real(sz), dimension(npoin, 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.](#)

```

00001 !-----
00002 !               M O D U L E   V O R T E X
00003 !-----
00014 !-----
00015
00016 MODULE pahm_vortex
00017
00018   USE pahm_sizes
00019   USE pahm_messages
00020
00021   IMPLICIT NONE
00022   SAVE
00023
00024   INTEGER, PARAMETER :: nquads = 4           ! Number of quadrants for which wind radii are
provided
00025   INTEGER, PARAMETER :: npoints = nquads + 2 ! Number of (theta, rMax) points for curve fit
00026   REAL(sz), DIMENSION(NPOINTS) :: rmaxes    ! Radius of maximum winds
00027   REAL(sz), DIMENSION(NPOINTS, 4) :: rmaxes4 ! (nautical miles)
00028
00029   REAL(sz) :: pn                               ! Ambient surface pressure (mb) !PV global
var?
00030   REAL(sz) :: pc                               ! Surface pressure at center of storm (mb) !PV
global var?
00031   REAL(sz) :: clat                             ! Latitude of storm center (degrees north)
!PV global var?
00032   REAL(sz) :: clon                             ! Longitude of storm center (degrees east )
!PV global var?

```

```

00033 REAL(sz)                                :: vmax                ! Max sustained wind velocity in storm (knots)
!PV global var?

00034
00035 REAL(sz)                                :: b                    ! Exponential shape parameter
00036 REAL(sz)                                :: corio                ! Coriolis force (1/s)
00037 REAL(sz)                                :: vr                    ! Velocity @ wind radii (knots)
00038 REAL(sz)                                :: phi
00039 REAL(sz), DIMENSION(NPOINTS)             :: phis                ! Correction factor to B and vh
00040 REAL(sz), DIMENSION(NPOINTS, 4)          :: phis4               ! Correction factor to B and vh
00041
00042 REAL(sz), DIMENSION(NPOINTS)             :: bs
00043 REAL(sz), DIMENSION(NPOINTS, 4)          :: bs4
00044 REAL(sz), DIMENSION(NPOINTS)             :: vmb1
00045 REAL(sz), DIMENSION(NPOINTS, 4)          :: vmb14
00046 INTEGER, DIMENSION(NPOINTS, 4)          :: quadflag4
00047 REAL(sz), DIMENSION(NPOINTS, 4)          :: quadir4
00048 REAL(sz), DIMENSION(NQUADS)              :: vrquadrant
00049 REAL(sz), DIMENSION(NQUADS)              :: radius                ! Wind radii - the distance
00050
00051 INTEGER                                  :: quad                ! Quadrant counter
00052
00053 REAL(sz)                                :: latestrmax            ! most recently calculated value of fitted
rmax
00054 REAL(sz)                                :: latestangle            ! angle of the most recently calculated node
w.r.t. the storm location
00055 LOGICAL                                  :: usequadrantvr
00056 LOGICAL                                  :: usevmaxesbl
00057
00058
00059 CONTAINS
00060
00061
00062 !-----
00063 ! SUBROUTINE CALC INTENSITY CHANGE
00064 !-----
00065 !-----
00078 !-----
00079 SUBROUTINE calcintensitychange(var, times, calcInt, status, order)
00080
00081 USE pahm_global, ONLY : deg2rad
00082 USE utilities, ONLY : sphericaldistance
00083
00084 IMPLICIT NONE
00085
00086 REAL(SZ), DIMENSION(:), INTENT(IN)       :: var, times
00087 INTEGER, OPTIONAL, INTENT(IN)             :: order
00088
00089 REAL(SZ), DIMENSION(:), INTENT(OUT)      :: calcInt
00090 INTEGER, INTENT(OUT)                     :: status
00091
00092 INTEGER                                  :: ordAcur
00093 REAL(SZ)                                 :: dt1, dt2
00094 LOGICAL                                  :: dt1OK, dt2OK
00095 REAL(SZ)                                 :: val1, val2
00096 INTEGER                                  :: iCnt, maxCnt
00097
00098 status = 0
00099 maxcnt = 0
00100
00101 CALL setmessagesource("CalcIntensityChange")
00102
00103 IF ((SIZE(shape(var)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
00104 WRITE(scratchmessage, '(a)') 'The rank of arrays var and times should be equal to 1 (vectors)'
00105 CALL allmessage(error, scratchmessage)
00106
00107 CALL unsetmessagesource()
00108
00109 status = 1
00110
00111 RETURN
00112 ELSE
00113 maxcnt = SIZE(var)
00114 END IF
00115
00116 ordacur = 2
00117 IF (PRESENT(order)) THEN
00118 IF (order <= 1) ordacur = 1
00119 IF (order > 1) ordacur = 2
00120 END IF
00121 IF (SIZE(var) < 3) ordacur = 1
00122
00123 ! Case 1st orded accuracy using backward differences

```

```

00124     IF (ordacur == 1 ) THEN
00125     DO icnt = 2, maxcnt
00126         dt1 = times(icnt) - times(icnt - 1)
00127         dtlok = (comparereals(dt1, 0.0_sz) /=0)
00128
00129         vall = 0.0_sz
00130         IF (dtlok) vall = (var(icnt) - var(icnt - 1)) / dt1
00131
00132         calcint(icnt) = vall
00133     END DO
00134     calcint(1) = calcint(2)
00135
00136     CALL unsetmessagesource()
00137
00138     RETURN
00139 END IF
00140
00141 ! Case 2nd order accuracy using Forward differences for the first point,
00142 ! backward differences for the last point and central differences in
00143 ! between points. Temporal spacing assumed to be uneven (general case).
00144 ! Forward, backward and central differences are all 2nd order accurate
00145 ! approximations.
00146
00147 !----- Forward differences (first point)
00148 icnt = 1
00149 dt1 = times(icnt + 1) - times(icnt)
00150 dtlok = (comparereals(dt1, 0.0_sz) /=0)
00151 dt2 = times(icnt + 2) - times(icnt + 1)
00152 dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00153
00154 vall = 0.0_sz
00155 IF (dtlok) vall = (var(icnt + 1) - var(icnt)) / dt1
00156
00157 val2 = 0.0_sz
00158 IF (dt2ok) val2 = (var(icnt + 2) - var(icnt + 1)) / dt2
00159
00160 IF (dtlok .AND. dt2ok) THEN
00161     calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * vall - (dt1 / (dt1 + dt2)) * val2
00162 ELSE IF (.NOT. dtlok) THEN
00163     calcint(icnt) = vall
00164 ELSE
00165     calcint(icnt) = 2.0_sz * vall - val2
00166 END IF
00167 !----- Forward differences (first point)
00168
00169 !----- Central differences
00170 DO icnt = 2, maxcnt - 1
00171     ! Forward
00172     dt1 = times(icnt + 1) - times(icnt)
00173     dtlok = (comparereals(dt1, 0.0_sz) /=0)
00174     ! Backward
00175     dt2 = times(icnt) - times(icnt - 1)
00176     dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00177
00178     vall = 0.0_sz
00179     IF (dtlok) vall = (var(icnt + 1) - var(icnt)) / dt1
00180
00181     val2 = 0.0_sz
00182     IF (dt2ok) val2 = (var(icnt) - var(icnt - 1)) / dt2
00183
00184     IF (dtlok .AND. dt2ok) THEN
00185         calcint(icnt) = (dt2 / (dt1 + dt2)) * vall + (dt1 / (dt1 + dt2)) * val2
00186     ELSE IF (.NOT. dtlok) THEN
00187         calcint(icnt) = vall
00188     ELSE
00189         calcint(icnt) = val2
00190     END IF
00191 END DO
00192 !----- Central differences
00193
00194 !----- Backward differences (last point)
00195 icnt = maxcnt
00196 dt1 = times(icnt) - times(icnt - 1)
00197 dtlok = (comparereals(dt1, 0.0_sz) /=0)
00198 dt2 = times(icnt - 1) - times(icnt - 2)
00199 dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00200
00201 vall = 0.0_sz
00202 IF (dtlok) vall = (var(icnt) - var(icnt - 1)) / dt1
00203
00204 val2 = 0.0_sz

```

```

00205      IF (dt2ok) val2 = (var(icnt - 1) - var(icnt - 2)) / dt2
00206
00207      IF (dtlok .AND. dt2ok) THEN
00208          calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * val1 - (dt1 / (dt1 + dt2)) * val2
00209      ELSE IF (.NOT. dtlok) THEN
00210          calcint(icnt) = val1
00211      ELSE
00212          calcint(icnt) = 2.0_sz * val1 - val2
00213      END IF
00214      !----- Backward differences (last point)
00215
00216      CALL unsetmessagesource()
00217
00218  END SUBROUTINE calcintensitychange
00219
00220  !=====
00221
00222  !-----
00223  ! SUBROUTINE UVTRANS
00224  !-----
00240  !-----
00241  SUBROUTINE uvtrans(lat, lon, times, u, v, status, order)
00242
00243      USE pahm_global, ONLY : deg2rad
00244      USE utilities, ONLY : sphericaldistance
00245
00246      IMPLICIT NONE
00247
00248      REAL(SZ), DIMENSION(:), INTENT(IN) :: lat, lon, times
00249      INTEGER, OPTIONAL, INTENT(IN) :: order
00250
00251      REAL(SZ), DIMENSION(:), INTENT(OUT) :: u, v
00252      INTEGER, INTENT(OUT) :: status
00253
00254      INTEGER :: ordAcur
00255      REAL(SZ) :: dx1, dy1, dx2, dy2
00256      REAL(SZ) :: dt1, dt2
00257      LOGICAL :: dt1OK, dt2OK
00258      REAL(SZ) :: u1, u2, v1, v2
00259      INTEGER :: iCnt, maxCnt
00260
00261      status = 0
00262      maxcnt = 0
00263
00264      CALL setmessagesource("UVTrans")
00265
00266      IF ((SIZE(shape(lat)) /= 1) .OR. (SIZE(shape(lon)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
00267          WRITE(scratchmessage, '(a)') 'The rank of arrays lat, lon and times should be equal to 1 (vectors)'
00268          CALL allmessage(error, scratchmessage)
00269
00270          CALL unsetmessagesource()
00271
00272          status = 1
00273
00274          RETURN
00275      ELSE
00276          maxcnt = SIZE(lat)
00277      END IF
00278
00279      ordacur = 2
00280      IF (PRESENT(order)) THEN
00281          IF (order <= 1) ordacur = 1
00282          IF (order > 1) ordacur = 2
00283      END IF
00284      IF (SIZE(lat) < 3) ordacur = 1
00285
00286      ! Case 1st orded accuracy using backward differences
00287      IF (ordacur == 1) THEN
00288          DO icnt = 2, maxcnt
00289              dx1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
00290              dy1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
00291              dt1 = abs(times(icnt) - times(icnt - 1))
00292              dtlok = (comparereals(dt1, 0.0_sz) /= 0)
00293
00294              u1 = 0.0_sz
00295              v1 = 0.0_sz
00296              IF (dtlok) THEN
00297                  u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
00298                  v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
00299              END IF
00300

```



```

00301      u(icnt) = u1
00302      v(icnt) = v1
00303      END DO
00304      u(1) = u(2)
00305      v(1) = v(2)
00306
00307      CALL unsetmessagesource()
00308
00309      RETURN
00310  END IF
00311
00312      ! Case 2nd order accuracy using Forward differences for the first point,
00313      ! backward differences for the last point and central differences in
00314      ! between points. Temporal spacing assumed to be uneven (general case).
00315      ! Forward, backward and central differences are all 2nd order accurate
00316      ! approximations.
00317
00318      !----- Forward differences (first point)
00319      icnt = 1
00320      dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
00321      dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
00322      dt1 = abs(times(icnt + 1) - times(icnt))
00323      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00324
00325      dx2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 1), lon(icnt + 2))
00326      dy2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 2), lon(icnt + 1))
00327      dt2 = abs(times(icnt + 2) - times(icnt + 1))
00328      dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00329
00330      u1 = 0.0_sz
00331      v1 = 0.0_sz
00332      IF (dt1ok) THEN
00333          u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
00334          v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
00335      END IF
00336
00337      u2 = 0.0_sz
00338      v2 = 0.0_sz
00339      IF (dt2ok) THEN
00340          u2 = sign(dx2 / dt2, (lon(icnt + 2) - lon(icnt + 1)))
00341          v2 = sign(dy2 / dt2, (lat(icnt + 2) - lat(icnt + 1)))
00342      END IF
00343
00344      IF (dt1ok .AND. dt2ok) THEN
00345          u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
00346          v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
00347      ELSE IF (.NOT. dt1ok) THEN
00348          u(icnt) = u1
00349          v(icnt) = v1
00350      ELSE
00351          u(icnt) = 2.0_sz * u1 - u2
00352          v(icnt) = 2.0_sz * v1 - v2
00353      END IF
00354      !----- Forward differences (first point)
00355
00356      !----- Central differences
00357      DO icnt = 2, maxcnt - 1
00358          ! Forward
00359          dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
00360          dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
00361          dt1 = abs(times(icnt + 1) - times(icnt))
00362          dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00363          ! Backward
00364          dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
00365          dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
00366          dt2 = abs(times(icnt) - times(icnt - 1))
00367          dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00368
00369          u1 = 0.0_sz
00370          v1 = 0.0_sz
00371          IF (dt1ok) THEN
00372              u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
00373              v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
00374          END IF
00375
00376          u2 = 0.0_sz
00377          v2 = 0.0_sz
00378          IF (dt2ok) THEN
00379              u2 = sign(dx2 / dt2, (lon(icnt) - lon(icnt - 1)))
00380              v2 = sign(dy2 / dt2, (lat(icnt) - lat(icnt - 1)))
00381          END IF

```

```

00382
00383     IF (dt1ok .AND. dt2ok) THEN
00384         u(icnt) = (dt2 / (dt1 + dt2)) * u1 + (dt1 / (dt1 + dt2)) * u2
00385         v(icnt) = (dt2 / (dt1 + dt2)) * v1 + (dt1 / (dt1 + dt2)) * v2
00386     ELSE IF (.NOT. dt1ok) THEN
00387         u(icnt) = u1
00388         v(icnt) = v1
00389     ELSE
00390         u(icnt) = u2
00391         v(icnt) = v2
00392     END IF
00393 END DO
00394 !----- Central differences
00395
00396 !----- Backward differences (last point)
00397 icnt = maxcnt
00398 dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt - 1))
00399 dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt - 1), lon(icnt))
00400 dt1 = abs(times(icnt) - times(icnt - 1))
00401 dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00402
00403 dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt - 2))
00404 dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 2), lon(icnt - 1))
00405 dt2 = abs(times(icnt - 1) - times(icnt - 2))
00406 dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00407
00408 u1 = 0.0_sz
00409 v1 = 0.0_sz
00410 IF (dt1ok) THEN
00411     u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
00412     v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
00413 END IF
00414
00415 u2 = 0.0_sz
00416 v2 = 0.0_sz
00417 IF (dt2ok) THEN
00418     u2 = sign(dx2 / dt2, (lon(icnt - 1) - lon(icnt - 2)))
00419     v2 = sign(dy2 / dt2, (lat(icnt - 1) - lat(icnt - 2)))
00420 END IF
00421
00422 IF (dt1ok .AND. dt2ok) THEN
00423     u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
00424     v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
00425 ELSE IF (.NOT. dt1ok) THEN
00426     u(icnt) = u1
00427     v(icnt) = v1
00428 ELSE
00429     u(icnt) = 2.0_sz * u1 - u2
00430     v(icnt) = 2.0_sz * v1 - v2
00431 END IF
00432 !----- Backward differences (last point)
00433
00434 CALL unsetmessagesource()
00435
00436 END SUBROUTINE uvtrans
00437
00438 !=====
00439
00440 !-----
00441 ! SUBROUTINE UVTRANSPOINT
00442 !-----
00443 !-----
00444 SUBROUTINE uvtranspoint(lat1, lon1, lat2, lon2, time1, time2, u, v)
00445
00446     USE pahm_global, ONLY : deg2rad
00447     USE utilities, ONLY : sphericaldistance
00448
00449     IMPLICIT NONE
00450
00451     ! Global variables
00452     REAL(SZ), INTENT(IN) :: lat1, lon1, lat2, lon2
00453     REAL(SZ), INTENT(IN) :: time1, time2
00454     REAL(SZ), INTENT(OUT) :: u, v
00455
00456     ! Local variables
00457     REAL(SZ) :: dx, dy, dt
00458     LOGICAL :: dtOK
00459
00460     dx = sphericaldistance(lat1, lon1, lat1, lon2)
00461     dy = sphericaldistance(lat1, lon1, lat2, lon1)
00462     dt = abs(time2 - time1)

```

```

00476      dtok = (comparereals(dt, 0.0_sz) /=0)
00477
00478      u = 0.0_sz
00479      v = 0.0_sz
00480      IF (dtok) THEN
00481          u = sign(dx / dt, (lon2 - lon1))
00482          v = sign(dy / dt, (lat2 - lat1))
00483      END IF
00484
00485  END SUBROUTINE uvtranspoint
00486
00487  !=====
00488  !-----
00489  ! SUBROUTINE NEW VORTEX
00490  !-----
00491  !-----
00492  !-----
00504  !-----
00505  SUBROUTINE newvortex(pinf, p0, lat, lon, vm)
00506
00507      USE pahm_global, ONLY : rhoair, deg2rad, omega, mb2pa, kt2ms
00508
00509      IMPLICIT NONE
00510
00511      REAL(SZ), INTENT(IN) :: pinf
00512      REAL(SZ), INTENT(IN) :: p0
00513      REAL(SZ), INTENT(IN) :: lat
00514      REAL(SZ), INTENT(IN) :: lon
00515      REAL(SZ), INTENT(IN) :: vm
00516
00517      ! set instance variables
00518      pn = pinf
00519      pc = p0
00520      clat = lat
00521      clon = lon
00522      vmax = vm
00523  !PV Check conversions
00524      ! evaluate basic physical params
00525      corio = 2.0_sz * omega * sin(deg2rad * clat)
00526      b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
00527      b = max(min(b, 2.0_sz), 1.0_sz) ! limit B to range 1.0->2.5
00528  !PV Data already have been converted
00529      ! added for compatibility of CalcRMaxes to use with simplified nws20
00530      bs(1:6) = b
00531      vmb1(1:6) = vmax
00532
00533  END SUBROUTINE newvortex
00534
00535  !=====
00536  !-----
00537  ! SUBROUTINE NEW VORTEX FULL
00538  !-----
00539  !-----
00542  !-----
00543  SUBROUTINE newvortexfull(pinf, p0, lat, lon, vm)
00544
00545      USE pahm_global, ONLY : rhoair, deg2rad, kt2ms, omega, mb2pa
00546
00547      IMPLICIT NONE
00548
00549      REAL(SZ), INTENT(IN) :: pinf
00550      REAL(SZ), INTENT(IN) :: p0
00551      REAL(SZ), INTENT(IN) :: lat
00552      REAL(SZ), INTENT(IN) :: lon
00553      REAL(SZ), INTENT(IN) :: vm
00554
00555      ! set instance variables
00556      pn = pinf
00557      pc = p0
00558      clat = lat
00559      clon = lon
00560      vmax = vm
00561
00562      ! evaluate basic physical params
00563      corio = 2.0_sz * omega * sin(deg2rad * clat)
00564      b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
00565      phi = 1.0_sz
00566      bs(1:6) = b
00567      phis(1:6) = phi
00568      vmb1(1:6) = vmax
00569

```

```

00570      ! Jie 2013.01
00571      ! B = MAX(MIN(B, 2.0_SZ), 1.0_SZ) ! limit B to range 1.0->2.5
00572
00573      END SUBROUTINE newvortexfull
00574
00575      !=====
00576
00577      !-----
00578      ! SUBROUTINE SET VORTEX
00579      !-----
00590      !-----
00591      SUBROUTINE setvortex(pinf, p0, lat, lon)
00592
00593          USE pahm_global, ONLY : deg2rad, omega
00594
00595          IMPLICIT NONE
00596
00597          REAL(SZ), INTENT(IN) :: pinf
00598          REAL(SZ), INTENT(IN) :: p0
00599          REAL(SZ), INTENT(IN) :: lat
00600          REAL(SZ), INTENT(IN) :: lon
00601
00602          ! set instance variables
00603          pn = pinf
00604          pc = p0
00605          clat = lat
00606          clon = lon
00607
00608          ! evaluate basic physical params
00609          corio = 2.0_sz * omega * sin(deg2rad * clat)
00610
00611      END SUBROUTINE setvortex
00612
00613      !=====
00614
00615      !-----
00616      ! SUBROUTINE SET RMAXES
00617      !-----
00618      SUBROUTINE setrmaxes(rMaxW)
00619
00620          IMPLICIT NONE
00621
00622          REAL(SZ), DIMENSION(4), INTENT(IN) :: rMaxW
00623          INTEGER :: i
00624
00625          DO i = 1, 4
00626              rmaxes(i + 1) = rmaxw(i)
00627          END DO
00628
00629      END SUBROUTINE setrmaxes
00630
00631      !=====
00632
00633      !-----
00634      ! SUBROUTINE GET RMAXES
00635      !-----
00636      SUBROUTINE getrmaxes(rMaxW)
00637
00638          IMPLICIT NONE
00639
00640          REAL(SZ), DIMENSION(4), INTENT(OUT) :: rMaxW
00641
00642          INTEGER :: i
00643
00644          DO i = 1, 4
00645              rmaxw(i) = rmaxes(i + 1)
00646          END DO
00647
00648      END SUBROUTINE getrmaxes
00649
00650      !=====
00651
00652      !-----
00653      ! SUBROUTINE CALC RMAXES
00654      !-----
00655      ! Jie 2014.07 Modified with quadrant-varying vmBL, which not only
00656      ! works for nws19 but for the simplified nws20
00657      !-----
00666      SUBROUTINE calcrmaxes()
00667
00668          IMPLICIT NONE

```

```

00669
00670 REAL(SZ)                :: root          ! Radius of maximum winds
00671 REAL(SZ), PARAMETER :: INNERRADIUS = 1.0_sz
00672 REAL(SZ), PARAMETER :: OUTERRADIUS = 400.0_sz
00673 REAL(SZ), PARAMETER :: ACCURACY    = 0.0001_sz
00674 REAL(SZ), PARAMETER :: ZOOM        = 0.01_sz
00675 INTEGER, PARAMETER :: ITERMAX      = 3
00676 REAL(SZ)            :: r1, r2, r3, r4, dr
00677 REAL(SZ)            :: vicinity
00678 INTEGER              :: n, iter
00679
00680 !-----
00681 ! Loop over quadrants of storm
00682 !-----
00683 DO n = 1, nquads
00684   ! set B and vMax values for each quadrant
00685   ! for nws19, B and vMax are constant
00686   ! for simplified nws20, B is constant, while vMax is not
00687   b = bs(n + 1)
00688   vmax = vmb1(n + 1)
00689
00690   quad = n
00691   root = -1.0_sz
00692   r1 = innerradius
00693   r2 = outerradius
00694   dr = 1.0_sz
00695   DO iter = 1, itermax
00696     root = findroot(vhwithcori, r1, r2, dr, r3, r4)
00697     r1 = r3
00698     r2 = r4
00699     dr = dr * zoom
00700   END DO
00701
00702   ! determine if rMax is actually in the vicinity of the
00703   ! isotach radius that we are using to solve for rMax,
00704   ! and if so, take another shot at finding the
00705   ! rMax using the gradient wind balance that neglects
00706   ! coriolis (and is appropriate in the vicinity of rMax)
00707   vicinity = abs(root - radius(quad)) / root
00708   IF ((root < 0.0_sz) .OR. (vicinity <= 0.0_sz)) THEN
00709     r1 = innerradius
00710     r2 = outerradius
00711     dr = 1.0_sz
00712     DO iter = 1, itermax
00713       root = findroot(vhnocori, r1, r2, dr, r3, r4)
00714       r1 = r3
00715       r2 = r4
00716       dr = dr * zoom
00717     END DO
00718   END IF
00719
00720   rmaxes(n + 1) = root
00721 END DO
00722
00723 END SUBROUTINE calcrmaxes
00724
00725 !=====
00726
00727 !-----
00728 ! SUBROUTINE CALC RMAXES FULL
00729 !-----
00741 ! Jie 2013.02 added looping procedures to calculate bs and phis
00742 !-----
00743 SUBROUTINE calcrmaxesfull()
00744
00745   USE pahm_global, ONLY : rhoair, nm2m, kt2ms, mb2pa
00746
00747   IMPLICIT NONE
00748
00749   REAL(SZ)                :: root          ! Radius of maximum winds
00750   REAL(SZ), PARAMETER :: INNERRADIUS = 1.0_sz
00751   REAL(SZ), PARAMETER :: OUTERRADIUS = 500.0_sz
00752   REAL(SZ), PARAMETER :: ACCURACY    = 0.0001_sz
00753   REAL(SZ), PARAMETER :: ZOOM        = 0.01_sz
00754   INTEGER, PARAMETER :: ITERMAX      = 3
00755   REAL(SZ)            :: r1, r2, r3, r4, dr
00756   INTEGER              :: n, iter, noRootFlag
00757   REAL(SZ)            :: bNew, bNew1
00758   REAL(SZ)            :: phiNew
00759   INTEGER, PARAMETER :: cont = 400      ! Max # of iterations
00760   INTEGER              :: iCont, ibCont ! iteration counter

```

```

00761
00762 211 FORMAT(a7, x ,i2, x, a38)
00763
00764 !-----
00765 ! Loop over quadrants of storm
00766 !-----
00767 DO n = 1, nquads
00768   norootflag = 0
00769
00770   ! initialize B and phi values for each quadrant
00771   b   = bs(n + 1)
00772   phi = phis(n + 1)
00773   vmax = vmbl(n + 1)
00774
00775   ! Loop the root-solving process to converge B, for in the
00776   ! new wind formulation, B is a function of rMax, vMax, f, and phi
00777   DO icont = 1, cont ! logical expre. is at the end to exit the loop
00778     norootflag = 0
00779     quad = n
00780     root = -1.0_sz
00781     r1 = innerradius
00782     r2 = outerradius
00783     dr = 1.0_sz
00784     DO iter = 1, itermax
00785       root = findroot(vhwithcorifull, r1, r2, dr, r3, r4)
00786       r1 = r3
00787       r2 = r4
00788       dr = dr * zoom
00789     END DO
00790
00791     ! avoid invalid B value when root is not found
00792     IF (root < 0.0_sz) THEN
00793       ! r1 = INNERRADIUS
00794       ! r2 = OUTERRADIUS
00795       ! dr = 1.0_SZ
00796       ! DO iter = 1, ITERMAX
00797       !   root = FindRoot(VhNoCori, r1, r2, dr, r3, r4)
00798       !   r1 = r3
00799       !   r2 = r4
00800       !   dr = dr * ZOOM
00801       ! END DO
00802       root = 1.0 * radius(quad)
00803       norootflag = 1
00804     END IF
00805
00806     rmaxes(n + 1) = root
00807
00808     ! Jie 2013.02
00809     ! determine if B converges, if yes, break loop and assign
00810     ! values to rMaxes, if not, continue the loop to re-calculate
00811     ! root and re-evaluate bs
00812     phinew = 1 + vmax * kt2ms * root * nm2m * corio /
00813             (b * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio)) &
00814     bnew = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) * &
00815           rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa)
00816     DO ibcont = 1, cont
00817       bnew1 = bnew
00818       phinew = 1 + vmax * kt2ms * root * nm2m * corio /
00819             (bnew * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio)) &
00820       bnew = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) * &
00821             rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa)
00822
00823       IF (abs(bnew - bnew1) <= 0.01_sz) EXIT
00824     END DO
00825
00826     ! debug with aswip
00827     !IF (ibCont >= cont) THEN
00828     !  WRITE(1111, 211) "iquad=", n, "bNew did not fully converge, procede"
00829     !END IF
00830     ! end debug with aswip
00831
00832     IF (abs(b - bnew) <= 0.01_sz) EXIT
00833
00834     ! update B and phi for next iteration
00835     ! warning: modifications made here also affect other subroutines
00836     b = bnew
00837     phi = phinew
00838   END DO !icont = 1, cont
00839
00840   ! update to the latest values for aswip output
00841   bs(n + 1) = bnew

```

```

00842     phis(n + 1) = phinew
00843
00844     ! debug with aswip
00845     ! IF (iCont >= cont) THEN
00846     !   WRITE(1111, 211) "iquad=", n, "B did not fully converge, procede"
00847     ! END IF
00848     ! end debug with aswip
00849
00850     ! determine if rMax is actually in the vicinity of the
00851     ! isotach radius that we are using to solve for rMax,
00852     ! and if so, take another shot at finding the
00853     ! rMax using the gradient wind equation that neglects
00854     ! coriolis (and is appropriate in the vicinity of rMax)
00855     ! Jie 2013.01
00856     ! vicinity = ABS(root - radius(quad)) / root
00857     IF (norootflag == 1) THEN
00858       WRITE(*, *) "iquad=", n, "No root found, return dist. to Isotach"
00859     END IF
00860     END DO ! n = 1, nQuads
00861
00862   END SUBROUTINE calcrmaxesfull
00863
00864   !=====
00865
00866   !-----
00867   ! SUBROUTINE FIT RMAXES
00868   !-----
00869   ! RJW 07 - 2009
00870   !-----
00880   SUBROUTINE fitrmaxes()
00881
00882     IMPLICIT NONE
00883
00884     ! Generate 2 additional (theta, rMax) points for curve-fit
00885     rmaxes(1) = rmaxes(5)
00886     rmaxes(6) = rmaxes(2)
00887
00888   END SUBROUTINE fitrmaxes
00889
00890   !=====
00891
00892   !-----
00893   ! SUBROUTINE FIT RMAXES 4
00894   !-----
00895   SUBROUTINE fitrmaxes4()
00896
00897     IMPLICIT NONE
00898
00899     ! Generate 2 additional points for curve-fit
00900     quadflag4(1, 1:4) = quadflag4(5, 1:4)
00901     quadflag4(6, 1:4) = quadflag4(2, 1:4)
00902
00903     quadir4(1, 1:4) = quadir4(5, 1:4)
00904     quadir4(6, 1:4) = quadir4(2, 1:4)
00905
00906     rmaxes4(1, 1:4) = rmaxes4(5, 1:4)
00907     rmaxes4(6, 1:4) = rmaxes4(2, 1:4)
00908
00909     bs4(1, 1:4) = bs4(5, 1:4)
00910     bs4(6, 1:4) = bs4(2, 1:4)
00911
00912     phis4(1, 1:4) = phis4(5, 1:4)
00913     phis4(6, 1:4) = phis4(2, 1:4)
00914
00915     vmbl4(1, 1:4) = vmbl4(5, 1:4)
00916     vmbl4(6, 1:4) = vmbl4(2, 1:4)
00917
00918   END SUBROUTINE fitrmaxes4
00919
00920   !=====
00921
00922   !-----
00923   ! SUBROUTINE SET VMAXES BL
00924   !-----
00925   SUBROUTINE setvmaxesbl(vMaxW)
00926
00927     IMPLICIT NONE
00928
00929     REAL(SZ), DIMENSION(4), INTENT(IN) :: vMaxW
00930
00931     INTEGER :: i

```

```

00932
00933      DO i = 1, 4
00934          vmb1(i + 1) = vmaxw(i)
00935      END DO
00936
00937  END SUBROUTINE setvmaxesbl
00938
00939  !=====
00940
00941  !-----
00942  ! SUBROUTINE GET VMAXES BL
00943  !-----
00944  SUBROUTINE getvmaxesbl(vMaxW)
00945
00946      IMPLICIT NONE
00947
00948      REAL(SZ), DIMENSION(4), INTENT(OUT) :: vMaxW
00949
00950      INTEGER :: i
00951
00952      DO i = 1, 4
00953          vmaxw(i) = vmb1(i + 1)
00954      END DO
00955
00956  END SUBROUTINE getvmaxesbl
00957
00958  !=====
00959
00960  !-----
00961  ! SUBROUTINE SET USE VMAXES BL
00962  !-----
00963  SUBROUTINE setusevmaxesbl(u)
00964
00965      IMPLICIT NONE
00966
00967      LOGICAL, INTENT(IN) :: u
00968
00969      usevmaxesbl = u
00970
00971  END SUBROUTINE setusevmaxesbl
00972
00973  !-----
00974  ! SUBROUTINE SET SHAPE PARAMETER
00975  !-----
00976  SUBROUTINE setshapeparameter(param)
00977
00978      IMPLICIT NONE
00979
00980      REAL(SZ) :: param
00981
00982      b = param
00983
00984  END SUBROUTINE setshapeparameter
00985
00986  !=====
00987
00988  !-----
00989  ! FUNCTION GET SHAPE PARAMETER
00990  !-----
00991  REAL(sz) function getshapeparameter() result(myvalout)
00992
00993      IMPLICIT NONE
00994
00995      myvalout = b
00996
00997      RETURN
00998
00999  END FUNCTION getshapeparameter
01000
01001  !=====
01002
01003  !-----
01004  ! FUNCTION GET SHAPE PARAMETERS
01005  !-----
01006  FUNCTION getshapeparameters() RESULT(myValOut)
01007
01008      IMPLICIT NONE
01009
01010      REAL(sz), DIMENSION(4) :: myvalout
01011
01012      INTEGER :: i

```



```

01013
01014     DO i = 1, 4
01015         myvalout(i) = bs(i + 1)
01016     END DO
01017
01018     RETURN
01019
01020 END FUNCTION getshapeparameters
01021
01022 !=====
01023
01024 !-----
01025 ! FUNCTION GET PHI FACTORS
01026 !-----
01027 FUNCTION getphifactors() RESULT(myValOut)
01028
01029     IMPLICIT NONE
01030
01031     REAL(sz), DIMENSION(4) :: myvalout
01032
01033     INTEGER :: i
01034
01035     DO i = 1, 4
01036         myvalout(i) = phis(i + 1)
01037     END DO
01038
01039     RETURN
01040
01041 END FUNCTION getphifactors
01042
01043 !=====
01044
01045 !-----
01046 ! SUBROUTINE SET ISOTACH RADII
01047 !-----
01048 SUBROUTINE setisotachradii(ir)
01049
01050     IMPLICIT NONE
01051
01052     REAL(SZ), DIMENSION(4), INTENT(IN) :: ir
01053
01054     radius(:) = ir(:)
01055
01056 END SUBROUTINE setisotachradii
01057
01058 !=====
01059
01060 !-----
01061 ! SUBROUTINE SET ISOTACH WIND SPEEDS
01062 !-----
01063 SUBROUTINE setisotachwindspeeds(vrQ)
01064
01065     IMPLICIT NONE
01066
01067     REAL(SZ), DIMENSION(4), INTENT(IN) :: vrQ
01068
01069     vrquadrant(:) = vrq(:)
01070
01071 END SUBROUTINE setisotachwindspeeds
01072
01073 !=====
01074
01075 !-----
01076 ! SUBROUTINE SET USE QUADRANT VR
01077 !-----
01078 SUBROUTINE setusequadrantvr(u)
01079
01080     IMPLICIT NONE
01081
01082     LOGICAL, INTENT(IN) :: u
01083
01084     usequadrantvr = u
01085
01086 END SUBROUTINE setusequadrantvr
01087
01088 !=====
01089
01090 !-----
01091 ! FUNCTION GET LATEST ANGLE
01092 !-----
01093 LOGICAL FUNCTION getusequadrantvr() RESULT(myValOut)

```

```

01094
01095     IMPLICIT NONE
01096
01097     myvalout = usequadrantvr
01098
01099     END FUNCTION getusequadrantvr
01100
01101 !=====
01102
01103 !-----
01104 ! F U N C T I O N   S P I N T E R P
01105 !-----
01106 !
01107 !-----
01124 REAL(sz) function spinterp(angle, dist, opt) result(myvalout)
01125
01126     IMPLICIT NONE
01127
01128     REAL(sz), INTENT(IN)           :: angle, dist
01129     INTEGER, INTENT(IN)            :: opt
01130     REAL(sz), DIMENSION(NPOINTS, 4) :: param
01131     REAL(sz)                       :: temp1, temp2
01132     REAL(sz)                       :: deltaangle
01133     INTEGER                        :: iquad
01134
01135     IF (opt == 1) THEN
01136         param = rmaxes4
01137     ELSE IF (opt == 2) THEN
01138         param = bs4
01139     ELSE IF (opt == 3) THEN
01140         param = vmb14
01141     END IF
01142
01143     IF (angle <= 45.0_sz) THEN
01144         iquad = 5
01145         deltaangle = 45.0_sz + angle
01146     ELSE IF (angle <= 135.0_sz) THEN
01147         iquad = 2
01148         deltaangle = angle - 45.0_sz
01149     ELSE IF (angle <= 225.0_sz) THEN
01150         iquad = 3
01151         deltaangle = angle - 135.0_sz
01152     ELSE IF (angle <= 315.0_sz) THEN
01153         iquad = 4
01154         deltaangle = angle - 225.0_sz
01155     ELSE IF (angle > 315.0_sz) THEN
01156         iquad = 5
01157         deltaangle = angle - 315.0_sz
01158     END IF
01159
01160     ! nearest neighbor weighted interpolation
01161     IF ( deltaangle < 1.0_sz ) THEN
01162         myvalout = interpr(param, iquad, dist)
01163     ELSE IF (deltaangle > 89.0_sz) THEN
01164         myvalout = interpr(param, iquad + 1, dist)
01165     ELSE
01166         temp1 = interpr(param, iquad, dist)
01167         temp2 = interpr(param, iquad + 1, dist)
01168         myvalout = (temp1 / deltaangle**2 + temp2 / (90.0 - deltaangle)**2) / &
01169             (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
01170     END IF
01171
01172     END FUNCTION spinterp
01173
01174 !=====
01175
01176 !-----
01177 ! F U N C T I O N   I N T E R P R
01178 !-----
01179 REAL(sz) function interpr(quadval, quadssel, quaddis) result(myvalout)
01180
01181     IMPLICIT NONE
01182
01183     REAL(sz), DIMENSION(NPOINTS, 4), INTENT(IN) :: quadval
01184     INTEGER, INTENT(IN)                        :: quadssel
01185     REAL(sz), INTENT(IN)                       :: quaddis
01186
01187     REAL(sz)                                   :: fac
01188     INTEGER                                    :: totalisot
01189
01190     totalisot = sum(quadflag4(quadssel, :))

```

```

01191     SELECT CASE(totalisot)
01192     CASE(1)
01193         myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))
01194     CASE(2)
01195         IF (quaddis > quadir4(quadsel, 1)) THEN
01196             myvalout = quadval(quadsel, 1)
01197         ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01198             fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01199             myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01200         ELSE
01201             myvalout = quadval(quadsel, 2)
01202         END IF
01203     CASE(3)
01204         IF (quaddis > quadir4(quadsel, 1)) THEN
01205             myvalout = quadval(quadsel, 1)
01206         ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01207             fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01208             myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01209         ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
01210             fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
01211             myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
01212         ELSE
01213             myvalout = quadval(quadsel, 3)
01214         END IF
01215     CASE(4)
01216         IF (quaddis > quadir4(quadsel, 1)) THEN
01217             myvalout = quadval(quadsel, 1)
01218         ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01219             fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01220             myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01221         ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
01222             fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
01223             myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
01224         ELSE IF (quaddis > quadir4(quadsel, 4)) THEN
01225             fac = (quaddis - quadir4(quadsel, 4)) / (quadir4(quadsel, 3) - quadir4(quadsel, 4))
01226             myvalout = quadval(quadsel, 3) * fac + quadval(quadsel, 4) * (1 - fac)
01227         ELSE
01228             myvalout = quadval(quadsel, 4)
01229         END IF
01230     CASE default
01231         ! For whatever reason if our algorithm fails, add the following
01232         ! line to avoid run-time errors
01233         myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))
01234         WRITE(*, *) "ERROR: InterpR failed in nws20get." !PV remove it of modify it?
01235     END SELECT
01236
01237     END FUNCTION interp
01238
01239 !=====
01240
01241 !-----
01242 ! F U N C T I O N   R M W
01243 !-----
01244 !
01245 !-----
01254 REAL(sz) function rmw(angle) result(myvalout)
01255
01256     IMPLICIT NONE
01257
01258     REAL(sz), INTENT(IN) :: angle
01259     INTEGER              :: basequadrant
01260     REAL(sz)             :: deltaangle
01261
01262     IF (angle <= 45.0_sz) THEN
01263         basequadrant = 5
01264         deltaangle = 45.0_sz + angle
01265     ELSE IF (angle <= 135.0_sz) THEN
01266         basequadrant = 2
01267         deltaangle = angle - 45.0_sz
01268     ELSE IF (angle <= 225.0_sz) THEN
01269         basequadrant = 3
01270         deltaangle = angle - 135.0_sz
01271     ELSE IF (angle <= 315.0_sz) THEN
01272         basequadrant = 4
01273         deltaangle = angle - 225.0_sz
01274     ELSE IF (angle > 315.0_sz) THEN
01275         basequadrant = 5
01276         deltaangle = angle - 315.0_sz
01277     END IF
01278
01279     ! nearest neighbor weighted interpolation

```

```

01280     IF ( deltaangle < 1.0_sz ) THEN
01281         myvalout = rmaxes(basequadrant) ! avoid div by zero
01282     ELSE IF ( deltaangle > 89.0_sz ) THEN
01283         myvalout = rmaxes(basequadrant + 1) ! avoid div by zero
01284     ELSE
01285         myvalout = (rmaxes(basequadrant) / deltaangle**2 + &
01286                     rmaxes(basequadrant + 1) / (90.0 - deltaangle)**2) / &
01287                     (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
01288     END IF
01289
01290     ! linearly interpolate
01291     !myValOut = (deltaAngle / 90.0_SZ) * &
01292     !           (rMaxes(baseQuadrant + 1) - rMaxes(baseQuadrant)) + &
01293     !           rMaxes(baseQuadrant)
01294
01295     END FUNCTION rmw
01296
01297 !=====
01298 !-----
01299 ! SUBROUTINE UVP
01300 !-----
01301 !
01302 !-----
01303 !
01304 SUBROUTINE uvp(lat, lon, uTrans, vTrans, u, v, p)
01305
01306     USE pahm_global, ONLY : windreduction, one2ten, deg2rad, rad2deg, mb2pa, kt2ms, nm2m, m2nm, rearth
01307
01308     IMPLICIT NONE
01309
01310     REAL(SZ), INTENT(IN) :: lat
01311     REAL(SZ), INTENT(IN) :: lon
01312     REAL(SZ), INTENT(IN) :: uTrans
01313     REAL(SZ), INTENT(IN) :: vTrans
01314
01315     REAL(SZ), INTENT(OUT) :: u
01316     REAL(SZ), INTENT(OUT) :: v
01317     REAL(SZ), INTENT(OUT) :: p
01318
01319     REAL(SZ) :: transSpdX !NWS8-style translation speed
01320     REAL(SZ) :: transSpdY !NWS8-style translation speed
01321
01322     REAL(SZ) :: dx
01323     REAL(SZ) :: dy
01324     REAL(SZ) :: dist
01325     REAL(SZ) :: rmx
01326     REAL(SZ) :: angle
01327     REAL(SZ) :: speed
01328     REAL(SZ) :: uf
01329     REAL(SZ) :: vf
01330     REAL(SZ) :: percentCoriolis
01331     REAL(SZ) :: speedAtRMax
01332     REAL(SZ) :: vMaxFactor
01333
01334     !-----
01335     ! Calculate distance and angle between eye of hurricane
01336     ! and input nodal point
01337     !-----
01338     dx = deg2rad * rearth * (lon - clon) * cos(deg2rad * clat)
01339     dy = deg2rad * rearth * (lat - clat)
01340     dist = sqrt(dx * dx + dy * dy)
01341
01342     !-----
01343     ! Handle special case at eye of hurricane
01344     ! in eye velocity is zero not translational velocity
01345     !-----
01346     IF (dist < 1.0_sz) THEN
01347         u = 0.0_sz
01348         v = 0.0_sz
01349         p = pc * mb2pa
01350
01351         RETURN
01352     END IF
01353
01354     dist = m2nm * dist
01355
01356     angle = 360.0_sz + rad2deg * atan2(dx, dy)
01357     IF (angle > 360.0_sz) angle = angle - 360.0_sz
01358
01359     latestangle = angle
01360     rmx = rmw(angle)

```

```

01381     latestrmax = rmx
01382
01383     !-----
01384     ! Compute (u,v) wind velocity components from the
01385     ! asymmetric hurricane vortex.
01386     !
01387     ! Note: the vortex winds are valid at the top of the
01388     ! surface layer, so reduce the winds to the surface.
01389     ! Also convert the winds from max sustained 1-minute
01390     ! averages to 10-minute averages for the storm surge
01391     ! model.
01392     !-----
01393     percentcoriolis = 1.0_sz
01394     speed = sqrt((vmax * kt2ms)**2 * (rmx / dist)**b * exp(1.0_sz - (rmx / dist)**b) + &
01395              (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2) &
01396              - nm2m * dist * percentcoriolis * corio / 2.0_sz
01397
01398     ! calculate the wind speed (m/s) at rMax, using
01399     ! equation that includes full coriolis
01400     speedatrmx = sqrt((vmax * kt2ms)**2 * exp(0.0_sz) + &
01401              (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2) &
01402              - nm2m * dist * percentcoriolis * corio / 2.0_sz
01403
01404     ! calculate a factor to place the velocity profile so that
01405     ! it hits vMax
01406     vmaxfactor = vmax * kt2ms / speedatrmx
01407
01408     ! jgf20111007: Calculate NWS8-like translation speed
01409     transspdx = (abs(speed / speedatrmx)) * utrans * kt2ms
01410     transspdy = (abs(speed / speedatrmx)) * vtrans * kt2ms
01411
01412     speed = speed * vmaxfactor
01413
01414     ! now reduce the wind speed to the surface
01415     speed = speed * windreduction
01416
01417     u = -speed * cos(deg2rad * angle)
01418     v = speed * sin(deg2rad * angle)
01419
01420     ! Alter wind direction by adding a frictional inflow angle
01421     CALL rotate(u, v, fang(dist, rmx), clat, uf, vf)
01422     u = uf
01423     v = vf
01424     !
01425     ! jgf20111007: Add in the translation velocity
01426     u = u + transspdx
01427     v = v + transspdy
01428     !
01429     ! convert from 1 minute averaged winds to 10 minute averaged
01430     ! winds for use in ADCIRC
01431     u = u * one2ten
01432     v = v * one2ten
01433
01434     ! Compute surface pressure from asymmetric hurricane vortex
01435     p = mb2pa * (pc + (pn - pc) * exp(-(rmx / dist)**b))
01436
01437     ! cut off the vortex field after 401nm
01438     ! TODO: 401nm should be replaced with something less
01439     ! arbitrary ... and find a better way to blend this
01440     !IF ( dist > 401.0_SZ ) THEN
01441     !   u = 0.0_SZ
01442     !   v = 0.0_SZ
01443     !   p = MB2PA * pn
01444     !END IF
01445
01446     END SUBROUTINE uvp
01447
01448     !=====
01449
01450
01451     !-----
01452     ! S U B R O U T I N E   U V P R
01453     !-----
01454     !
01455     !-----
01481
01482     SUBROUTINE uvpr(iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, &
01483              uTrans, vTrans, geof, u, v, p)
01484
01485     USE pahm_global, ONLY : windreduction, one2ten, deg2rad, mb2pa, kt2ms, nm2m
01486
01487     IMPLICIT NONE

```

```

01488
01489 REAL(SZ), INTENT(IN) :: iDist
01490 REAL(SZ), INTENT(IN) :: iAngle
01491 REAL(SZ), INTENT(IN) :: iRmx
01492 REAL(SZ), INTENT(IN) :: iRmxTrue
01493 REAL(SZ), INTENT(IN) :: iB
01494 REAL(SZ), INTENT(IN) :: iVm
01495 REAL(SZ), INTENT(IN) :: iPhi
01496 REAL(SZ), INTENT(IN) :: uTrans
01497 REAL(SZ), INTENT(IN) :: vTrans
01498 INTEGER, INTENT(IN) :: geof
01499
01500 REAL(SZ), INTENT(OUT) :: u
01501 REAL(SZ), INTENT(OUT) :: v
01502 REAL(SZ), INTENT(OUT) :: p
01503
01504 REAL(SZ) :: transSpdX !NWS8-style translation speed
01505 REAL(SZ) :: transSpdY !NWS8-style translation speed
01506 REAL(SZ) :: rmx
01507 REAL(SZ) :: speed
01508 REAL(SZ) :: uf
01509 REAL(SZ) :: vf
01510 REAL(SZ) :: percentCoriolis
01511
01512 rmx = irmx
01513 b = ib
01514 vmax = ivm
01515 phi = iphi
01516
01517 !-----
01518 ! Handle special case at eye of hurricane
01519 ! in eye velocity is zero not translational velocity
01520 !-----
01521 IF (idist < 1.0_sz) THEN
01522   u = 0.0_sz
01523   v = 0.0_sz
01524   p = pc * mb2pa
01525
01526   RETURN
01527 END IF
01528
01529 !-----
01530 ! Compute (u, v) wind velocity components from the
01531 ! asymmetric hurricane vortex.
01532 !
01533 ! Note: the vortex winds are valid at the top of the
01534 ! surface layer, so reduce the winds to the surface.
01535 ! Also convert the winds from max sustained 1-minute
01536 ! averages to 10-minute averages for the storm surge
01537 ! model.
01538 !-----
01539 percentcoriolis = 1.0_sz
01540 ! Jie 2014.07
01541 IF (geof == 1) THEN
01542   speed = sqrt(((vmax * kt2ms)**2 + vmax * kt2ms * rmx * nm2m * percentcoriolis * corio) * &
01543               (rmx / idist)**b * exp(phi * (1.0_sz - (rmx / idist)**b)) + &
01544               (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) - &
01545               nm2m * idist * percentcoriolis * corio / 2.0_sz
01546 ELSE
01547   speed = sqrt((vmax * kt2ms)**2 * (rmx / idist)**b * exp(1.0_sz - (rmx / idist)**b) + &
01548               (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) - &
01549               nm2m * idist * percentcoriolis * corio / 2.0_sz
01550 ENDIF
01551
01552 ! jgf20111007: Calculate NWS8-like translation speed
01553 transspdx = (abs(speed / (vmax * kt2ms))) * utrans * kt2ms
01554 transspdy = (abs(speed / (vmax * kt2ms))) * vtrans * kt2ms
01555
01556 ! now reduce the wind speed to the surface
01557 speed = speed * windreduction
01558
01559 u = -speed * cos(deg2rad * iangle)
01560 v = speed * sin(deg2rad * iangle)
01561 !
01562 ! Alter wind direction by adding a frictional inflow angle
01563 CALL rotate(u, v, fang(idist, irmxtrue), clat, uf, vf)
01564 u = uf
01565 v = vf
01566 !
01567 ! jgf20111007: Add in the translation velocity
01568 u = u + transspdx

```

```

01569     v = v + transspdy
01570
01571     ! convert from 1 minute averaged winds to 10 minute averaged
01572     ! winds for use in ADCIRC
01573     u = u * one2ten
01574     v = v * one2ten
01575
01576     ! Compute surface pressure from asymmetric hurricane vortex
01577     IF (geof == 1) THEN
01578         p = mb2pa * (pc + (pn - pc) * exp( - phi * (rmx / idist)**b))
01579     ELSE
01580         p = mb2pa * (pc + (pn - pc) * exp(-(rmx / idist)**b))
01581     ENDIF
01582
01583     ! cut off the vortex field after 401nm
01584     ! TODO: 401nm should be replaced with something less
01585     ! arbitrary ... and find a better way to blend this
01586     !if ( dist > 401.0_SZ ) then
01587     !u = 0.0_SZ
01588     !v = 0.0_SZ
01589     !p = MB2PA * pn
01590     !endif
01591
01592     END SUBROUTINE uvpr
01593
01594     !=====
01595
01596     !-----
01597     ! FUNCTION FANG
01598     !-----
01599     !
01600     !-----
01601
01602     REAL(sz) function fang(r, rmx) result(myvalout)
01603
01604     IMPLICIT NONE
01605
01606     REAL(sz), INTENT(IN) :: r
01607     REAL(sz), INTENT(IN) :: rmx
01608
01609     IF ((0.0_sz <= r) .AND. (r < rmx)) THEN
01610         myvalout = 10.0_sz * r / rmx
01611     ELSE IF ((rmx <= r) .AND. (r < 1.0_sz * rmx)) THEN
01612         myvalout = 10.0_sz + 75.0_sz * (r / rmx - 1.0_sz)
01613     ELSE IF (r >= 1.0_sz * rmx) THEN
01614         myvalout = 25.0_sz
01615     ELSE
01616         myvalout = 0.0_sz
01617     END IF
01618
01619     END FUNCTION fang
01620
01621     !=====
01622
01623     !-----
01624     ! SUBROUTINE ROTATE
01625     !-----
01626     !
01627     !-----
01628
01629     SUBROUTINE rotate(x, y, angle, whichWay, xr, yr)
01630
01631     USE pahm_global, ONLY : deg2rad
01632
01633     IMPLICIT NONE
01634
01635     REAL(SZ), INTENT(IN) :: x
01636     REAL(SZ), INTENT(IN) :: y
01637     REAL(SZ), INTENT(IN) :: angle
01638     REAL(SZ), INTENT(IN) :: whichWay
01639
01640     REAL(SZ), INTENT(OUT) :: xr
01641     REAL(SZ), INTENT(OUT) :: yr
01642
01643     REAL(SZ) :: A, cosA, sinA
01644
01645     a = sign(1.0_sz, whichway) * deg2rad * angle
01646     cosa = cos(a)
01647     sina = sin(a)
01648
01649     xr = x * cosa - y * sina
01650     yr = x * sina + y * cosa
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670

```

```

01671     END SUBROUTINE rotate
01672
01673 !=====
01674
01675 !-----
01676 ! FUNCTION GET LATEST RMAX
01677 !-----
01678 REAL(sz) function getlatestrmax() result(myvalout)
01679
01680     IMPLICIT NONE
01681
01682     myvalout = latestrmax
01683
01684     END FUNCTION getlatestrmax
01685
01686 !=====
01687
01688 !-----
01689 ! FUNCTION GET LATEST ANGLE
01690 !-----
01691 REAL(sz) function getlatestangle() result(myvalout)
01692
01693     IMPLICIT NONE
01694
01695     myvalout = latestangle
01696
01697     END FUNCTION getlatestangle
01698
01699 !=====
01700
01701 !-----
01702 ! FUNCTION VH WITH CORI FULL
01703 !-----
01704 ! Jie 2013.02 Modified to use the full gradient wind eq.
01705 !-----
01706 REAL(sz) function vhwithcorifull(testrmax) result(myvalout)
01707
01708     USE pahm_global, ONLY : nm2m, kt2ms, ms2kt
01709
01710     IMPLICIT NONE
01711
01712     REAL(sz), INTENT(IN) :: testrmax
01713
01714     REAL(sz) :: thisvr ! the radial wind speed we've been given
01715     REAL(sz) :: vh
01716
01717     !-----
01718     ! func(x = rMax) = vh - vr
01719     !-----
01720     IF (getusequadrantvr() .EQV. .true.) THEN
01721         thisvr = vrquadrant(quad)
01722     ELSE
01723         thisvr = vr
01724     END IF
01725
01726     ! Jie 2013.02
01727     vh = ms2kt * (sqrt(((vmax * kt2ms)**2 + vmax * kt2ms * testrmax * nm2m * corio) * &
01728         (testrmax / radius(quad))**b * &
01729         exp(phi * (1.0_sz - (testrmax / radius(quad))**b)) + &
01730         (nm2m * radius(quad) * corio / 2.0_sz)**2) - &
01731         nm2m * radius(quad) * corio / 2.0_sz)
01732
01733     myvalout = vh - thisvr
01734
01735     RETURN
01736
01737     END FUNCTION vhwithcorifull
01738
01739 !=====
01740
01741 !-----
01742 ! FUNCTION VH WITH CORI
01743 !-----
01744 REAL(sz) function vhwithcori(testrmax) result(myvalout)
01745
01746     USE pahm_global, ONLY : nm2m, kt2ms, ms2kt
01747
01748     IMPLICIT NONE
01749
01750     REAL(sz), INTENT(IN) :: testrmax

```



```

01775
01776 REAL(sz)          :: thisvr ! the radial wind speed we've been given
01777 REAL(sz)          :: vh
01778
01779 !-----
01780 ! func(x = rMax) = vh - vr
01781 !-----
01782 IF (getusequadrantvr() .EQV. .true.) THEN
01783   thisvr = vrquadrant(quad)
01784 ELSE
01785   thisvr = vr
01786 END IF
01787
01788 vh = ms2kt * (sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b *
01789                  exp(1.0_sz - (testrmax / radius(quad))**b) +
01790                  (nm2m * radius(quad) * corio / 2.0_sz)**2) -
01791                  nm2m * radius(quad) * corio / 2.0_sz)
01792
01793 myvalout = vh - thisvr
01794
01795 RETURN
01796
01797 END FUNCTION vhwthcori
01798
01799 !=====
01800
01801 !-----
01802 ! FUNCTION V H N O C O R I
01803 !-----
01804 REAL(sz) function vhnocori(testrmax) result(myvalout)
01805
01806   USE pahm_global, ONLY : kt2ms, ms2kt
01807
01808   IMPLICIT NONE
01809
01810   REAL(sz), INTENT(IN) :: testrmax
01811
01812   REAL(sz) :: thisvr ! the radial wind speed we've been given
01813
01814   IF (getusequadrantvr() .EQV. .true.) THEN
01815     thisvr = vrquadrant(quad)
01816   ELSE
01817     thisvr = vr
01818   END IF
01819
01820   myvalout = abs(ms2kt * sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b *
01821                             exp(1 - (testrmax / radius(quad))**b))) - thisvr
01822
01823   RETURN
01824
01825 END FUNCTION vhnocori
01826
01827 !=====
01828
01829 !-----
01830 ! FUNCTION F I N D R O O T
01831 !-----
01832 !-----
01833 !-----
01834 !-----
01845 REAL(sz) function findroot(func, x1, x2, dx, a, b) result(myroot)
01846 !PV Need to check for the x2 variable is not used anywhere next
01847 IMPLICIT NONE
01848
01849 REAL(sz), EXTERNAL :: func
01850 REAL(sz), INTENT(IN) :: x1, x2 ! Search interval [x1,x2]
01851 REAL(sz), INTENT(IN) :: dx ! Marching increment
01852 REAL(sz), INTENT(OUT) :: a, b ! x values that bracket root
01853
01854 INTEGER, PARAMETER :: itermax = 400 ! Max # of iterations
01855 INTEGER :: iter ! iteration counter
01856 REAL(sz) :: fa, fb ! function values f(x)
01857
01858 ! Initialize left side of interval
01859 a = x1
01860 fa = func(a)
01861
01862 ! March along interval until root is found
01863 ! or solution diverges.
01864 myroot = a
01865 DO iter = 1, itermax
01866   b = x1 + iter * dx
01867   fb = func(b)

```

```
01868
01869      ! Check progress
01870      IF ((fa * fb < 0.0_sz) .OR. (abs(fb) > abs(fa))) THEN
01871          ! Assign root
01872          IF (abs(fb) > abs(fa)) THEN
01873              myroot = a
01874          ELSE
01875              myroot = b
01876          END IF
01877      EXIT
01878      END IF
01879
01880      ! Move right search interval values to left side
01881      ! for next iteration.
01882      a = b
01883      fa = fb
01884      END DO
01885
01886      IF (iter >= itermax) THEN
01887          print *, "FUNCTION FindRoot: exceeded max # of iterations"
01888          myroot = -99999.0
01889      END IF
01890
01891      RETURN
01892
01893      END FUNCTION findroot
01894
01895      !=====
01896      !=====
01897
01898      END MODULE pahm_vortex
```

Index

a

- csv_module, 8
- csv_parameters, 10

add

- csv_module::csv_file, 180

add_cell

- csv_module::csv_file, 181

add_matrix

- csv_module::csv_file, 181

add_vector

- csv_module::csv_file, 181

agrid

- pahm_mesh, 41

airdensity

- pahm_global, 21

allmessage_1

- pahm_messages, 45
- pahm_messages::allmessage, 159

allmessage_2

- pahm_messages, 45
- pahm_messages::allmessage, 159

allocatenodalandelementalarrays

- pahm_mesh, 39

allocbtrstruct

- parwind, 90

allochollstruct

- parwind, 90

an

- csv_module, 8
- csv_parameters, 10

arraycopydouble

- sortutils, 101
- sortutils::arraycopy, 160

arraycopyint

- sortutils, 101
- sortutils::arraycopy, 162

arraycopysingle

- sortutils, 102
- sortutils::arraycopy, 162

arrayequaldouble

- sortutils, 102
- sortutils::arrayequal, 164

arrayequalint

- sortutils, 102
- sortutils::arrayequal, 164

arrayequalsingle

- sortutils, 103
- sortutils::arrayequal, 165

arthdouble

- sortutils, 103
- sortutils::arth, 166

arthint

- sortutils, 104
- sortutils::arth, 166

arthsingle

- sortutils, 104
- sortutils::arth, 167

b

- pahm_vortex, 84

backgroundatmpress

- pahm_global, 21

base_netcdfcheckerr

- pahm_netcdfio, 54

basee

- pahm_global, 22

basin

- parwind::besttrackdata_t, 169
- parwind::hollanddata_t, 198

begdate

- pahm_global, 22

begdatespecified

- pahm_global, 22

begdatetime

- pahm_global, 22

begday

- pahm_global, 22

beghour

- pahm_global, 22

begmin

- pahm_global, 23

begmonth

- pahm_global, 23

begsec

- pahm_global, 23

begsimspecified

- pahm_global, 23

begsimtime

- pahm_global, 23

begtime

- pahm_global, 23

begyear

- pahm_global, 24

besttrackdata

- parwind, 99

besttrackfilename

- pahm_global, 24

besttrackfilenamespecified

- pahm_global, 24

bladjustfac

- pahm_global, 24

blank

- pahm_sizes, 64
- bs
 - pahm_vortex, 84
- bs4
 - pahm_vortex, 85
- calcintensitychange
 - pahm_vortex, 68
- calcrmaxes
 - pahm_vortex, 69
- calcrmaxesfull
 - pahm_vortex, 69
- casttime
 - parwind::hollanddata_t, 198
- casttype
 - parwind::hollanddata_t, 199
- cell
 - csv_module, 8
- character
 - csv_module, 8
 - csv_module::csv_file, 184
- charunique
 - utilities, 134
- checkcontrolfileinputs
 - utilities, 135
- chunk_size
 - csv_module::csv_file, 184
- clat
 - pahm_vortex, 85
- clon
 - pahm_vortex, 85
- close
 - csv_module::csv_file, 181
- closelogfile
 - pahm_messages, 45
- closetol
 - utilities, 158
- cnttimebegin
 - pahm_drivermod, 18
- cnttimeend
 - pahm_drivermod, 18
- comparedoublereals
 - pahm_sizes, 61
 - pahm_sizes::comparereals, 175
- comparesinglereals
 - pahm_sizes, 62
 - pahm_sizes::comparereals, 176
- controlfilename
 - pahm_global, 24
- convlon
 - utilities, 136
- corio
 - pahm_vortex, 85
- cpptogeo_1d
 - utilities, 136
 - utilities::cpptogeo, 177
- cpptogeo_scalar
 - utilities, 137
 - utilities::cpptogeo, 178
- cprdt
 - parwind::hollanddata_t, 199
- cpress
 - parwind::hollanddata_t, 199
- csv_data
 - csv_module::csv_file, 184
- csv_get_value
 - csv_module::csv_file, 181
- csv_module, 6
 - a, 8
 - an, 8
 - cell, 8
 - character, 8
 - csv_type_double, 8
 - csv_type_integer, 9
 - csv_type_logical, 9
 - csv_type_string, 9
 - initialize_csv_file, 7
 - integer, 9
 - logical, 9
 - real, 9
 - string, 10
- csv_module.F90, 225, 227
- csv_module::csv_file, 179
 - add, 180
 - add_cell, 181
 - add_matrix, 181
 - add_vector, 181
 - character, 184
 - chunk_size, 184
 - close, 181
 - csv_data, 184
 - csv_get_value, 181
 - delimiter, 184
 - destroy, 181
 - enclose_all_in_quotes, 184
 - enclose_strings_in_quotes, 184
 - get, 181
 - get_character_column, 181
 - get_column, 182
 - get_csv_data_as_str, 182
 - get_csv_string_column, 182
 - get_header, 182
 - get_header_csv_str, 182
 - get_header_str, 182
 - get_integer_column, 182
 - get_logical_column, 182
 - get_real_column, 183
 - header, 184

- icol, 185
- initialize, 183
- iunit, 185
- logical_false_string, 185
- logical_true_string, 185
- n_cols, 185
- n_rows, 185
- next_row, 183
- open, 183
- quotation, 185
- quote, 185
- read, 183
- read_line_from_file, 183
- tokenize, 183
- variable_types, 183
- csv_module::csv_string, 186
- str, 186
- csv_parameters, 10
 - a, 10
 - an, 10
 - default_int_fmt, 10
 - default_real_fmt, 11
 - integer, 11
 - length, 11
 - max_integer_str_len, 11
 - max_real_str_len, 11
 - maximum, 11
 - number, 11
 - of, 12
 - real, 12
 - string, 12
- csv_parameters.F90, 243, 244
- csv_type_double
 - csv_module, 8
- csv_type_integer
 - csv_module, 9
- csv_type_logical
 - csv_module, 9
- csv_type_string
 - csv_module, 9
- csv_utilities, 12
 - expand_vector, 12
 - sort_ascending, 13
 - unique, 13
- csv_utilities.F90, 244, 245
- cyclenum
 - parwind::besttrackdata_t, 169
- cynum
 - parwind::besttrackdata_t, 169
- datetime2string
 - timedateutils, 115
- day
 - parwind::besttrackdata_t, 169
- parwind::hollanddata_t, 199
- dayofyear
 - timedateutils, 116
- dayofyeartogreg
 - timedateutils, 116
- deallocbtrstruct
 - parwind, 91
- deallochoollstruct
 - parwind, 91
- debug
 - pahm_messages, 51
- def_ncnam_pres
 - pahm_global, 24
- def_ncnam_wndx
 - pahm_global, 25
- def_ncnam_wndy
 - pahm_global, 25
- default_int_fmt
 - csv_parameters, 10
- default_real_fmt
 - csv_parameters, 11
- defv_atmpress
 - pahm_global, 25
- defv_bladjustfac
 - pahm_global, 25
- defv_gravity
 - pahm_global, 25
- defv_rhoair
 - pahm_global, 25
- defv_rhowater
 - pahm_global, 26
- deg2rad
 - pahm_global, 26
- delimiter
 - csv_module::csv_file, 184
- destroy
 - csv_module::csv_file, 181
- dev_doc.md, 225
- dir
 - parwind::besttrackdata_t, 170
- doxy_sample.F90, 247
- dp
 - pahm_mesh, 41
- drealcan
 - utilities, 138
- driver_mod.F90, 248, 249
- dtg
 - parwind::besttrackdata_t, 170
 - parwind::hollanddata_t, 199
- dvalstr
 - utilities, 139
- echo
 - pahm_messages, 51

- elapsedsecs
 - timedateutils, 117
- enclose_all_in_quotes
 - csv_module::csv_file, 184
- enclose_strings_in_quotes
 - csv_module::csv_file, 184
- enddate
 - pahm_global, 26
- enddatespecified
 - pahm_global, 26
- enddatetime
 - pahm_global, 26
- endday
 - pahm_global, 26
- endhour
 - pahm_global, 27
- endmin
 - pahm_global, 27
- endmonth
 - pahm_global, 27
- endsec
 - pahm_global, 27
- endsimspecified
 - pahm_global, 27
- endsimtime
 - pahm_global, 27
- endtime
 - pahm_global, 28
- endyear
 - pahm_global, 28
- error
 - pahm_messages, 51
- ew
 - parwind::besttrackdata_t, 170
- expand_vector
 - csv_utilities, 12
- eye
 - parwind::besttrackdata_t, 170
- fang
 - pahm_vortex, 70
- fcstinc
 - parwind::hollanddata_t, 199
- filefound
 - pahm_netcdfio::filedata_t, 187
- filename
 - pahm_netcdfio::filedata_t, 188
 - parwind::besttrackdata_t, 170
 - parwind::hollanddata_t, 199
- filereccounter
 - pahm_netcdfio::filedata_t, 188
- findroot
 - pahm_vortex, 70
- firstgregdate
 - timedateutils, 130
- firstgregtime
 - timedateutils, 130
- fitrmaxes
 - pahm_vortex, 71
- fitrmaxes4
 - pahm_vortex, 71
- fixnearwholedoublereal
 - pahm_sizes, 62
 - pahm_sizes::fixnearwholereal, 189
- fixnearwholesinglereal
 - pahm_sizes, 63
 - pahm_sizes::fixnearwholereal, 189
- fnamelen
 - pahm_sizes, 64
- geotocpp_1d
 - utilities, 140
 - utilities::geotocpp, 191
- geotocpp_scalar
 - utilities, 140
 - utilities::geotocpp, 191
- get
 - csv_module::csv_file, 181
- get_character_column
 - csv_module::csv_file, 181
- get_column
 - csv_module::csv_file, 182
- get_csv_data_as_str
 - csv_module::csv_file, 182
- get_csv_string_column
 - csv_module::csv_file, 182
- get_header
 - csv_module::csv_file, 182
- get_header_csv_str
 - csv_module::csv_file, 182
- get_header_str
 - csv_module::csv_file, 182
- get_integer_column
 - csv_module::csv_file, 182
- get_logical_column
 - csv_module::csv_file, 182
- get_real_column
 - csv_module::csv_file, 183
- gethollandfields
 - parwind, 92, 93
- getlatestangle
 - pahm_vortex, 72
- getlatestrmax
 - pahm_vortex, 72
- getlinerecord
 - utilities, 141
- getlocandratio
 - utilities, 142

- getphifactors
 - pahm_vortex, [72](#)
- getprogramcmdlargs
 - pahm_drivermod, [15](#)
- getrmaxes
 - pahm_vortex, [72](#)
- getshapeparameter
 - pahm_vortex, [72](#)
- getshapeparameters
 - pahm_vortex, [72](#)
- gettimeconvsec
 - timedateutils, [118](#)
- getusequadrantvr
 - pahm_vortex, [73](#)
- getvmaxesbl
 - pahm_vortex, [73](#)
- global.F90, [251](#), [253](#)
- gravity
 - pahm_global, [28](#)
- gregtojulday2
 - timedateutils, [119](#)
 - timedateutils::gregtojulday, [193](#)
- gregtojuldayisec
 - timedateutils, [120](#)
 - timedateutils::gregtojulday, [194](#)
- gregtojuldayrsec
 - timedateutils, [121](#)
 - timedateutils::gregtojulday, [195](#)
- gusts
 - parwind::besttrackdata_t, [170](#)
- header
 - csv_module::csv_file, [184](#)
- hour
 - parwind::besttrackdata_t, [170](#)
 - parwind::hollanddata_t, [199](#)
- hp
 - pahm_sizes, [64](#)
- icol
 - csv_module::csv_file, [185](#)
- icompxchg
 - sortutils.F90, [352](#)
- icpress
 - parwind::hollanddata_t, [200](#)
- ics
 - pahm_mesh, [41](#)
- ilat
 - parwind::hollanddata_t, [200](#)
- ilon
 - parwind::hollanddata_t, [200](#)
- imissv
 - pahm_sizes, [64](#)
- indexxdouble
 - sortutils, [105](#)
- sortutils::indexx, [203](#)
- indexxint
 - sortutils, [105](#)
 - sortutils::indexx, [204](#)
- indexxint8
 - sortutils, [106](#)
 - sortutils::indexx, [205](#)
- indexxsingle
 - sortutils, [106](#)
 - sortutils::indexx, [206](#)
- indexxstring
 - sortutils, [107](#)
 - sortutils::indexx, [207](#)
- info
 - pahm_messages, [52](#)
- initadcircnetcdfoutfile
 - pahm_netcdfio, [55](#)
- initialize
 - csv_module::csv_file, [183](#)
- initialize_csv_file
 - csv_module, [7](#)
- initialized
 - pahm_netcdfio::filedata_t, [188](#)
 - pahm_netcdfio::timedata_t, [224](#)
- initials
 - parwind::besttrackdata_t, [170](#)
- initlogging
 - pahm_messages, [45](#)
- int1
 - pahm_sizes, [64](#)
- int16
 - pahm_sizes, [64](#)
- int2
 - pahm_sizes, [64](#)
- int4
 - pahm_sizes, [65](#)
- int8
 - pahm_sizes, [65](#)
- integer
 - csv_module, [9](#)
 - csv_parameters, [11](#)
- interpr
 - pahm_vortex, [73](#)
- intlat
 - parwind::besttrackdata_t, [171](#)
- intlon
 - parwind::besttrackdata_t, [171](#)
- intmslp
 - parwind::besttrackdata_t, [171](#)
- intpouter
 - parwind::besttrackdata_t, [171](#)
- intrad1
 - parwind::besttrackdata_t, [171](#)
- intrad2

- parwind::besttrackdata_t, 171
- intrad3
 - parwind::besttrackdata_t, 171
- intrad4
 - parwind::besttrackdata_t, 171
- intrmw
 - parwind::besttrackdata_t, 172
- introuter
 - parwind::besttrackdata_t, 172
- intscan
 - utilities, 142
- intspeed
 - parwind::besttrackdata_t, 172
- intvalstr
 - utilities, 143
- intvmax
 - parwind::besttrackdata_t, 172
- ip
 - pahm_sizes, 65
- irmw
 - parwind::hollanddata_t, 200
- irrp
 - parwind::hollanddata_t, 200
- ismeshok
 - pahm_mesh, 41
- ispeed
 - parwind::hollanddata_t, 200
- iunit
 - csv_module::csv_file, 185
- joindate
 - timedateutils, 122
- juldaytogreg
 - timedateutils, 123
- kt2ms
 - pahm_global, 28
- lat
 - parwind::besttrackdata_t, 172
 - parwind::hollanddata_t, 200
- latestangle
 - pahm_vortex, 85
- latestrmax
 - pahm_vortex, 85
- leapyear
 - timedateutils, 124
- length
 - csv_parameters, 11
- llong
 - pahm_sizes, 65
- loaded
 - parwind::besttrackdata_t, 172
 - parwind::hollanddata_t, 200
- loadintvar
 - utilities, 144
- loadlogvar
 - utilities, 145
- loadrealvar
 - utilities, 146
- logfilename
 - pahm_global, 28
- logfileopened
 - pahm_messages, 52
- logical
 - csv_module, 9
- logical_false_string
 - csv_module::csv_file, 185
- logical_true_string
 - csv_module::csv_file, 185
- loginitcalled
 - pahm_messages, 52
- loglevelnames
 - pahm_messages, 52
- logmessage_1
 - pahm_messages, 46
 - pahm_messages::logmessage, 209
- logmessage_2
 - pahm_messages, 46
 - pahm_messages::logmessage, 209
- lon
 - parwind::besttrackdata_t, 172
 - parwind::hollanddata_t, 201
- long
 - pahm_sizes, 65
- lun_btrk
 - pahm_global, 28
- lun_btrk1
 - pahm_global, 29
- lun_ctrl
 - pahm_global, 29
- lun_inp
 - pahm_global, 29
- lun_inp1
 - pahm_global, 29
- lun_log
 - pahm_global, 29
- lun_out
 - pahm_global, 29
- lun_out1
 - pahm_global, 30
- lun_screen
 - pahm_global, 30
- m2nm
 - pahm_global, 30
- mainpage.md, 225
- max_integer_str_len
 - csv_parameters, 11

max_real_str_len
 csv_parameters, 11
maxfacenodes
 pahm_mesh, 42
maximum
 csv_parameters, 11
maxseas
 parwind::besttrackdata_t, 172
mb2kpa
 pahm_global, 30
mb2pa
 pahm_global, 30
mdbegsimtime
 pahm_global, 30
mdendsimtime
 pahm_global, 31
mdjdate
 timedateutils, 131
mdjoffset
 timedateutils, 131
mdjtime
 timedateutils, 131
mdoutdt
 pahm_global, 31
mesh.F90, 257, 258
meshfileform
 pahm_global, 31
meshfilename
 pahm_global, 31
meshfilenamespecified
 pahm_global, 31
meshfiletype
 pahm_global, 31
messages.F90, 262, 264
messagesources
 pahm_messages, 52
model.md, 225
modeldate
 timedateutils, 131
modeltime
 timedateutils, 131
modeltype
 pahm_global, 32
modjuldate
 timedateutils, 131
modjultime
 timedateutils, 131
month
 parwind::besttrackdata_t, 173
 parwind::hollanddata_t, 201
monthdays
 timedateutils, 125
ms2kt
 pahm_global, 32
mytime
 pahm_netcdfio, 60
n_cols
 csv_module::csv_file, 185
n_rows
 csv_module::csv_file, 185
nbtrfiles
 pahm_global, 32
nbyte
 pahm_sizes, 65
ncdeflate
 pahm_global, 32
ncdlevel
 pahm_global, 32
ncshuffle
 pahm_global, 32
ncvarnam_pres
 pahm_global, 33
ncvarnam_wndx
 pahm_global, 33
ncvarnam_wndy
 pahm_global, 33
ne
 pahm_mesh, 42
NetCDFCheckErr
 netcdfio-nems.F90, 269
 netcdfio-orig.F90, 281
 netcdfio.F90, 292
netcdfio-nems.F90, 268, 269
 NetCDFCheckErr, 269
netcdfio-orig.F90, 280, 281
 NetCDFCheckErr, 281
netcdfio.F90, 291, 293
 NetCDFCheckErr, 292
netcdfterminate
 pahm_netcdfio, 56
newadcircnetcdfoutfile
 pahm_netcdfio, 57
newvortex
 pahm_vortex, 74
newvortexfull
 pahm_vortex, 74
next_row
 csv_module::csv_file, 183
nfn
 pahm_mesh, 42
nm
 pahm_mesh, 42
nm2m
 pahm_global, 33
noutdt
 pahm_global, 33
np

- pahm_mesh, 42
- npoints
 - pahm_vortex, 86
- nquads
 - pahm_vortex, 86
- ns
 - parwind::besttrackdata_t, 173
- nscreen
 - pahm_messages, 53
- number
 - csv_parameters, 11
- numrec
 - parwind::besttrackdata_t, 173
 - parwind::hollanddata_t, 201
- of
 - csv_parameters, 12
- offfirstgregday
 - timedateutils, 132
- offmodeljulday
 - timedateutils, 132
- offmodjulday
 - timedateutils, 132
- offunixjulday
 - timedateutils, 132
- omega
 - pahm_global, 33
- one2ten
 - pahm_global, 34
- open
 - csv_module::csv_file, 183
- openfileforread
 - utilities, 146
- openlogfile
 - pahm_messages, 46
- outdt
 - pahm_global, 34
- outfilename
 - pahm_global, 34
- outfilenamespecified
 - pahm_global, 34
- pahm
 - pahm.F90, 304
- pahm.F90, 303, 305
 - pahm, 304
- pahm_drivermod, 14
 - cnttimebegin, 18
 - cnttimeend, 18
 - getprogramcmdlargs, 15
 - pahm_finalize, 15
 - pahm_init, 16
 - pahm_run, 17
- pahm_finalize
 - pahm_drivermod, 15
- pahm_global, 19
 - airdensity, 21
 - backgroundatmpress, 21
 - basee, 22
 - begdate, 22
 - begdatespecified, 22
 - begdatetime, 22
 - begday, 22
 - beghour, 22
 - begmin, 23
 - begmonth, 23
 - begsec, 23
 - begsimspecified, 23
 - begsimtime, 23
 - begtime, 23
 - begyear, 24
 - besttrackfilename, 24
 - besttrackfilenamespecified, 24
 - bladjustfac, 24
 - controlfilename, 24
 - def_ncnam_pres, 24
 - def_ncnam_wndx, 25
 - def_ncnam_wndy, 25
 - defv_atmpress, 25
 - defv_bladjustfac, 25
 - defv_gravity, 25
 - defv_rhoair, 25
 - defv_rhowater, 26
 - deg2rad, 26
 - enddate, 26
 - enddatespecified, 26
 - enddatetime, 26
 - endday, 26
 - endhour, 27
 - endmin, 27
 - endmonth, 27
 - endsec, 27
 - endsimspecified, 27
 - endsimtime, 27
 - endtime, 28
 - endyear, 28
 - gravity, 28
 - kt2ms, 28
 - logfilename, 28
 - lun_btrk, 28
 - lun_btrk1, 29
 - lun_ctrl, 29
 - lun_inp, 29
 - lun_inp1, 29
 - lun_log, 29
 - lun_out, 29
 - lun_out1, 30
 - lun_screen, 30
 - m2nm, 30

- mb2kpa, [30](#)
- mb2pa, [30](#)
- mdbegsimtime, [30](#)
- mdendsimtime, [31](#)
- mdoutdt, [31](#)
- meshfileform, [31](#)
- meshfilename, [31](#)
- meshfilenamespecified, [31](#)
- meshfiletype, [31](#)
- modeltype, [32](#)
- ms2kt, [32](#)
- nbtrfiles, [32](#)
- ncdeflate, [32](#)
- ncdlevel, [32](#)
- ncshuffle, [32](#)
- ncvarnam_pres, [33](#)
- ncvarnam_wndx, [33](#)
- ncvarnam_wndy, [33](#)
- nm2m, [33](#)
- noutdt, [33](#)
- omega, [33](#)
- one2ten, [34](#)
- outdt, [34](#)
- outfilename, [34](#)
- outfilenamespecified, [34](#)
- pi, [34](#)
- rad2deg, [34](#)
- rearth, [35](#)
- refdate, [35](#)
- refdatespecified, [35](#)
- refdatetime, [35](#)
- refday, [35](#)
- refhour, [35](#)
- refmin, [36](#)
- refmonth, [36](#)
- refsec, [36](#)
- reftime, [36](#)
- refyear, [36](#)
- rhoair, [36](#)
- rhowater, [37](#)
- ten2one, [37](#)
- times, [37](#)
- title, [37](#)
- unittime, [37](#)
- windreduction, [37](#)
- wpress, [38](#)
- writparams, [38](#)
- wvelx, [38](#)
- wvely, [38](#)
- pahm_init
 - pahm_drivermod, [16](#)
- pahm_mesh, [38](#)
 - agrid, [41](#)
 - allocatenodalandelementalarrays, [39](#)
 - dp, [41](#)
 - ics, [41](#)
 - ismeshok, [41](#)
 - maxfacenodes, [42](#)
 - ne, [42](#)
 - nfn, [42](#)
 - nm, [42](#)
 - np, [42](#)
 - readmesh, [39](#)
 - readmeshasciifort14, [40](#)
 - sfea, [42](#)
 - sfea0, [43](#)
 - slam, [43](#)
 - slam0, [43](#)
 - xcslam, [43](#)
 - ycsfea, [43](#)
- pahm_messages, [44](#)
 - allmessage_1, [45](#)
 - allmessage_2, [45](#)
 - closelogfile, [45](#)
 - debug, [51](#)
 - echo, [51](#)
 - error, [51](#)
 - info, [52](#)
 - initlogging, [45](#)
 - logfileopened, [52](#)
 - loginitcalled, [52](#)
 - loglevelnames, [52](#)
 - logmessage_1, [46](#)
 - logmessage_2, [46](#)
 - messagesources, [52](#)
 - nscreen, [53](#)
 - openlogfile, [46](#)
 - programhelp, [47](#)
 - programversion, [47](#)
 - scratchformat, [53](#)
 - scratchmessage, [53](#)
 - screenmessage_1, [48](#)
 - screenmessage_2, [48](#)
 - setmessagesource, [48](#)
 - sourcenumber, [53](#)
 - terminate, [49](#)
 - unsetmessagesource, [50](#)
 - warning, [53](#)
- pahm_messages::allmessage, [159](#)
 - allmessage_1, [159](#)
 - allmessage_2, [159](#)
- pahm_messages::logmessage, [208](#)
 - logmessage_1, [209](#)
 - logmessage_2, [209](#)
- pahm_messages::screenmessage, [209](#)
 - screenmessage_1, [210](#)
 - screenmessage_2, [210](#)
- pahm_netcdfio, [54](#)

- base_netcdfcheckerr, 54
- initadcircnetcdfoutfile, 55
- mytime, 60
- netcdfterminate, 56
- newadcircnetcdfoutfile, 57
- setrecordcounterandstoretime, 58
- writenetcdfrecord, 59
- pahm_netcdfio::filedata_t, 187
 - filefound, 187
 - filename, 188
 - filereccounter, 188
 - initialized, 188
- pahm_netcdfio::timedata_t, 224
 - initialized, 224
 - time, 224
 - timedimid, 225
 - timedims, 225
 - timeid, 225
 - timelen, 225
- pahm_run
 - pahm_drivermod, 17
- pahm_sizes, 60
 - blank, 64
 - comparedoublereals, 61
 - comparesinglereals, 62
 - fixnearwholedoublereal, 62
 - fixnearwholesinglereal, 63
 - framelen, 64
 - hp, 64
 - imissv, 64
 - int1, 64
 - int16, 64
 - int2, 64
 - int4, 65
 - int8, 65
 - ip, 65
 - llong, 65
 - long, 65
 - nbyte, 65
 - rmissv, 65
 - sp, 66
 - sz, 66
 - wp, 66
- pahm_sizes::comparereals, 175
 - comparedoublereals, 175
 - comparesinglereals, 176
- pahm_sizes::fixnearwholereal, 188
 - fixnearwholedoublereal, 189
 - fixnearwholesinglereal, 189
- pahm_vortex, 66
 - b, 84
 - bs, 84
 - bs4, 85
 - calcintensitychange, 68
 - calcrmaxes, 69
 - calcrmaxesfull, 69
 - clat, 85
 - clon, 85
 - corio, 85
 - fang, 70
 - findroot, 70
 - fitrmaxes, 71
 - fitrmaxes4, 71
 - getlatestangle, 72
 - getlatestrmax, 72
 - getphifactors, 72
 - getrmaxes, 72
 - getshapeparameter, 72
 - getshapeparameters, 72
 - getusequadrantvr, 73
 - getvmaxesbl, 73
 - interpr, 73
 - latestangle, 85
 - latestrmax, 85
 - newvortex, 74
 - newvortexfull, 74
 - npoints, 86
 - nquads, 86
 - pc, 86
 - phi, 86
 - phis, 86
 - phis4, 86
 - pn, 87
 - quad, 87
 - quadflag4, 87
 - quadir4, 87
 - radius, 87
 - rmaxes, 87
 - rmaxes4, 88
 - rmw, 75
 - rotate, 75
 - setisotachradii, 76
 - setisotachwindspeeds, 76
 - setrmaxes, 77
 - setshapeparameter, 77
 - setusequadrantvr, 77
 - setusevmaxesbl, 77
 - setvmaxesbl, 77
 - setvortex, 78
 - spinterp, 78
 - usequadrantvr, 88
 - usevmaxesbl, 88
 - uvp, 78
 - uvpr, 79
 - uvtrans, 80
 - uvtranspoint, 81
 - vhncori, 82
 - vhwithcori, 82

- vhwithcorifull, 83
- vmax, 88
- vmbl, 88
- vmbl4, 88
- vr, 89
- vrquadrant, 89
- parseline
 - utilities, 147
- parwind, 89
 - allocbtrstruct, 90
 - allochollstruct, 90
 - besttrackdata, 99
 - deallocbtrstruct, 91
 - deallochollstruct, 91
 - gethollandfields, 92, 93
 - processhollanddata, 94
 - readbesttrackfile, 95
 - readcsvbesttrackfile, 96
 - windreftime, 99
 - writebesttrackdata, 97
- parwind-orig.F90, 306, 307
- parwind.F90, 326, 327
- parwind::besttrackdata_t, 168
 - basin, 169
 - cyclenum, 169
 - cynum, 169
 - day, 169
 - dir, 170
 - dtg, 170
 - ew, 170
 - eye, 170
 - filename, 170
 - gusts, 170
 - hour, 170
 - initials, 170
 - intlnt, 171
 - intlnt, 171
 - intmslp, 171
 - intpouter, 171
 - intrad1, 171
 - intrad2, 171
 - intrad3, 171
 - intrad4, 171
 - intrmw, 172
 - introuter, 172
 - intspeed, 172
 - intvmax, 172
 - lat, 172
 - loaded, 172
 - lon, 172
 - maxseas, 172
 - month, 173
 - ns, 173
 - numrec, 173
 - rad, 173
 - stormname, 173
 - subregion, 173
 - tau, 173
 - tech, 173
 - technum, 174
 - thisstorm, 174
 - ty, 174
 - windcode, 174
 - year, 174
- parwind::hollanddata_t, 197
 - basin, 198
 - casttime, 198
 - casttype, 199
 - cprdt, 199
 - cpress, 199
 - day, 199
 - dtg, 199
 - fcstinc, 199
 - filename, 199
 - hour, 199
 - icpress, 200
 - ilat, 200
 - ilon, 200
 - irmw, 200
 - irrp, 200
 - ispeed, 200
 - lat, 200
 - loaded, 200
 - lon, 201
 - month, 201
 - numrec, 201
 - rmw, 201
 - rrp, 201
 - speed, 201
 - stormnumber, 201
 - thisstorm, 201
 - trvx, 202
 - trvy, 202
 - year, 202
- pc
 - pahm_vortex, 86
- phi
 - pahm_vortex, 86
- phis
 - pahm_vortex, 86
- phis4
 - pahm_vortex, 86
- pi
 - pahm_global, 34
- pn
 - pahm_vortex, 87
- preprocessdatetimestring
 - timedateutils, 126

printmodelparams
 utilities, 149
processhollanddata
 parwind, 94
programhelp
 pahm_messages, 47
programversion
 pahm_messages, 47

quad
 pahm_vortex, 87
quadflag4
 pahm_vortex, 87
quadir4
 pahm_vortex, 87
quicksort
 sortutils, 107
quotation
 csv_module::csv_file, 185
quote
 csv_module::csv_file, 185

rad
 parwind::besttrackdata_t, 173
rad2deg
 pahm_global, 34
radius
 pahm_vortex, 87
read
 csv_module::csv_file, 183
read_line_from_file
 csv_module::csv_file, 183
readbesttrackfile
 parwind, 95
readcontrolfile
 utilities, 149
readcsvbesttrackfile
 parwind, 96
readmesh
 pahm_mesh, 39
readmeshasciifort14
 pahm_mesh, 40
real
 csv_module, 9
 csv_parameters, 12
realscan
 utilities, 151
rearth
 pahm_global, 35
refdate
 pahm_global, 35
refdatespecified
 pahm_global, 35
refdatetime
 pahm_global, 35

refday
 pahm_global, 35
refhour
 pahm_global, 35
refmin
 pahm_global, 36
refmonth
 pahm_global, 36
refsec
 pahm_global, 36
reftime
 pahm_global, 36
refyear
 pahm_global, 36
rhoair
 pahm_global, 36
rhowater
 pahm_global, 37
rmaxes
 pahm_vortex, 87
rmaxes4
 pahm_vortex, 88
rmissv
 pahm_sizes, 65
rmw
 pahm_vortex, 75
 parwind::hollanddata_t, 201
rotate
 pahm_vortex, 75
rrp
 parwind::hollanddata_t, 201

scratchformat
 pahm_messages, 53
scratchmessage
 pahm_messages, 53
screenmessage_1
 pahm_messages, 48
 pahm_messages::screenmessage, 210
screenmessage_2
 pahm_messages, 48
 pahm_messages::screenmessage, 210
setisotachradii
 pahm_vortex, 76
setisotachwindspeeds
 pahm_vortex, 76
setmessagesource
 pahm_messages, 48
setrecordcounterandstoretime
 pahm_netcdfio, 58
setrmaxes
 pahm_vortex, 77
setshapeparameter
 pahm_vortex, 77

- setusequadrantvr
 - pahm_vortex, [77](#)
- setusevmaxesbl
 - pahm_vortex, [77](#)
- setvmaxesbl
 - pahm_vortex, [77](#)
- setvortex
 - pahm_vortex, [78](#)
- sfea
 - pahm_mesh, [42](#)
- sfea0
 - pahm_mesh, [43](#)
- sizes.F90, [346](#), [348](#)
- slam
 - pahm_mesh, [43](#)
- slam0
 - pahm_mesh, [43](#)
- sort2
 - sortutils, [108](#)
- sort_ascending
 - csv_utilities, [13](#)
- sortutils, [99](#)
 - arraycopydouble, [101](#)
 - arraycopyint, [101](#)
 - arraycopysingle, [102](#)
 - arrayequaldouble, [102](#)
 - arrayequalint, [102](#)
 - arrayequalsingle, [103](#)
 - arthdouble, [103](#)
 - arthint, [104](#)
 - arthsingle, [104](#)
 - indexxdouble, [105](#)
 - indexxint, [105](#)
 - indexxint8, [106](#)
 - indexxsingle, [106](#)
 - indexxstring, [107](#)
 - quicksort, [107](#)
 - sort2, [108](#)
 - stringlexcomp, [109](#)
 - swapdouble, [110](#)
 - swapdoublevec, [111](#)
 - swapint, [111](#)
 - swapintvec, [112](#)
 - swapsingle, [112](#)
 - swapsinglevec, [113](#)
- sortutils.F90, [351](#), [353](#)
 - icompchg, [352](#)
- sortutils::arraycopy, [160](#)
 - arraycopydouble, [160](#)
 - arraycopyint, [162](#)
 - arraycopysingle, [162](#)
- sortutils::arrayequal, [163](#)
 - arrayequaldouble, [164](#)
 - arrayequalint, [164](#)
 - arrayequalsingle, [165](#)
- sortutils::arth, [165](#)
 - arthdouble, [166](#)
 - arthint, [166](#)
 - arthsingle, [167](#)
- sortutils::indexx, [202](#)
 - indexxdouble, [203](#)
 - indexxint, [204](#)
 - indexxint8, [205](#)
 - indexxsingle, [206](#)
 - indexxstring, [207](#)
- sortutils::swap, [216](#)
 - swapdouble, [217](#)
 - swapdoublevec, [217](#)
 - swapint, [218](#)
 - swapintvec, [219](#)
 - swapsingle, [219](#)
 - swapsinglevec, [220](#)
- sourcenumbr
 - pahm_messages, [53](#)
- sp
 - pahm_sizes, [66](#)
- speed
 - parwind::hollanddata_t, [201](#)
- sphericaldistance_1d
 - utilities, [152](#)
 - utilities::sphericaldistance, [211](#)
- sphericaldistance_2d
 - utilities, [153](#)
 - utilities::sphericaldistance, [212](#)
- sphericaldistance_scalar
 - utilities, [154](#)
 - utilities::sphericaldistance, [213](#)
- sphericaldistanceharv
 - utilities, [154](#)
- sphericalfracpoint
 - utilities, [155](#)
- spinterp
 - pahm_vortex, [78](#)
- splitdate
 - timedateutils, [127](#)
- splitdatetimestring
 - timedateutils, [127](#)
 - timedateutils::splitdatetimestring, [214](#)
- splitdatetimestring2
 - timedateutils, [128](#)
 - timedateutils::splitdatetimestring, [215](#)
- stormname
 - parwind::besttrackdata_t, [173](#)
- stormnumber
 - parwind::hollanddata_t, [201](#)
- str
 - csv_module::csv_string, [186](#)
- string

- csv_module, 10
 - csv_parameters, 12
- stringlexcomp
 - sortutils, 109
- subregion
 - parwind::besttrackdata_t, 173
- swapdouble
 - sortutils, 110
 - sortutils::swap, 217
- swapdoublevec
 - sortutils, 111
 - sortutils::swap, 217
- swapint
 - sortutils, 111
 - sortutils::swap, 218
- swapintvec
 - sortutils, 112
 - sortutils::swap, 219
- swapsingle
 - sortutils, 112
 - sortutils::swap, 219
- swapsinglevec
 - sortutils, 113
 - sortutils::swap, 220
- sz
 - pahm_sizes, 66
- tau
 - parwind::besttrackdata_t, 173
- tech
 - parwind::besttrackdata_t, 173
- technum
 - parwind::besttrackdata_t, 174
- ten2one
 - pahm_global, 37
- terminate
 - pahm_messages, 49
- thisstorm
 - parwind::besttrackdata_t, 174
 - parwind::hollanddata_t, 201
- time
 - pahm_netcdfio::timedata_t, 224
- timeconvisec
 - timedateutils, 128
 - timedateutils::timeconv, 221
- timeconvrsec
 - timedateutils, 129
 - timedateutils::timeconv, 222
- timedateutils, 114
 - datetime2string, 115
 - dayofyear, 116
 - dayofyeartogreg, 116
 - elapsedsecs, 117
 - firstgregdate, 130
 - firstgregtime, 130
 - gettimeconvsec, 118
 - gregtojulday2, 119
 - gregtojuldayisec, 120
 - gregtojuldayrsec, 121
 - joindate, 122
 - juldaytogreg, 123
 - leapyear, 124
 - mdjdate, 131
 - mdjoffset, 131
 - mdjtime, 131
 - modeldate, 131
 - modeltime, 131
 - modjuldate, 131
 - modjultime, 131
 - monthdays, 125
 - offfirstgregday, 132
 - offmodeljulday, 132
 - offmodjulday, 132
 - offunixjulday, 132
 - preprocessdatetimestring, 126
 - splitdate, 127
 - splitdatetimestring, 127
 - splitdatetimestring2, 128
 - timeconvisec, 128
 - timeconvrsec, 129
 - unixdate, 132
 - unixtime, 132
 - usemodjulday, 132
 - yeardays, 130
- timedateutils.F90, 372, 374
- timedateutils::gregtojulday, 192
 - gregtojulday2, 193
 - gregtojuldayisec, 194
 - gregtojuldayrsec, 195
- timedateutils::splitdatetimestring, 214
 - splitdatetimestring, 214
 - splitdatetimestring2, 215
- timedateutils::timeconv, 221
 - timeconvisec, 221
 - timeconvrsec, 222
- timedimid
 - pahm_netcdfio::timedata_t, 225
- timedims
 - pahm_netcdfio::timedata_t, 225
- timeid
 - pahm_netcdfio::timedata_t, 225
- timelen
 - pahm_netcdfio::timedata_t, 225
- times
 - pahm_global, 37
- title
 - pahm_global, 37
- tokenize

- csv_module::csv_file, 183
- tolowercase
 - utilities, 156
- touppercase
 - utilities, 157
- trvx
 - parwind::hollanddata_t, 202
- trvy
 - parwind::hollanddata_t, 202
- ty
 - parwind::besttrackdata_t, 174
- unique
 - csv_utilities, 13
- unittime
 - pahm_global, 37
- unixdate
 - timedateutils, 132
- unixtime
 - timedateutils, 132
- unsetmessagesource
 - pahm_messages, 50
- usemodjulday
 - timedateutils, 132
- usequadrantvr
 - pahm_vortex, 88
- usevmaxesbl
 - pahm_vortex, 88
- utilities, 133
 - charunique, 134
 - checkcontrolfileinputs, 135
 - closetol, 158
 - convlon, 136
 - cpptogeo_1d, 136
 - cpptogeo_scalar, 137
 - drealscan, 138
 - dvalstr, 139
 - geotocpp_1d, 140
 - geotocpp_scalar, 140
 - getlinerecord, 141
 - getlocandratio, 142
 - intscan, 142
 - intvalstr, 143
 - loadintvar, 144
 - loadlogvar, 145
 - loadrealvar, 146
 - openfileforread, 146
 - parseline, 147
 - printmodelparams, 149
 - readcontrolfile, 149
 - realscan, 151
 - sphericaldistance_1d, 152
 - sphericaldistance_2d, 153
 - sphericaldistance_scalar, 154
 - sphericaldistanceharv, 154
 - sphericalfracpoint, 155
 - tolowercase, 156
 - touppercase, 157
 - valstr, 157
- utilities.F90, 388, 390
- utilities::cpptogeo, 177
 - cpptogeo_1d, 177
 - cpptogeo_scalar, 178
- utilities::geotocpp, 190
 - geotocpp_1d, 191
 - geotocpp_scalar, 191
- utilities::sphericaldistance, 211
 - sphericaldistance_1d, 211
 - sphericaldistance_2d, 212
 - sphericaldistance_scalar, 213
- uvp
 - pahm_vortex, 78
- uvpr
 - pahm_vortex, 79
- uvtrans
 - pahm_vortex, 80
- uvtranspoint
 - pahm_vortex, 81
- valstr
 - utilities, 157
- variable_types
 - csv_module::csv_file, 183
- vhncori
 - pahm_vortex, 82
- vhwithcori
 - pahm_vortex, 82
- vhwithcorifull
 - pahm_vortex, 83
- vmax
 - pahm_vortex, 88
- vmbl
 - pahm_vortex, 88
- vmbl4
 - pahm_vortex, 88
- vortex.F90, 421, 423
- vr
 - pahm_vortex, 89
- vrquadrant
 - pahm_vortex, 89
- warning
 - pahm_messages, 53
- windcode
 - parwind::besttrackdata_t, 174
- windreduction
 - pahm_global, 37
- windreftime
 - parwind, 99

- wp
 - pahm_sizes, [66](#)
- wpress
 - pahm_global, [38](#)
- writebesttrackdata
 - parwind, [97](#)
- writenetcdfrecord
 - pahm_netcdfio, [59](#)
- writeparams
 - pahm_global, [38](#)
- wvelx
 - pahm_global, [38](#)
- wvely
 - pahm_global, [38](#)
- xcslam
 - pahm_mesh, [43](#)
- ycsfea
 - pahm_mesh, [43](#)
- year
 - parwind::besttrackdata_t, [174](#)
 - parwind::hollanddata_t, [202](#)
- yeardays
 - timedateutils, [130](#)