

PaHM

Generated by Doxygen 1.9.3

1 PaHM Manual	1
2 Abstract	2
3 List of Tables	2
4 List of Figures	2
5 Introduction	2
5.1 The Parametric Hurricane Modeling System (<i>PaHM</i>)	3
5.2 Downloading <i>PaHM</i>	3
5.2.1 Directory Structure	4
5.3 Building <i>PaHM</i>	5
5.3.1 System Requirements	5
5.3.2 The Build System	5
5.3.3 CMake Configuration Files and Modules	5
5.3.4 Installation	5
5.4 Using <i>PaHM</i>	6
5.4.1 Standalone Configuration	9
5.4.2 Coupling Configuration	9
6 Parametric Models in <i>PaHM</i>	9
6.1 Rankine Vortex Model	10
6.2 Holland Symmetric Vortex Models	10
6.3 Willoughby Symmetric Vortex Model	10
6.4 Generalized Asymmetric Vortex Holland model (<i>GAHM</i>)	10
7 <i>PaHM</i> Features and Capabilities	10
7.1 Data Input Interfaces	10
7.2 Model Grids	10
7.3 Modeling Multiple Interacting Storms	10
7.4 Coupling Environment	11
8 Model Application and Implementation Technology	11
8.1 Standalone Model Application	11
8.2 Coupled Model Application	11
9 Model Evaluation - Hurricane Florence (2018) Study	11
9.1 Statistical Performance Measures	11
9.2 Standalone Model Evaluation	13
9.2.1 Model Results and Discussion	13
9.3 Coupled Model Evaluation	15

9.3.1 Coupled Model Results and Discussion	16
9.4 Conclusions	18
10 List of Deliverables	18
11 Glossary	18
12 Credits	18
13 References	18
14 PaHM Code	19
14.1 Third-Party Libraries	19
14.2 Functional Parts	19
14.3 Modules	19
14.4 Data Types	19
14.4.1 Data Types List	19
14.4.2 Data Fields	19
14.5 Files	19
14.5.1 File List	19
14.5.2 File Members	19
15 Modules Index	19
15.1 Modules List	19
16 Data Type Index	20
16.1 Data Types List	20
17 File Index	21
17.1 Files	21
18 Module Documentation	22
18.1 csv_module Module Reference	22
18.1.1 Function/Subroutine Documentation	23
18.1.2 Variable Documentation	46
18.2 csv_parameters Module Reference	47
18.2.1 Variable Documentation	47
18.3 csv_utilities Module Reference	48
18.3.1 Function/Subroutine Documentation	49
18.3.2 Variable Documentation	52
18.4 pahm_drivermod Module Reference	52
18.4.1 Function/Subroutine Documentation	52
18.4.2 Variable Documentation	57

18.5 pahm_global Module Reference	57
18.5.1 Function/Subroutine Documentation	60
18.5.2 Variable Documentation	60
18.6 pahm_mesh Module Reference	77
18.6.1 Function/Subroutine Documentation	78
18.6.2 Variable Documentation	79
18.7 pahm_messages Module Reference	82
18.7.1 Function/Subroutine Documentation	83
18.7.2 Variable Documentation	90
18.8 pahm_ncdfio Module Reference	93
18.8.1 Function/Subroutine Documentation	94
18.8.2 Variable Documentation	99
18.9 pahm_sizes Module Reference	102
18.9.1 Function/Subroutine Documentation	103
18.9.2 Variable Documentation	105
18.10 pahm_vortex Module Reference	108
18.10.1 Function/Subroutine Documentation	110
18.10.2 Variable Documentation	130
18.11 parwind Module Reference	135
18.11.1 Function/Subroutine Documentation	136
18.11.2 Variable Documentation	148
18.12 sortutils Module Reference	149
18.12.1 Function/Subroutine Documentation	150
18.13 timedateutils Module Reference	163
18.13.1 Function/Subroutine Documentation	165
18.13.2 Variable Documentation	183
18.14 utilities Module Reference	186
18.14.1 Function/Subroutine Documentation	187
18.14.2 Variable Documentation	212
19 Data Type Documentation	212
19.1 pahm_ncdfio::adcirccoorddata_t Type Reference	212
19.1.1 Detailed Description	213
19.1.2 Member Data Documentation	213
19.2 pahm_ncdfio::adcircvardata3d_t Type Reference	215
19.2.1 Detailed Description	215
19.2.2 Member Data Documentation	215
19.3 pahm_ncdfio::adcircvardata_t Type Reference	217
19.3.1 Detailed Description	217

19.3.2 Member Data Documentation	218
19.4 pahm_messages::allmessage Interface Reference	219
19.4.1 Detailed Description	219
19.4.2 Member Function/Subroutine Documentation	219
19.5 sortutils::arraycopy Interface Reference	220
19.5.1 Detailed Description	221
19.5.2 Member Function/Subroutine Documentation	221
19.6 sortutils::arrayequal Interface Reference	222
19.6.1 Detailed Description	223
19.6.2 Member Function/Subroutine Documentation	223
19.7 sortutils::arth Interface Reference	225
19.7.1 Detailed Description	225
19.7.2 Member Function/Subroutine Documentation	225
19.8 parwind::asymmetricvortexdata_t Type Reference	227
19.8.1 Detailed Description	229
19.8.2 Member Data Documentation	229
19.9 parwind::besttrackdata_t Type Reference	236
19.9.1 Detailed Description	237
19.9.2 Member Data Documentation	238
19.10 pahm_sizes::comparereals Interface Reference	243
19.10.1 Detailed Description	243
19.10.2 Member Function/Subroutine Documentation	243
19.11 utilities::cpptogeо Interface Reference	245
19.11.1 Detailed Description	245
19.11.2 Member Function/Subroutine Documentation	246
19.12 csv_module::csv_file Type Reference	248
19.12.1 Detailed Description	249
19.12.2 Member Function/Subroutine Documentation	249
19.12.3 Member Data Documentation	254
19.13 csv_module::csv_string Type Reference	256
19.13.1 Detailed Description	256
19.13.2 Member Data Documentation	256
19.14 pahm_netcdfio::filedata_t Type Reference	257
19.14.1 Detailed Description	257
19.14.2 Member Data Documentation	257
19.15 pahm_sizes::fixnearwholereal Interface Reference	258
19.15.1 Detailed Description	258
19.15.2 Member Function/Subroutine Documentation	258
19.16 utilities::geotocpp Interface Reference	260

19.16.1 Detailed Description	260
19.16.2 Member Function/Subroutine Documentation	260
19.17 <code>timedateutils::gregtojulday</code> Interface Reference	262
19.17.1 Detailed Description	262
19.17.2 Member Function/Subroutine Documentation	262
19.18 <code>parwind::hollanddata_t</code> Type Reference	267
19.18.1 Detailed Description	268
19.18.2 Member Data Documentation	268
19.19 <code>sortutils::indexx</code> Interface Reference	272
19.19.1 Detailed Description	272
19.19.2 Member Function/Subroutine Documentation	273
19.20 <code>pahm_messages::logmessage</code> Interface Reference	277
19.20.1 Detailed Description	277
19.20.2 Member Function/Subroutine Documentation	278
19.21 <code>pahm_messages::screenmessage</code> Interface Reference	279
19.21.1 Detailed Description	279
19.21.2 Member Function/Subroutine Documentation	279
19.22 <code>utilities::sphericaldistance</code> Interface Reference	280
19.22.1 Detailed Description	281
19.22.2 Member Function/Subroutine Documentation	281
19.23 <code>timedateutils::splittatetimestring</code> Interface Reference	284
19.23.1 Detailed Description	284
19.23.2 Constructor & Destructor Documentation	284
19.23.3 Member Function/Subroutine Documentation	285
19.24 <code>sortutils::swap</code> Interface Reference	286
19.24.1 Detailed Description	287
19.24.2 Member Function/Subroutine Documentation	287
19.25 <code>timedateutils::timeconv</code> Interface Reference	291
19.25.1 Detailed Description	291
19.25.2 Member Function/Subroutine Documentation	291
19.26 <code>pahm_netcdffio::timedata_t</code> Type Reference	293
19.26.1 Detailed Description	294
19.26.2 Member Data Documentation	294
20 File Documentation	295
20.1 <code>user-guide/abstract.md</code> File Reference	295
20.2 <code>user-guide/application.md</code> File Reference	295
20.3 <code>user-guide/code.md</code> File Reference	295
20.4 <code>user-guide/credits.md</code> File Reference	295

20.5 user-guide/deliverables.md File Reference	295
20.6 user-guide/evaluation.md File Reference	295
20.7 user-guide/features.md File Reference	295
20.8 user-guide/figures.md File Reference	295
20.9 user-guide/glossary.md File Reference	295
20.10 user-guide/intro.md File Reference	295
20.11 user-guide/models.md File Reference	295
20.12 user-guide/pahm_manual.md File Reference	295
20.13 user-guide/references-dox.md File Reference	295
20.14 user-guide/tables.md File Reference	295
20.15 /home/takis/CSDL/parwinds-doc/src/csv_module.F90 File Reference	295
20.15.1 Detailed Description	297
20.16 csv_module.F90	298
20.17 /home/takis/CSDL/parwinds-doc/src/csv_parameters.F90 File Reference	314
20.17.1 Detailed Description	315
20.18 csv_parameters.F90	315
20.19 /home/takis/CSDL/parwinds-doc/src/csv_utilities.F90 File Reference	316
20.19.1 Detailed Description	316
20.19.2 Function/Subroutine Documentation	316
20.20 csv_utilities.F90	318
20.21 /home/takis/CSDL/parwinds-doc/src/driver_mod.F90 File Reference	320
20.21.1 Detailed Description	321
20.22 driver_mod.F90	321
20.23 /home/takis/CSDL/parwinds-doc/src/global.F90 File Reference	323
20.23.1 Detailed Description	326
20.24 global.F90	326
20.25 /home/takis/CSDL/parwinds-doc/src/mesh.F90 File Reference	330
20.25.1 Detailed Description	331
20.26 mesh.F90	331
20.27 /home/takis/CSDL/parwinds-doc/src/messages.F90 File Reference	335
20.27.1 Detailed Description	336
20.28 messages.F90	336
20.29 /home/takis/CSDL/parwinds-doc/src/netcdfio.F90 File Reference	341
20.29.1 Detailed Description	342
20.29.2 Macro Definition Documentation	342
20.30 netcdfio.F90	343
20.31 /home/takis/CSDL/parwinds-doc/src/pahm.F90 File Reference	353
20.31.1 Detailed Description	353
20.31.2 Function/Subroutine Documentation	354

20.32 pahm.F90	355
20.33 /home/takis/CSDL/parwinds-doc/src/parwind.F90 File Reference	356
20.33.1 Detailed Description	357
20.34 parwind.F90	357
20.35 /home/takis/CSDL/parwinds-doc/src/sizes.F90 File Reference	400
20.35.1 Detailed Description	401
20.36 sizes.F90	401
20.37 /home/takis/CSDL/parwinds-doc/src/sortutils.F90 File Reference	404
20.37.1 Detailed Description	406
20.37.2 Function/Subroutine Documentation	406
20.38 sortutils.F90	407
20.39 /home/takis/CSDL/parwinds-doc/src/timedateutils.F90 File Reference	426
20.39.1 Detailed Description	427
20.40 timedateutils.F90	428
20.41 /home/takis/CSDL/parwinds-doc/src/utilities.F90 File Reference	441
20.41.1 Detailed Description	443
20.42 utilities.F90	443
20.43 /home/takis/CSDL/parwinds-doc/src/vortex.F90 File Reference	474
20.43.1 Detailed Description	476
20.44 vortex.F90	476
Index	499

1 *PaHM* Manual

Parametric Hurricane Modeling System 

User's Guide

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

February 2022

2 Abstract

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 to support coastal applications.

PaHM, an [ESMF/NUOPC](#) compatible modeling system, 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](#)). The core modeling components of the system are the Holland-B Models ([[1980](#)], [[2010](#)]) and the Generalized Holland Parametric Tropical Cyclone Model (Gao et al. [[2015](#)]).

PaHM is developed at the Coastal Marine Modeling Branch under the office of Coastal of Coast Survey at NOAA's National Ocean Service. In a "standalone" configuration it outputs gridded atmospheric fields to force any ocean/wave model while, in its "coupling" configuration, it feeds the atmospheric fields to the couple ocean/wave model via its own NUOPC Cap. The source code of *PaHM* can be accessed at: <https://github.com/noaa-ocs-modeling/PaHM>.

3 List of Tables

4 List of Figures

5 Introduction

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 to support coastal applications.

PaHM is not an atmospheric model but rather a modeling system that contains multiple parametric tropical cyclone (TC) models that can be activated during run time to generate the required atmospheric wind fields. The functional parametric models are the [Holland models \(1980, 2010\)](#) and the [Generalized Asymmetric Vortex Model \(GAHM\)](#). Two additional parametric modeling components are in development that is, the [Rankine Vortex Model](#) and the [Willoughby Model](#).

5.1 The Parametric Hurricane Modeling System (*PaHM*)

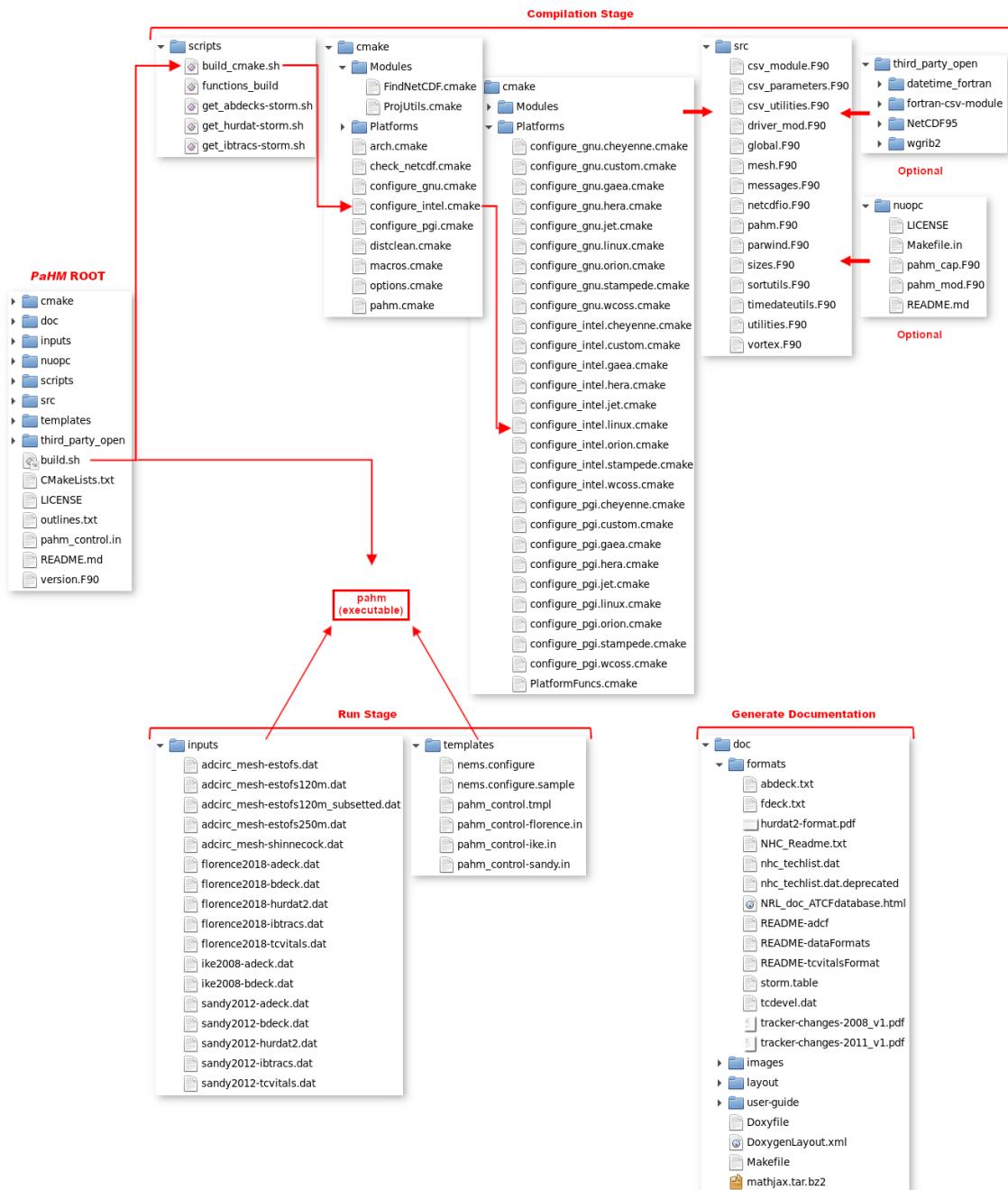
PaHM is a modeling system that reads "best track" type of files (e.g., those produced by the [National Hurricane Center](#)) to generate gridded atmospheric fields (usually, 10-m wind velocities and atmospheric pressures converted to mean sea level). The file formats currently recognized by *PaHM* are: (a) [a/b-deck](#), (b) [HurDat2](#), (c) [IBTrACS](#) and (d) [TCVitals](#). *PaHM* has a built-in CSV I/O interface therefore, it can read and write any of these files in ASCII format.

5.2 Downloading *PaHM*

The latest source code of *PaHM* can be downloaded from its GitHub repository at: <https://github.com/noaa-ocs-modeling/PaHM>. Binary distributions of *PaHM* are not available currently. The full online documentation of the modeling system can be accessed at: <https://noaa-ocs-modeling.github.io/PaHM/html/index.html>.

```
git clone https://github.com/noaa-ocs-modeling/PaHM PaHM
```

5.2.1 Directory Structure



5.3 Building *PaHM*

5.3.1 System Requirements

5.3.2 The Build System

```

Usage: "build.sh" [{-|--}option1[=|space]option_value1] [{-|--}option2[=|space]option_value2]] ...

-h|--help|--h|--help
Show this help screen.

-c|--c|--clean|--clean [=|space] "0|1|2|-3|-2|-1|yes|no" (OPTIONAL).
Only clean the already compiled CMake build system.
Default: 0|no.
Example: --clean=1 Clean the system (make clean) and exit.
         =2 Completely clean the system (make distclean) and exit.
         =-1 During the compilation stage, clean the system (make clean)
              and continue with the compilation.
         =-2 During the compilation stage, completely clean the system (make distclean)
              and continue with the compilation.
         =-3 Do not clean anything but continue with the compilation.
         =0 Do not clean anything (default).

-cmake_flags|--cmake_flags [=|space] "cmake_flags" (OPTIONAL).
Additional flags to pass to the cmake program.
Example: --cmake_flags "-DFLAG1=VAL1 -DFLAG2=VAL2 ...".
Default: none.

-compiler|--compiler [=|space] "compiling_system" (OPTIONAL).
The compiling system to use (gnu, intel, pgi).
Default: none.

-j|--j [=|space] "N" (OPTIONAL).
Define the number of make jobs to run simultaneously.
Default: 1.

-par|--par|--parallel|--parallel [=|space] "0|1|yes|no" (OPTIONAL).
Activate the use of parallel compilers.
Default: 0|no.

-plat|--plat|--platform|--platform [=|space] "platform" (OPTIONAL).
The name of the compute HPC platform to consider.
Selecting a platform additional macros are defined for the
compilation stage that are specific to that platform.
Supported platforms: custom, linux, cheyenne, gaea, hera, jet, orion, stampede, wcoress.
Default: OS.

-prefix|--prefix [=|space] "install_dir" (OPTIONAL).
The path to the installation directory.
Default: The location of this script.

-proj|--proj [=|space] "project_dir" (OPTIONAL).
The path to the user's project directory.
Default: The location of this script.

-t|--t|-type|--type [=|space] "cmake_build_type" (OPTIONAL).
To set the CMAKE_BUILD_TYPE option (Debug Release RelWithDebInfo MinSizeRel).
D = Debug.
R = Release.
RD = RelWithDebInfo.
MR = MinSizeRel.
Default: R.

-v|--v|--verbose|--verbose [=|space] "0|1|yes|no" (OPTIONAL).
Enable verbosity in the make files during compilation.
Default: 0|no.

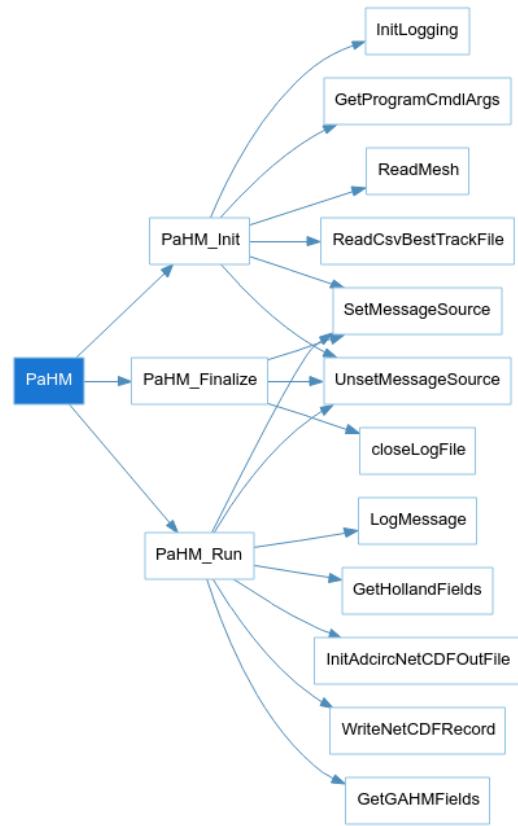
```

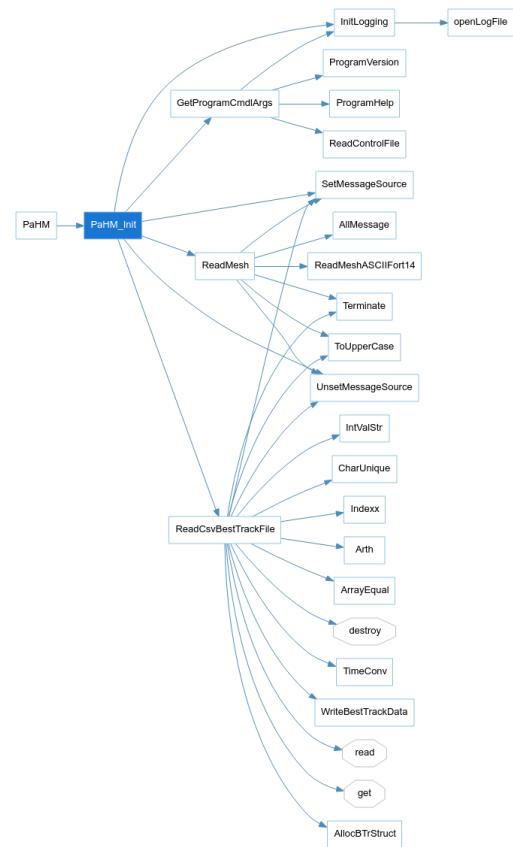
5.3.3 CMake Configuration Files and Modules

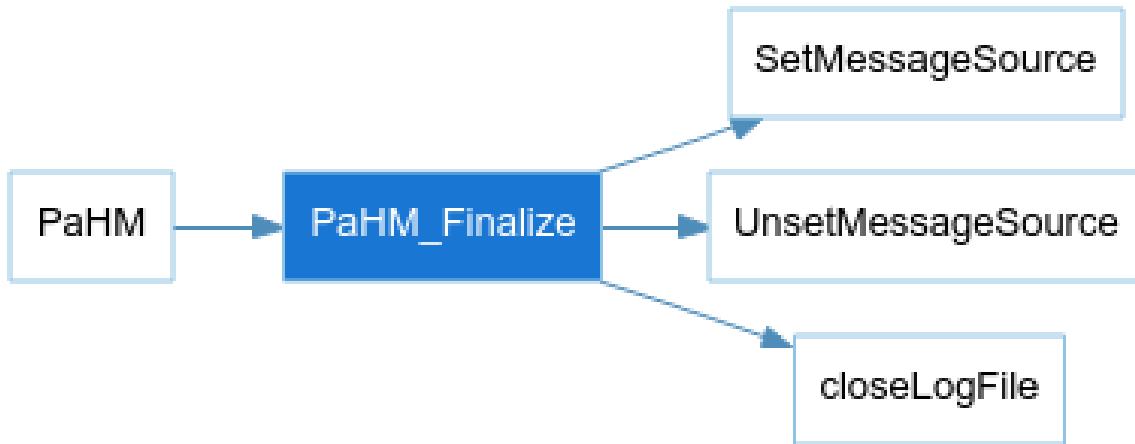
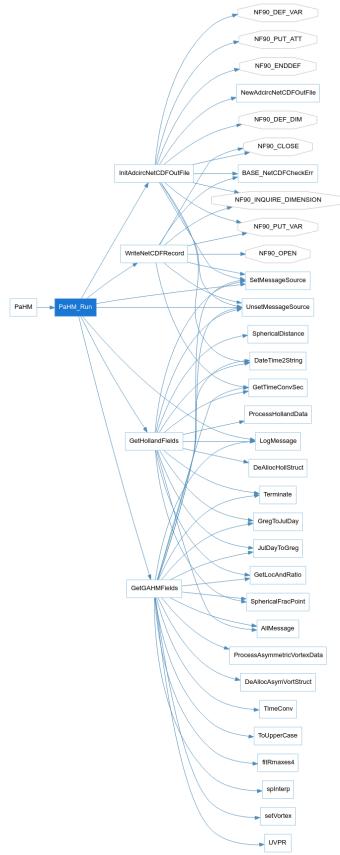
5.3.4 Installation

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

5.4 Using *PaHM*







5.4.1 Standalone Configuration

5.4.2 Coupling Configuration

6 Parametric Models in *PaHM*

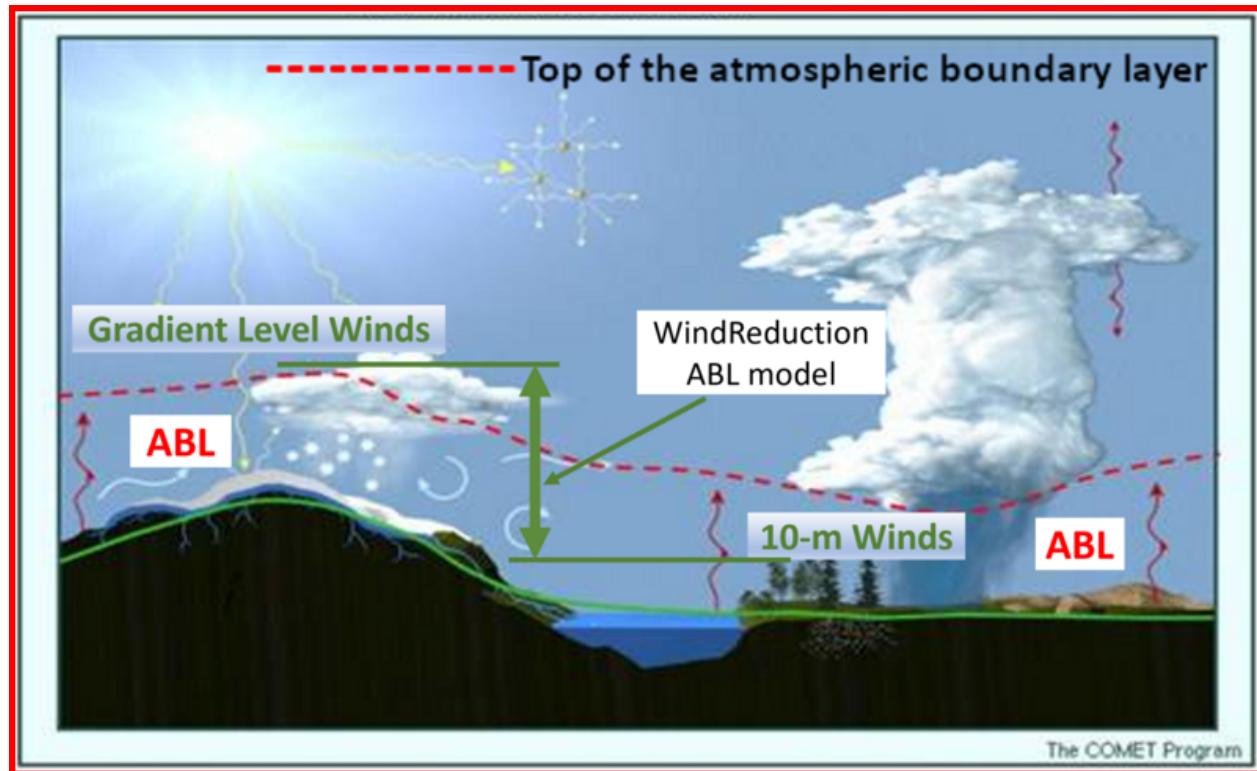


Figure 1 Figure1:

6.1 Rankine Vortex Model

6.2 Holland Symmetric Vortex Models

6.3 Willoughby Symmetric Vortex Model

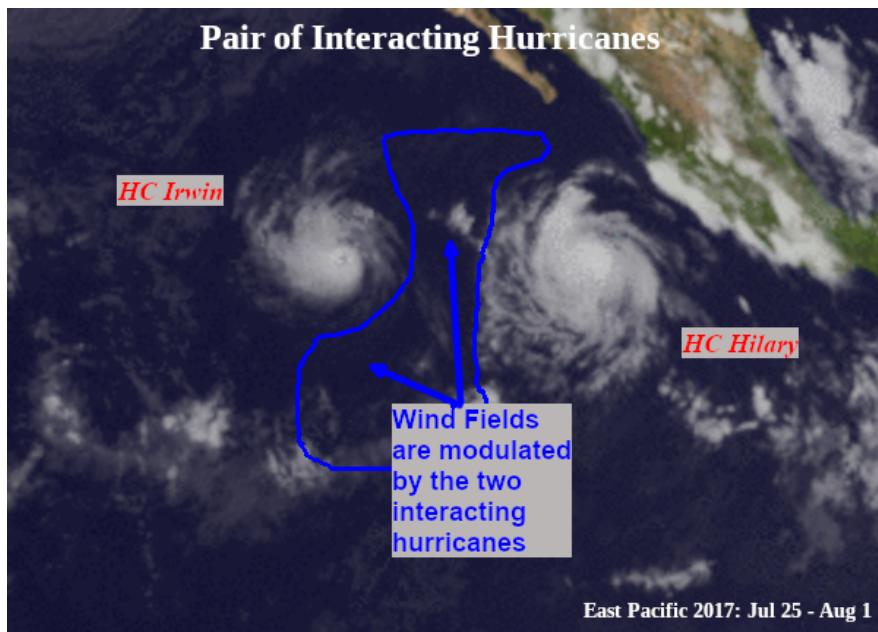
6.4 Generalized Asymmetric Vortex Holland model (*GAHM*)

7 PaHM Features and Capabilities

7.1 Data Input Interfaces

7.2 Model Grids

7.3 Modeling Multiple Interacting Storms



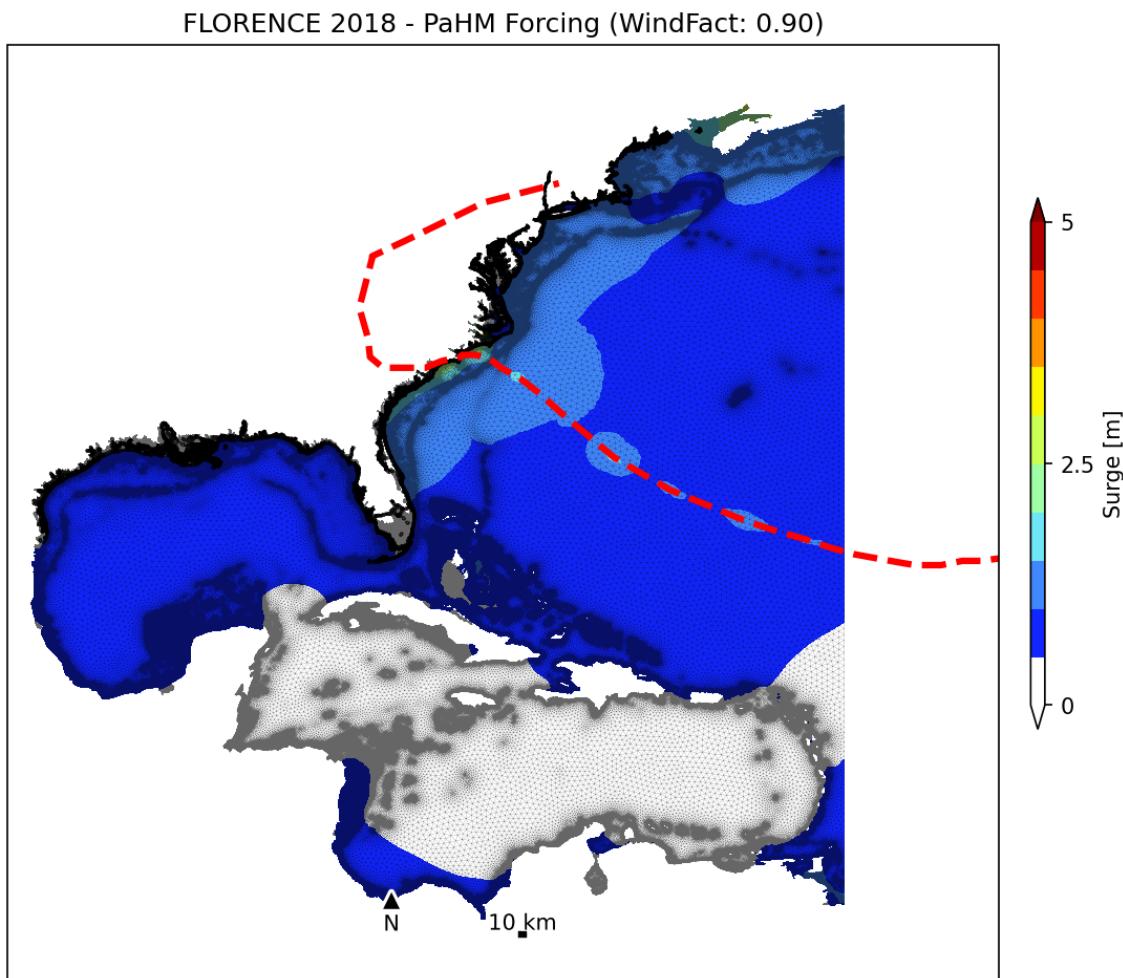
7.4 Coupling Environment

8 Model Application and Implementation Technology

8.1 Standalone Model Application

8.2 Coupled Model Application

9 Model Evaluation - Hurricane Florence (2018) Study



9.1 Statistical Performance Measures

Only parametric statistical tests are used in the performance evaluation of the developed model that include (a) the mean (m) of the differences between the calculated and the measured or observed data sets, (b) the standard deviation (SD

), (c) the root mean square difference (*RMSE*), (d) the coefficient of determination (*R²*), (e) the bias (*bias*), (f) the Willmott (2012) skill (*WS12 skill*) and (g) the Nash and Sutcliffe (1970) skill (*NS skill*).

a) Mean: The mean of the differences between the modeled and measured data provides a gross overall measure of the model performance and is calculated as:

$$m = \frac{\sum_{i=1}^n (M_i - O_i)}{n}$$

where n is the total number of observation or modeled points, M_i are the modeled and O_i are the observed values of each evaluated variable. The smaller the mean difference the better the agreement between the model and the observed values, with a value of zero denoting absolute agreement.

b) Standard Deviation: The standard deviation (*SD*) is a measure of the distance of the difference between the calculated and observed data from the mean difference. Small standard deviations indicate that the differences are closer to the mean. The standard deviation is calculated as:

$$SD = \sqrt{\frac{\sum_{i=1}^n [(M_i - O_i) - m]^2}{n}}$$

c) Root Mean Square Difference: The root mean square difference (*RMSE*) is another test of the overall model performance that measures how close the modeled value of a variable is to the observed value. Mathematically, the test is defined as:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (M_i - O_i)^2}{n}}$$

The differences between the modeled and observed data are squared so that more weight is given to larger errors.

d) Coefficient of Determination: The coefficient of determination *R²*, where R is the correlation coefficient, indicates the proportion of the variance in the dependent variable that is predicted by linear regression and the independent variable. In general, a high *R²* value indicates that the model is a good fit for the data. An *R² = 0.62*, indicates that 62% of the variation in the outcome has been explained. A value of 1 would indicate that the regression line represents all of the data (the best fit) while, a value of 0 shows no association at all. Note that the coefficient of determination shows only the magnitude of the association, not whether that association is statistically significant.

$$R = \frac{\sum_{i=1}^n (O_i - \bar{O})(M_i - \bar{M})}{\sqrt{\sum_{i=1}^n (O_i - \bar{O})^2(M_i - \bar{M})^2}}$$

e) Model Bias: Bias is the tendency of a statistical estimator to overestimate or underestimate a parameter. The bias of a statistical estimator is the difference between the expected value of the statistic and the true value of the sample (population) parameter. If the bias is close to zero then the statistical estimator is an unbiased estimator, otherwise it is considered a biased estimator. The statistical estimator used here is the sample or population mean.

$$bias = \bar{O} - \bar{M}$$

f) Willmott Skill Index: The evaluation of model performance, that is the comparison model estimates with observed values, is a fundamental step for model development and use. This validation process includes criteria that rely on

mathematical measurements of how well model results simulate the observed values. The parameter (WS12 skill), called index of agreement, is a relative average error and bounded measure. The best agreement between model results and observations will yield a skill of one while, a value of ≤ 0 denotes a complete disagreement. This statistic is calculated using the following equations:

$$\begin{aligned} SM1 &= \sum_{i=1}^n O_i - M_i ; & SM2 &= \sum_{i=1}^n M_i - \bar{O} \quad O_i - \bar{O} \\ WS12skill &= 1.0 - SM1 / (2.0 \cdot SM2) & \text{for } SM1 \leq 2.0 \cdot SM2 \\ WS12skill &= 2.0 \cdot SM2 / SM1 - 1.0 & \text{for } SM1 > 2.0 \cdot SM2 \end{aligned}$$

The range of qualification for the Willmott's skill index is given in the following table:

0.8	$WS12 \leq 1.0$	<i>Excellent</i>
0.6	$WS12 \leq 0.8$	<i>Good</i>
0.3	$WS12 \leq 0.6$	<i>Reasonable</i>
0.0	$WS12 \leq 0.3$	<i>Poor</i>
	$WS12 \leq 0.0$	<i>Bad</i>

g) Nash and Sutcliff Skill Index: The Nash and Sutcliffe skill index is similar to the Willmott's skill index and it is calculated as:

$$NSskill = 1.0 - \frac{\sum_{i=1}^n (M_i - O_i)^2}{\sum_{i=1}^n (O_i - \bar{M})^2}$$

For a perfect model with an estimation error variance equal to zero, the Nash and Sutcliffe index equals 1. Values of the Nash and Sutcliffe index close to 1, suggest a model with more predictive skill. The range of qualification presented in the case of the Willmott's skill index can be used for the Nash and Sutcliffe skill index case as well.

All the above tests give information on the size, but not of the nature of the error, which make them adequate measures for a preliminary model evaluation. However, a deeper analysis might require specific tests that can reveal the nature of the errors and help with future model improvements.

9.2 Standalone Model Evaluation

9.2.1 Model Results and Discussion

Table 1 Caption Text

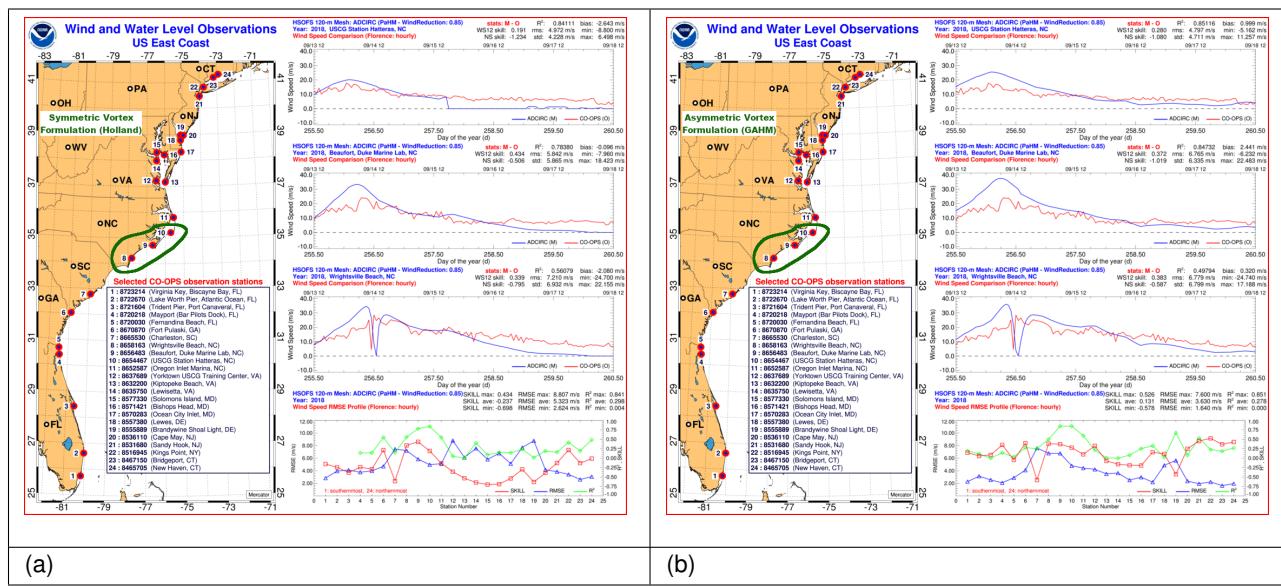


Table 2 Caption Text

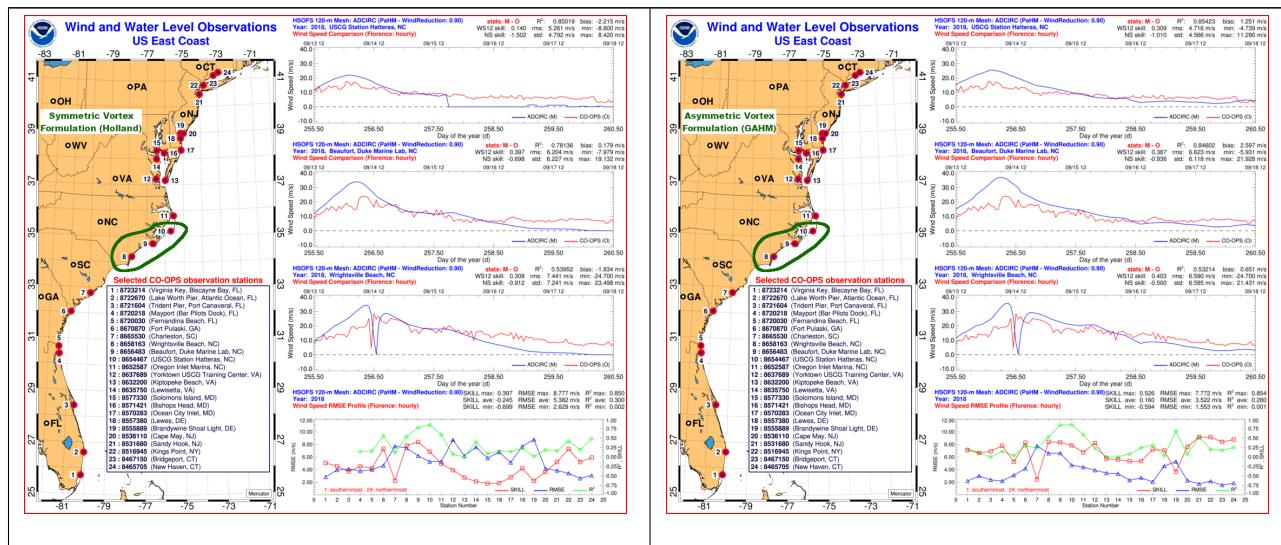
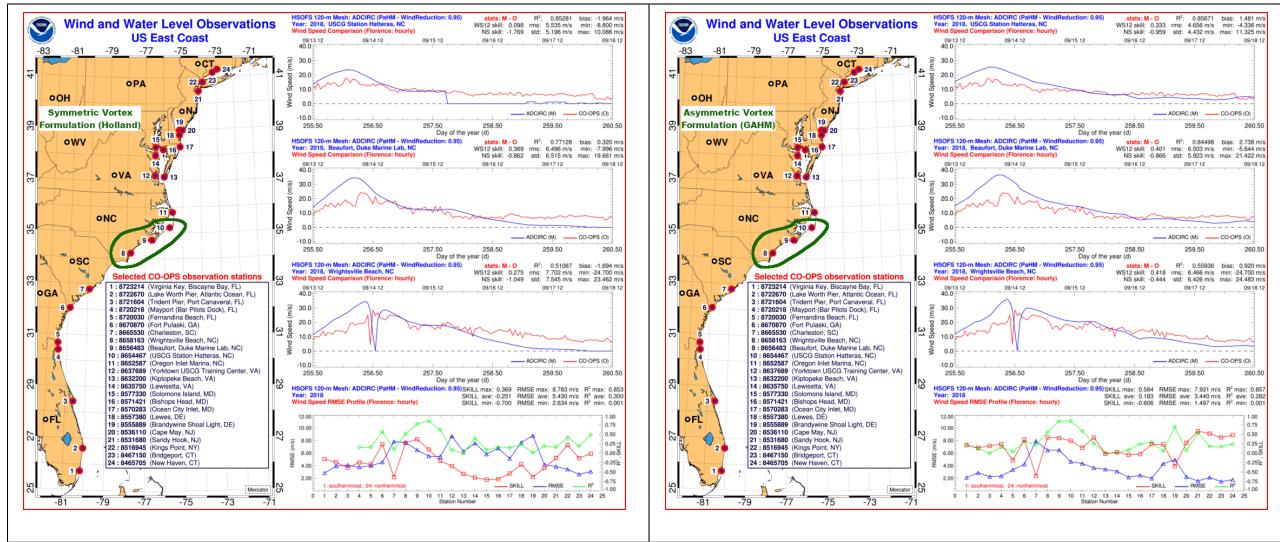


Table 3 Caption Text



9.3 Coupled Model Evaluation

Table 4 Caption Text

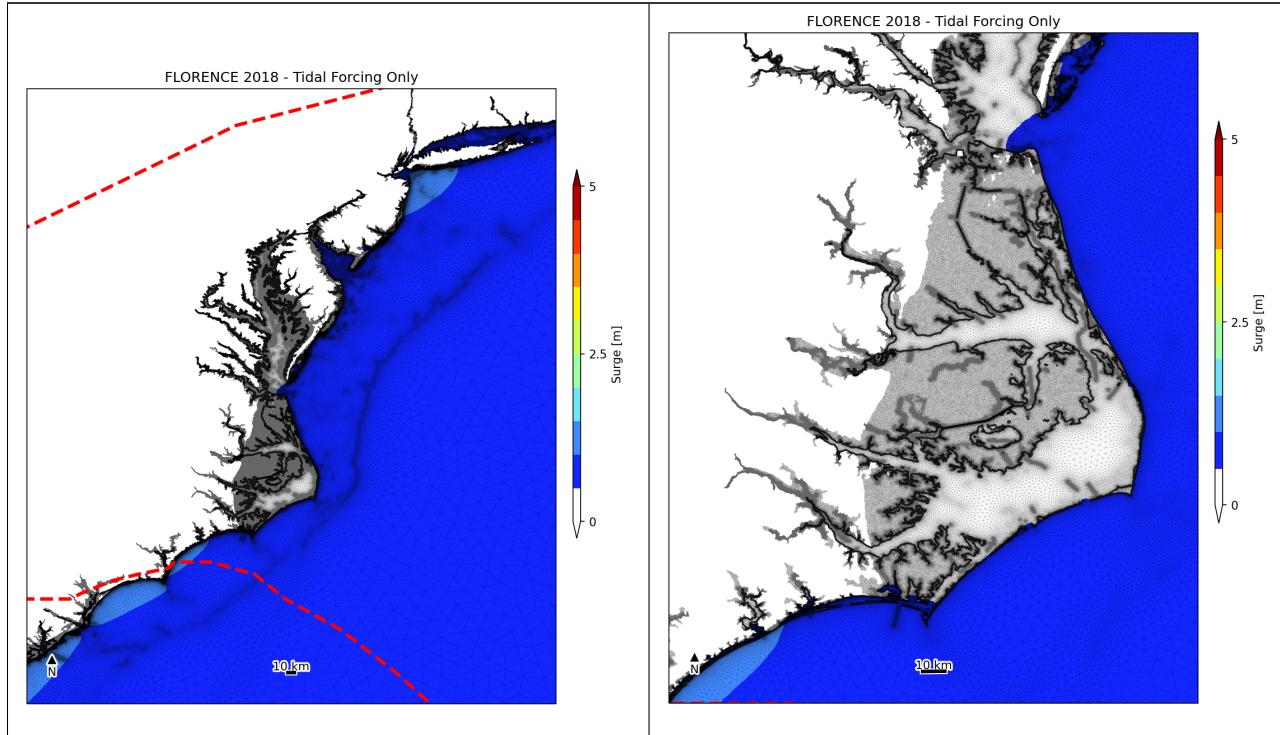


Table 5 Caption Text

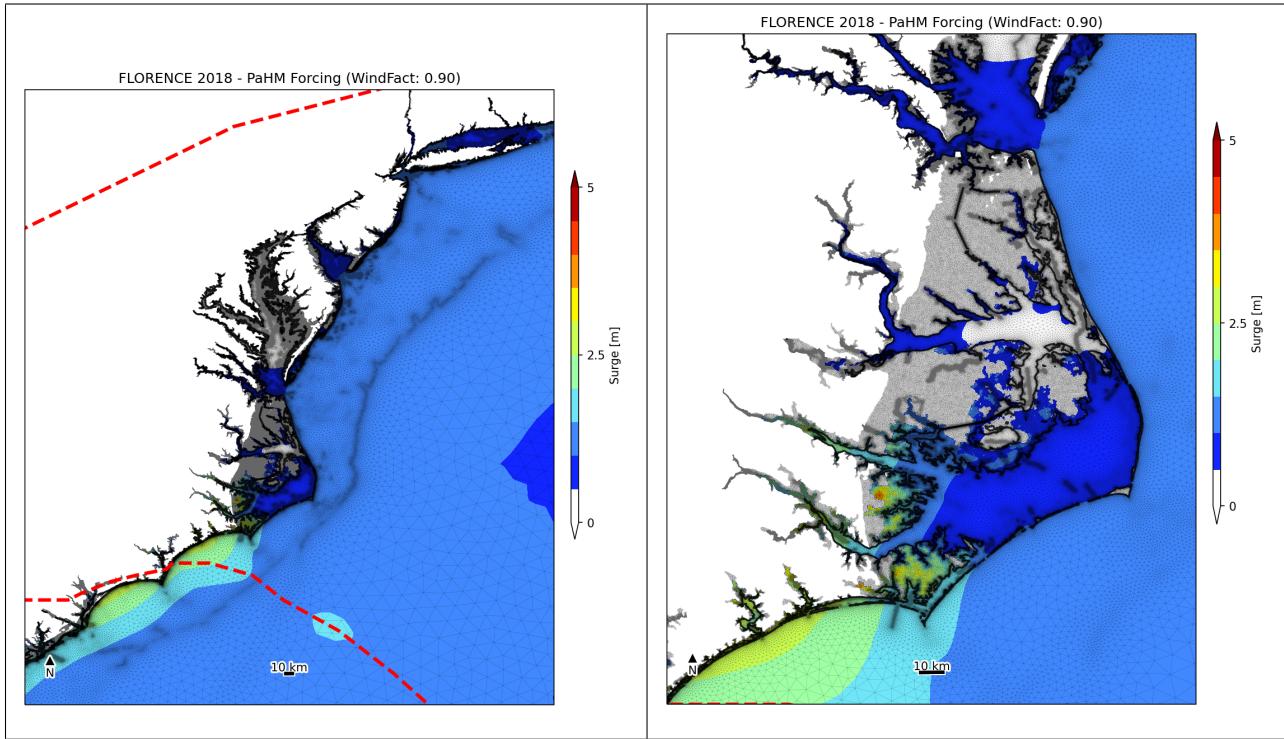
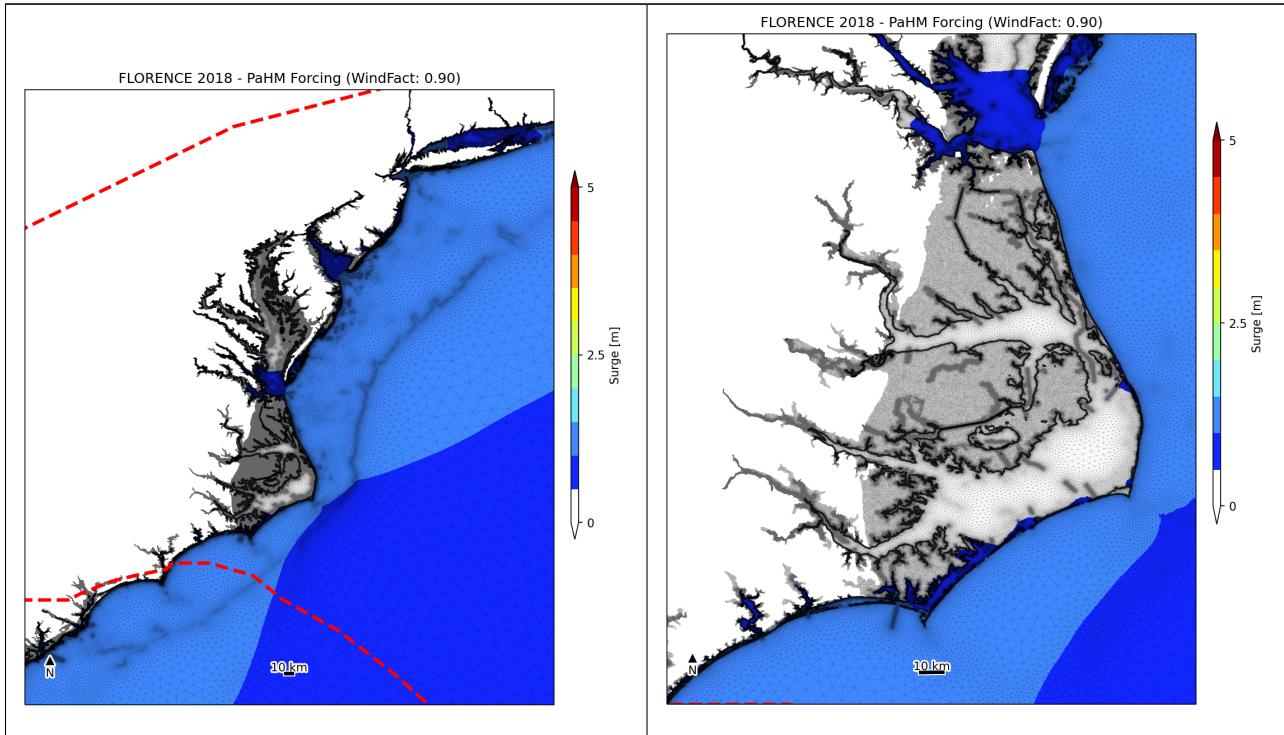


Table 6 Caption Text



9.3.1 Coupled Model Results and Discussion

Table 7 Caption Text

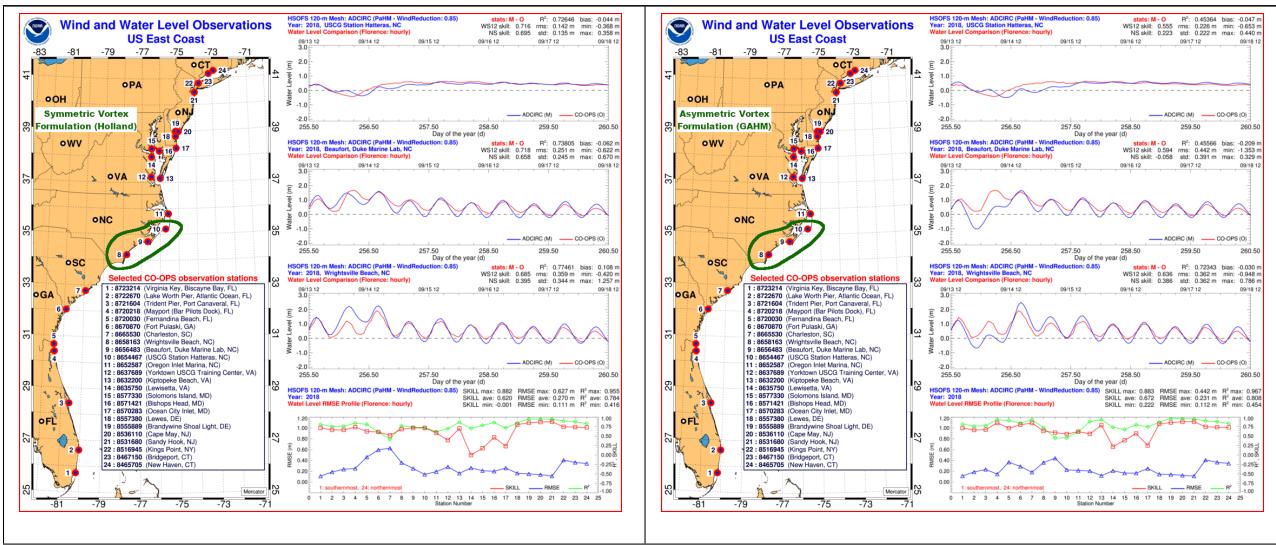


Table 8 Caption Text

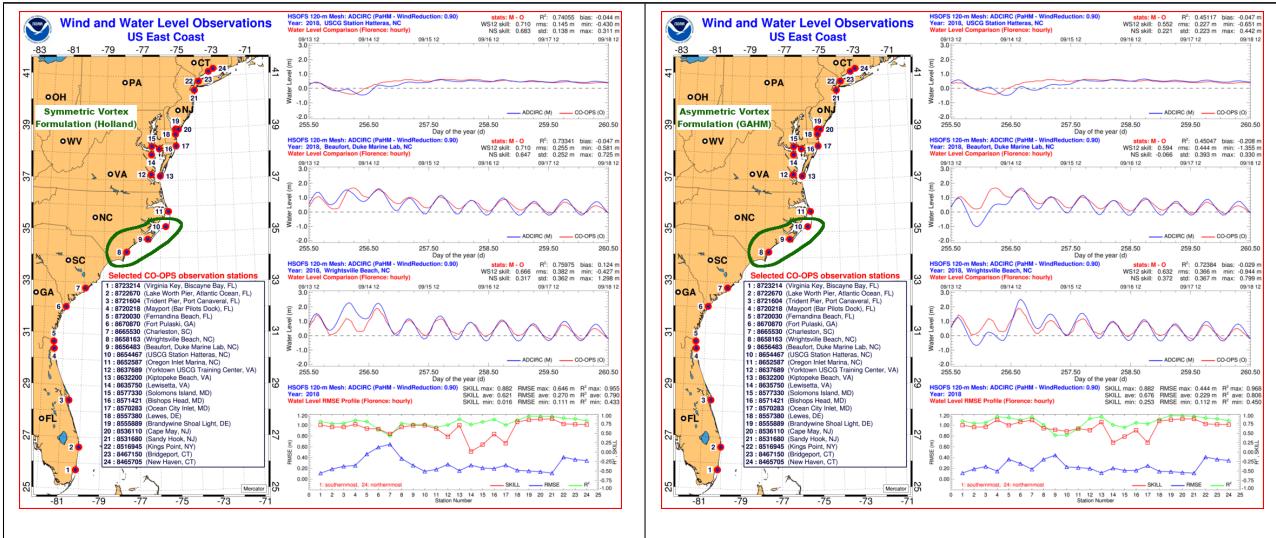
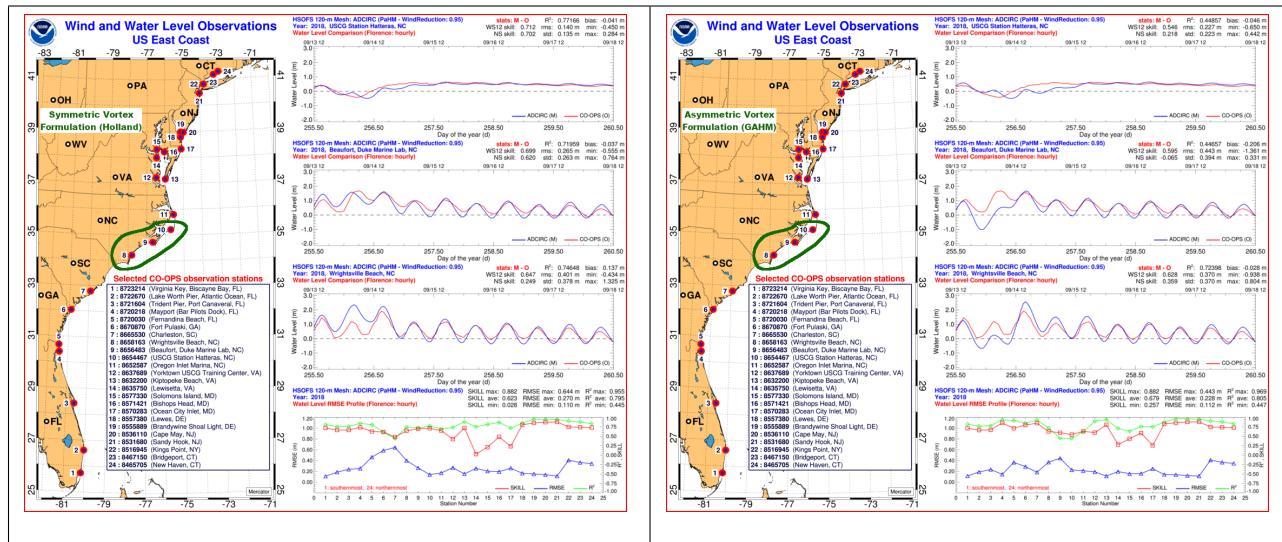


Table 9 Caption Text



9.4 Conclusions

10 List of Deliverables

11 Glossary

12 Credits

13 References

- [1] Jie Gao, Rick Luettich, and Jason Fleming. Generalization of the Holland Parametric Tropical Cyclone Model for Forecast Applications. 14th International Workshop on Wave Hindcasting and Forecasting / 5th Coastal Hazards Symposium / 2nd International Storm Surge Symposium, 2015.
- [2] Jie Gao. On the Surface Wind Stress for Storm Surge Modeling. PhD thesis, The University of North Carolina, Chapel Hill, NC, 2018.
- [3] Greg J. Holland, James I. Belanger, and Angela Fritz. A Revised Model for Radial Profiles of Hurricane Winds. Monthly Weather Review, 138:4393-4401, 2010.
- [4] Greg J. Holland. An Analytic Model of the Wind and Pressure Profiles in Hurricanes. Monthly Weather Review, 108:1212-1218, 1980.
- [5] Peter J. Vickery, Forrest J. Masters, Mark D. Powell, and Dhiraj Wadhera. Hurricane Hazard Modeling: The Past, Present and Future. Keynote lecture at the ICWE12 Cairns, Australia 2007, page 29, 2007.
- [6] H. E. Willoughby and M. E. Rahn. Parametric Representation of the Primary Hurricane Vortex. Part I: Observations and Evaluation of the Holland (1980) Model. Monthly Weather Review, 132:3033-3048, 2004.
- [7] H. E. Willoughby, R. W. R. Darling, and M. E. Rahn. Parametric Representation of the Primary Hurricane Vortex. Part II: A New Family of Sectionally Continuous Profiles. Monthly Weather Review, 134:1102-1120, 2005.

14 PaHM Code

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.

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

14.1 Third-Party Libraries

14.2 Functional Parts

14.3 Modules

14.4 Data Types

14.4.1 Data Types List

14.4.2 Data Fields

14.5 Files

14.5.1 File List

14.5.2 File Members

15 Modules Index

15.1 Modules List

Here is a list of all modules with brief descriptions:

csv_module	22
csv_parameters	47
csv_utilities	48
pahm_drivermod	52
pahm_global	57

pahm_mesh	77
pahm_messages	82
pahm_netcdfio	93
pahm_sizes	102
pahm_vortex	108
parwind	135
sortutils	149
timedateutils	163
utilities	186

16 Data Type Index

16.1 Data Types List

Here are the data types with brief descriptions:

pahm_netcdfio::adcirccoorddata_t	212
pahm_netcdfio::adcircvardata3d_t	215
pahm_netcdfio::adcircvardata_t	217
pahm_messages::allmessage	219
sortutils::arraycopy	220
sortutils::arrayequal	222
sortutils::arth	225
parwind::asymmetricvortexdata_t	227
parwind::besttrackdata_t	236
pahm_sizes::comparereals	243
utilities::cptogeogeo	245
csv_module::csv_file	248
csv_module::csv_string	256
pahm_netcdfio::filedata_t	257
pahm_sizes::fixnearwholereal	258
utilities::geotocpp	260

timedateutils::gregtojulday	262
parwind::hollanddata_t	267
sortutils::indexx	272
pahm_messages::logmessage	277
pahm_messages::screenmessage	279
utilities::sphericaldistance	280
timedateutils::splittatetimestring	284
sortutils::swap	286
timedateutils::timeconv	291
pahm_netcdfio::timedata_t	293

17 File Index

17.1 Files

Here is a list of all files with brief descriptions:

/home/takis/CSDL/parwinds-doc/src/csv_module.F90 For reading and writing CSV files	295
/home/takis/CSDL/parwinds-doc/src/csv_parameters.F90 Various parameters	314
/home/takis/CSDL/parwinds-doc/src/csv_utilities.F90 Utility routines	316
/home/takis/CSDL/parwinds-doc/src driver_mod.F90	320
/home/takis/CSDL/parwinds-doc/src/global.F90	323
/home/takis/CSDL/parwinds-doc/src/mesh.F90 Contains all the mesh related utilities	330
/home/takis/CSDL/parwinds-doc/src/messages.F90	335
/home/takis/CSDL/parwinds-doc/src/netcdfio.F90	341
/home/takis/CSDL/parwinds-doc/src/pahm.F90 Main PaHM program, calls Init, Run and Finalize procedures	353
/home/takis/CSDL/parwinds-doc/src/parwind.F90	356
/home/takis/CSDL/parwinds-doc/src/sizes.F90 Contains the definitions of various number types and utilities used in PaHM	400

/home/takis/CSDL/parwinds-doc/src/ sortutils.F90	404
/home/takis/CSDL/parwinds-doc/src/ timedateutils.F90	426
/home/takis/CSDL/parwinds-doc/src/ utilities.F90	441
/home/takis/CSDL/parwinds-doc/src/ vortex.F90	474

18 Module Documentation

18.1 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)]].
- subroutine [destroy_csv_file](#) (me)
Destroy a [[csv_file(type)]].
- subroutine [read_csv_file](#) (me, filename, header_row, skip_rows, status_ok)
Reads a CSV file.
- subroutine [open_csv_file](#) (me, filename, n_cols, status_ok, append)
Open a CSV file for writing.
- subroutine [close_csv_file](#) (me, status_ok)
Close a CSV file after writing.
- subroutine [add_cell](#) (me, val, int_fmt, real_fmt, trim_str)
Adds a cell to a CSV file.
- subroutine [add_vector](#) (me, val, int_fmt, real_fmt, trim_str)
Adds a vector to a CSV file.
- subroutine [add_matrix](#) (me, val, int_fmt, real_fmt, trim_str)
Adds a matrix to a CSV file.
- subroutine [next_row](#) (me)
Advance to the next row in the CSV file (write any blank cells that are necessary to finish the row).
- subroutine [get_header_csv_str](#) (me, header, status_ok)
Returns the header as a 'type(csv_string)' array ('read' must have already been called to read the file).
- subroutine [get_header_str](#) (me, header, status_ok)
Returns the header as a 'character(len=)' array.*
- subroutine [get_csv_data_as_str](#) (me, csv_data, status_ok)
Returns a 'character(len=)' array containing the csv data ('read' must have already been called to read the file).*
- pure elemental subroutine [to_real](#) (str, val, status_ok)
Converts a string to a 'real(wp)'.

- pure elemental subroutine `to_integer` (str, val, status_ok)
Converts a string to a 'integer(ip)'.
- pure elemental subroutine `to_logical` (str, val, status_ok)
Converts a string to a 'logical'.
- subroutine `variable_types` (me, itypes, status_ok)
Returns an array indicating the variable type of each columns.
- subroutine `infer_variable_type` (str, itype)
Infers the variable type.
- subroutine `csv_get_value` (me, row, col, val, status_ok)
Get an individual value from the 'csv_data' structure in the CSV class.
- subroutine `get_column` (me, icol, r, status_ok)
Return a column from a CSV file vector.
- subroutine `get_real_column` (me, icol, r, status_ok)
Return a column from a CSV file as a 'real(wp)' vector.
- subroutine `get_integer_column` (me, icol, r, status_ok)
Return a column from a CSV file as a 'integer(ip)' vector.
- subroutine `get_logical_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'logical' vector.
- subroutine `get_character_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'character(len=)' vector.*
- subroutine `get_csv_string_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'type(csv_string)' vector.
- subroutine `tokenize_csv_line` (me, line, cells)
Tokenize a line from a CSV file.
- integer function `number_of_lines_in_file` (iunit)
Returns the number of lines in a text file.
- subroutine `read_line_from_file` (me, iunit, line, status_ok)
Reads the next line from a file.
- pure subroutine `split` (str, token, chunk_size, vals)
Splits a character string using a token.

Variables

- integer, parameter, public `csv_type_string` = 1
- integer, parameter, public `csv_type_double` = 2
- integer, parameter, public `csv_type_integer` = 3
- integer, parameter, public `csv_type_logical` = 4
- real(wp), parameter `zero` = 0.0_wp

18.1.1 Function/Subroutine Documentation

```
18.1.1.1 add_cell() subroutine csv_module::add_cell (
    class(csv_file), intent(inout) me,
    class(*), intent(in) val,
    character(len=*), intent(in), optional int_fmt,
    character(len=*), intent(in), optional real_fmt,
    logical, intent(in), optional trim_str ) [private]
```

Adds a cell to a CSV file.

Parameters

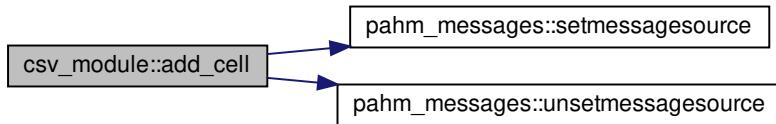
in,out	<i>me</i>	The input/output csv_file class
in	<i>val</i>	The value to add
in	<i>int_fmt</i>	If ' <i>val</i> ' is an integer, use this format string (optional)
in	<i>real_fmt</i>	If ' <i>val</i> ' is a real, use this format string (optional)
out	<i>trim_str</i>	If ' <i>val</i> ' is a string, then trim it (optional)

Definition at line 537 of file [csv_module.F90](#).

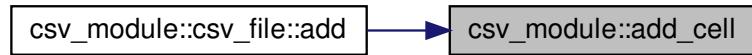
References [csv_parameters::default_int_fmt](#), [csv_parameters::default_real_fmt](#), [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::unsetmessagesource\(\)](#), and [pahm_sizes::wp](#).

Referenced by [csv_module::csv_file::add\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.1.1.2 add_matrix() subroutine csv_module::add_matrix (  
    class(csv\_file), intent(inout) me,  
    class(*), dimension(:, :), intent(in) val,  
    character(len=*), intent(in), optional int_fmt,  
    character(len=*), intent(in), optional real_fmt,  
    logical, intent(in), optional trim_str ) [private]
```

Adds a matrix to a CSV file.

Each row is added as a new line. Line breaks are added at the end of each line (in this way it differs from the other 'add' routines).

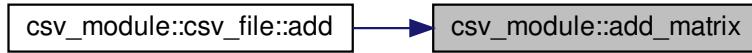
Parameters

<i>in,out</i>	<i>me</i>	The input/output csv_file class
<i>in</i>	<i>val</i>	The values to add
<i>in</i>	<i>int_fmt</i>	If 'val' is an integer, use this format string (optional)
<i>in</i>	<i>real_fmt</i>	If 'val' is a real, use this format string (optional)
<i>out</i>	<i>trim_str</i>	If 'val' is a string, then trim it (optional)

Definition at line 714 of file [csv_module.F90](#).

Referenced by [csv_module::csv_file::add\(\)](#).

Here is the caller graph for this function:



```
18.1.1.3 add_vector() subroutine csv_module::add_vector (
    class(csv\_file), intent(inout) me,
    class(*), dimension(:), intent(in) val,
    character(len=*), intent(in), optional int_fmt,
    character(len=*), intent(in), optional real_fmt,
    logical, intent(in), optional trim_str ) [private]
```

Adds a vector to a CSV file.

Each element is added as a cell to the current line.

Parameters

<i>in,out</i>	<i>me</i>	The input/output csv_file class
<i>in</i>	<i>val</i>	The values to add
<i>in</i>	<i>int_fmt</i>	If 'val' is an integer, use this format string (optional)
<i>in</i>	<i>real_fmt</i>	If 'val' is a real, use this format string (optional)
<i>out</i>	<i>trim_str</i>	If 'val' is a string, then trim it (optional)

Definition at line 659 of file [csv_module.F90](#).

Referenced by [csv_module::csv_file::add\(\)](#).

Here is the caller graph for this function:



18.1.1.4 close_csv_file() subroutine csv_module::close_csv_file (class([csv_file](#)), intent(inout) *me*, logical, intent(out) *status_ok*) [private]

Close a CSV file after writing.

Parameters

in,out	<i>me</i>	The input/ouput csv_file class
out	<i>status_ok</i>	Status flag

Definition at line 499 of file [csv_module.F90](#).

18.1.1.5 csv_get_value() subroutine csv_module::csv_get_value (class([csv_file](#)), intent(inout) *me*, integer, intent(in) *row*, integer, intent(in) *col*, class(*), intent(out) *val*, logical, intent(out) *status_ok*) [private]

Get an individual value from the 'csv_data' structure in the CSV class.

The output 'val' can be an 'integer(ip)', 'real(wp)', 'logical', or 'character(len=*)' variable.

Parameters

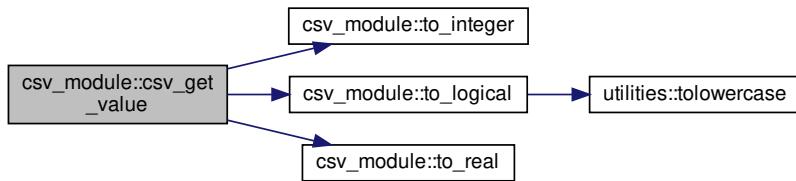
in,out	<i>me</i>	The input/ouput csv_file class
in	<i>row</i>	The row number
in	<i>col</i>	The column number
out	<i>val</i>	The returned value
out	<i>status_ok</i>	Status flag

Definition at line 1200 of file [csv_module.F90](#).

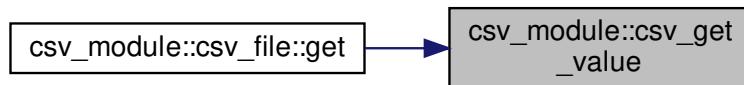
References [to_integer\(\)](#), [to_logical\(\)](#), [to_real\(\)](#), and [pahm_sizes::wp](#).

Referenced by [csv_module::csv_file::get\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.6 destroy_csv_file() subroutine csv_module::destroy_csv_file (class([csv_file](#)), intent(out) me) [private]

Destroy a [[[csv_file\(type\)](#)]].

Parameters

out	me	The ouput csv_file class
-----	----	--

Definition at line 229 of file [csv_module.F90](#).

18.1.1.7 get_character_column() subroutine csv_module::get_character_column (class([csv_file](#)), intent(inout) me,

```
integer, intent(in) icol,
character(len=*), dimension(:), intent(out), allocatable r,
logical, intent(out) status_ok ) [private]
```

Convert a column from a '[csv_string](#)' matrix to a 'character(len=*)' vector.

Parameters

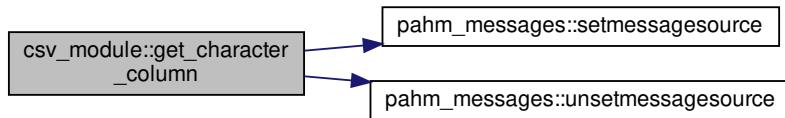
in,out	<i>me</i>	The input/output csv_file class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

Definition at line 1480 of file [csv_module.F90](#).

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

Referenced by [csv_module::csv_file::get\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.1.1.8 get_column() subroutine csv_module::get_column (
    class(csv\_file), intent(inout) me,
    integer, intent(in) icol,
    class(*), dimension(:), intent(out) r,
    logical, intent(out) status_ok ) [private]
```

Return a column from a CSV file vector.

Parameters

in,out	<i>me</i>	The input/output csv_file class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column. It is assumed to have been allocated to the correct size by the caller ('n_rows').
out	<i>status_ok</i>	Status flag

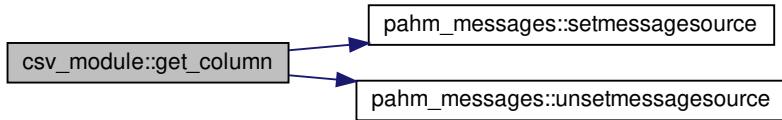
Note

This routine requires that the 'r' array already be allocated. This is because Fortran doesn't want to allow to you pass a non-polymorphic variable into a routine with a dummy variable with 'class(*),dimension(← :),allocatable,intent(out)' attributes.

Definition at line 1260 of file [csv_module.F90](#).

References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::unsetmessagesource\(\)](#), [pahm_sizes::wp](#), and [zero](#).

Here is the call graph for this function:



```

18.1.1.9 get_csv_data_as_str() subroutine csv_module::get_csv_data_as_str (
    class(csv\_file), intent(inout) me,
    character(len=*), dimension(:, :, ), intent(out), allocatable csv_data,
    logical, intent(out) status_ok ) [private]
  
```

Returns a 'character(len=*)' array containing the csv data ('read' must have already been called to read the file).

Parameters

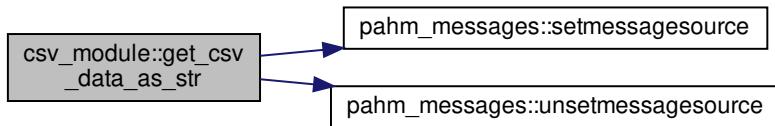
in,out	<i>me</i>	The input/output csv_file class
out	<i>csv_data</i>	The data
out	<i>status_ok</i>	Status flag

Definition at line 897 of file [csv_module.F90](#).

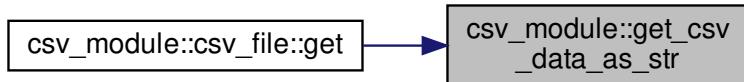
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::csv_file::get\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.10 get_csv_string_column() subroutine `csv_module::get_csv_string_column` (
 `class(csv_file)`, intent(inout) `me`,
 `integer`, intent(in) `icol`,
 `type(csv_string)`, dimension(:), intent(out), allocatable `r`,
 `logical`, intent(out) `status_ok`) [private]

Convert a column from a 'csv_string' matrix to a 'type(csv_string)' vector.

Parameters

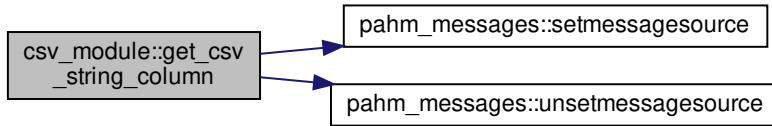
in,out	<code>me</code>	The input/output <code>csv_file</code> class
in	<code>icol</code>	The column number
out	<code>r</code>	The returned column
out	<code>status_ok</code>	Status flag

Definition at line 1524 of file [csv_module.F90](#).

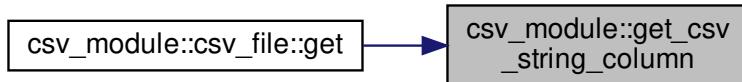
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::csv_file::get\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.11 get_header_csv_str() subroutine `csv_module::get_header_csv_str (`
`class(csv_file), intent(inout) me,`
`type(csv_string), dimension(:), intent(out), allocatable header,`
`logical, intent(out) status_ok) [private]`

Returns the header as a 'type(csv_string)' array ('read' must have already been called to read the file).

Parameters

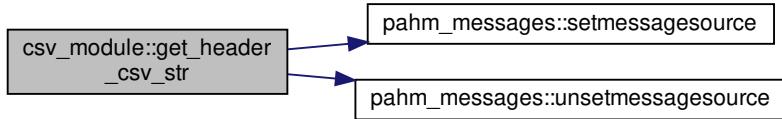
in,out	<i>me</i>	The input/ouput <code>csv_file</code> class
out	<i>header</i>	The header of the CSV file
out	<i>status_ok</i>	Status flag

Definition at line 799 of file `csv_module.F90`.

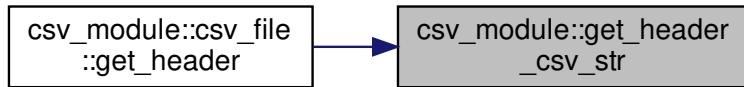
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::csv_file::get_header\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.12 get_header_str() subroutine `csv_module::get_header_str (`
`class(csv_file), intent(inout) me,`
`character(len=*), dimension(:), intent(out), allocatable header,`
`logical, intent(out) status_ok) [private]`

Returns the header as a 'character(len=*)' array.

('read' must have already been called to read the file).

Parameters

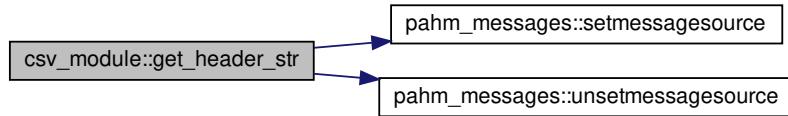
in,out	<i>me</i>	The input/output csv_file class
out	<i>header</i>	The header of the CSV file
out	<i>status_ok</i>	Status flag

Definition at line 848 of file [csv_module.F90](#).

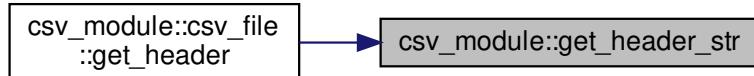
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::csv_file::get_header\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.13 get_integer_column() subroutine `csv_module::get_integer_column (`
 `class(csv_file), intent(inout) me,`
 `integer, intent(in) icol,`
 `integer(ip), dimension(:), intent(out), allocatable r,`
 `logical, intent(out) status_ok) [private]`

Return a column from a CSV file as a 'integer(ip)' vector.

Parameters

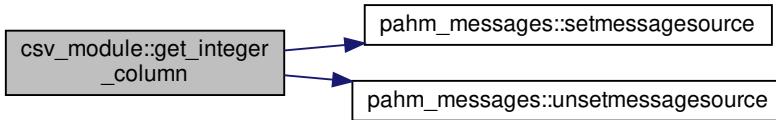
in,out	<i>me</i>	The input/output csv_file class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

Definition at line 1392 of file [csv_module.F90](#).

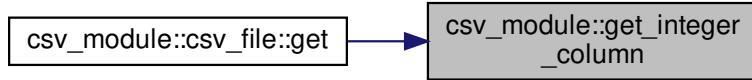
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::csv_file::get\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.14 get_logical_column() subroutine `csv_module::get_logical_column (`
 `class(csv_file), intent(inout) me,`
 `integer, intent(in) icol,`
 `logical, dimension(:), intent(out), allocatable r,`
 `logical, intent(out) status_ok) [private]`

Convert a column from a '`csv_string`' matrix to a 'logical' vector.

Parameters

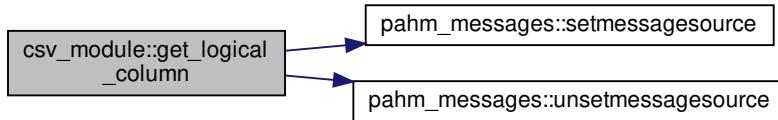
in,out	<i>me</i>	The input/output <code>csv_file</code> class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

Definition at line 1436 of file `csv_module.F90`.

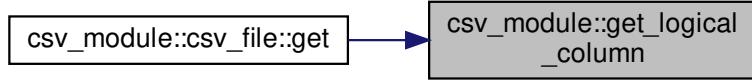
References `pahm_messages::error`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

Referenced by `csv_module::csv_file::get()`.

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.15 get_real_column() subroutine `csv_module::get_real_column` (

<code>class(csv_file), intent(inout) me,</code> <code>integer, intent(in) icol,</code> <code>real(wp), dimension(:), intent(out), allocatable r,</code> <code>logical, intent(out) status_ok) [private]</code>
--

Return a column from a CSV file as a 'real(wp)' vector.

Parameters

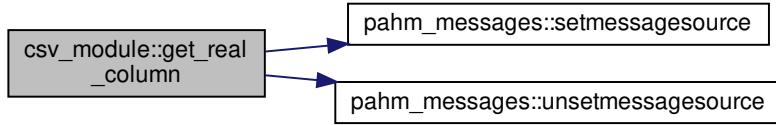
<code>in,out</code>	<code>me</code>	The input/output <code>csv_file</code> class
<code>in</code>	<code>icol</code>	The column number
<code>out</code>	<code>r</code>	The returned column
<code>out</code>	<code>status_ok</code>	Status flag

Definition at line 1348 of file `csv_module.F90`.

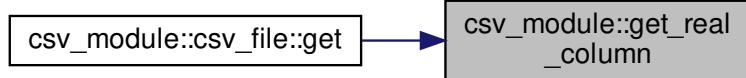
References `pahm_messages::error`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

Referenced by `csv_module::csv_file::get()`.

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.1.1.16 infer_variable_type() subroutine csv_module::infer_variable_type (
    character(len=*), intent(in) str,
    integer, intent(out) itype ) [private]
```

Infers the variable type.

Infers the variable type, assuming the following precedence:

```
integer, double, logical, character
```

Parameters

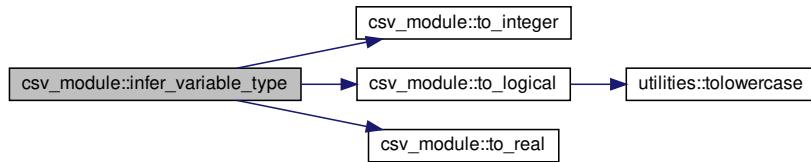
in	<i>str</i>	The input string
out	<i>itype</i>	The type of the cell value 1: a character string cell 2: a 'real(wp)' cell 3: an 'integer(ip)' cell 4: a logical cell
out	<i>status_ok</i>	Status flag

Definition at line 1142 of file [csv_module.F90](#).

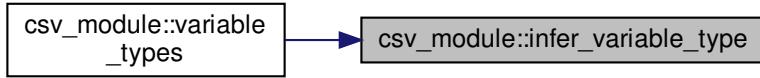
References `csv_type_double`, `csv_type_integer`, `csv_type_logical`, `csv_type_string`, `to_integer()`, `to_logical()`, and `to_real()`.

Referenced by `variable_types()`.

Here is the call graph for this function:



Here is the caller graph for this function:



```

18.1.1.17 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

<code>out</code>	<code>me</code>	The ouput <code>csv_file</code> class
<code>in</code>	<code>quote</code>	Can only be one character (optional, default is "")
<code>in</code>	<code>delimiter</code>	Can only be one character (optional, default is ',')
<code>in</code>	<code>enclose_strings_in_quotes</code>	Logical flag; if true, all string cells will be enclosed in quotes (optional, default is 'T')

Parameters

in	<i>enclose_all_in_quotes</i>	Logical flag; if true, <i>all</i> cells will be enclosed in quotes (optional, default is 'F')
in	<i>logical_true_string</i>	Logical flag; when writing a logical 'true' value to a CSV file, this is the string to use (optional, default is 'T')
in	<i>logical_false_string</i>	Logical flag; when writing a logical 'false' value to a CSV file, this is the string to use (optional, default is 'T')
in	<i>chunk_size</i>	Factor for expanding vectors (default is 100)

Definition at line 166 of file [csv_module.F90](#).

```
18.1.1.18 next_row() subroutine csv_module::next_row (
    class(csv\_file), intent(inout) me ) [private]
```

Advance to the next row in the CSV file (write any blank cells that are necessary to finish the row).

Parameters

in,out	<i>me</i>	The input/output csv_file class
--------	-----------	---

Definition at line 749 of file [csv_module.F90](#).

```
18.1.1.19 number_of_lines_in_file() integer function csv_module::number_of_lines_in_file (
    integer, intent(in) iunit ) [private]
```

Returns the number of lines in a text file.

Parameters

in	<i>iunit</i>	The file unit number (assumed to be open)
----	--------------	---

Returns

n_lines The number of lines in the file

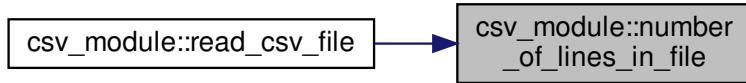
Note

It rewinds the file back to the beginning when finished.

Definition at line 1630 of file [csv_module.F90](#).

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

Here is the caller graph for this function:



```
18.1.1.20 open_csv_file() subroutine csv_module::open_csv_file (
    class(csv\_file), intent(inout) me,
    character(len=*), intent(in) filename,
    integer, intent(in) n_cols,
    logical, intent(out) status_ok,
    logical, intent(in), optional append ) [private]
```

Open a CSV file for writing.

Use 'initialize' to set options for the CSV file.

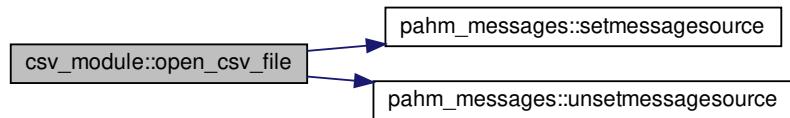
Parameters

<i>in,out</i>	<i>me</i>	The input/output csv_file class
<i>in</i>	<i>filename</i>	The CSV file to open
<i>in</i>	<i>n_cols</i>	The number of columns in the file
<i>out</i>	<i>status_ok</i>	Status flag
<i>in</i>	<i>append</i>	Logical flag, append if file exists (optional)

Definition at line 437 of file [csv_module.F90](#).

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

Here is the call graph for this function:



```
18.1.1.21 read_csv_file() subroutine csv_module::read_csv_file (
    class(csv\_file), intent(inout) me,
    character(len=*), intent(in) filename,
    integer, intent(in), optional header_row,
    integer, dimension(:), intent(in), optional skip_rows,
    logical, intent(out) status_ok ) [private]
```

Reads a CSV file.

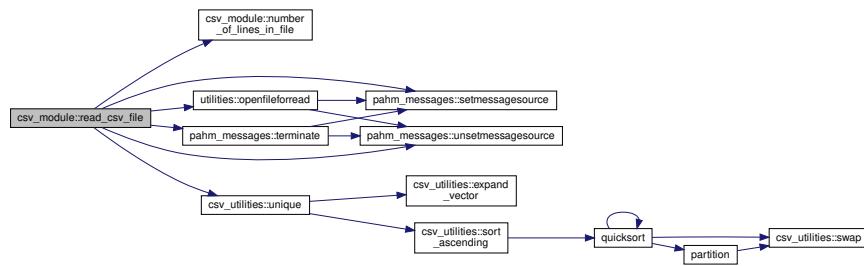
Parameters

in,out	<i>me</i>	The input/output csv_file class
in	<i>filename</i>	The CSV file to open
out	<i>status_ok</i>	Status flag
in	<i>header_row</i>	The header row (optional)
in	<i>skip_rows</i>	The number of rows to skip (optional)

Definition at line 259 of file [csv_module.F90](#).

References [pahm_messages::error](#), [pahm_messages::info](#), [pahm_global::lun_btrk](#), [number_of_lines_in_file\(\)](#), [utilities::openfileforread\(\)](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::terminate\(\)](#), [csv_utilities::unique\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Here is the call graph for this function:



```
18.1.1.22 read_line_from_file() subroutine csv_module::read_line_from_file (
    class(csv\_file), intent(in) me,
    integer, intent(in) iunit,
    character(len=:), intent(out), allocatable line,
    logical, intent(out) status_ok ) [private]
```

Reads the next line from a file.

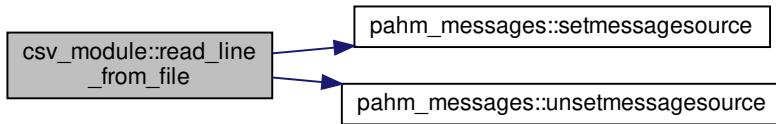
Parameters

<i>in,out</i>	<i>me</i>	The input/output <code>csv_file</code> class
<i>in</i>	<i>iunit</i>	The file unit number (assumed to be open)
<i>out</i>	<i>line</i>	The line in the file
<i>out</i>	<i>status_ok</i>	Status flag

Definition at line 1672 of file `csv_module.F90`.

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

Here is the call graph for this function:



```

18.1.1.23 split() pure subroutine csv_module::split (
    character(len=*), intent(in) str,
    character(len=*), intent(in) token,
    integer, intent(in) chunk_size,
    type(csv_string), dimension(:), intent(out), allocatable vals ) [private]
  
```

Splits a character string using a token.

This routine is inspired by the Python `split` function.

```

### Example
'''Fortran
character(len,:),allocatable :: s
type(csv_string),dimension(:),allocatable :: vals
s = '1,2,3,4,5'
call split(s,',',vals)
'''
  
```

Parameters

<i>in</i>	<i>str</i>	The input string
<i>in</i>	<i>token</i>	The tokens to use in splitting the string
<i>out</i>	<i>chunk_size</i>	The chunk size to use for expanding vectors
<i>out</i>	<i>vals</i>	The returned values

Warning

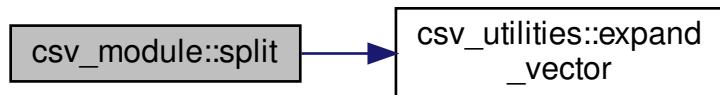
Does not account for tokens contained within quotes string

Definition at line 1745 of file [csv_module.F90](#).

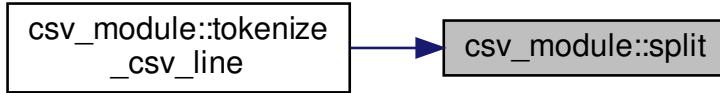
References [csv_utilities::expand_vector\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.1.1.24 to_integer() pure elemental subroutine csv_module::to_integer (
    character(len=*), intent(in) str,
    integer(ip), intent(out) val,
    logical, intent(out) status_ok ) [private]
```

Converts a string to a 'integer(ip)'.

Parameters

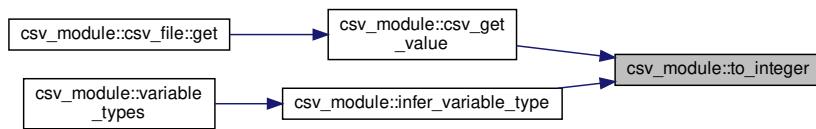
in	<i>str</i>	The input string
out	<i>val</i>	The converted value
out	<i>status_ok</i>	Status flag

Definition at line 986 of file [csv_module.F90](#).

References [csv_parameters::default_int_fmt](#).

Referenced by [csv_get_value\(\)](#), and [infer_variable_type\(\)](#).

Here is the caller graph for this function:



18.1.1.25 to_logical() pure elemental subroutine `csv_module::to_logical` (
`character(len=*)`, intent(in) `str`,
`logical`, intent(out) `val`,
`logical`, intent(out) `status_ok`) [private]

Converts a string to a 'logical'.

The string match is not case sensitive.

Evaluates to '.true.' for strings ['1','t','true','.true.'].
Evaluates to '.false.' for strings ['0','f','false','.false.'].

Parameters

in	<code>str</code>	The input string
out	<code>val</code>	The converted value
out	<code>status_ok</code>	Status flag

Definition at line 1028 of file [csv_module.F90](#).

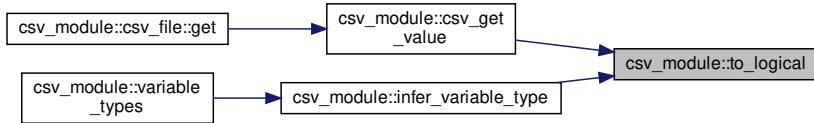
References [utilities::tolowercase\(\)](#).

Referenced by [csv_get_value\(\)](#), and [infer_variable_type\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.1.1.26 `to_real()` pure elemental subroutine `csv_module::to_real` (character(len=*), intent(in) `str`, real(wp), intent(out) `val`, logical, intent(out) `status_ok`) [private]

Converts a string to a 'real(wp)'.

Parameters

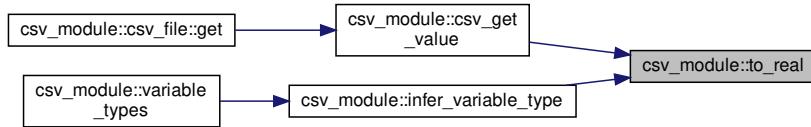
in	<code>str</code>	The input string
out	<code>val</code>	The converted value
out	<code>status_ok</code>	Status flag

Definition at line 948 of file [csv_module.F90](#).

References [zero](#).

Referenced by [csv_get_value\(\)](#), and [infer_variable_type\(\)](#).

Here is the caller graph for this function:



18.1.1.27 tokenize_csv_line() subroutine `csv_module:::tokenize_csv_line (`
 `class(csv_file), intent(inout) me,`
 `character(len=*), intent(in) line,`
 `type(csv_string), dimension(:), intent(out), allocatable cells) [private]`

Tokenize a line from a CSV file.

The result is an array of 'csv_string' types.

Quotes are removed if the entire cell is contained in quotes.

Parameters

in, out	<i>me</i>	The input/output <code>csv_file</code> class
in	<i>line</i>	The line in the CSV file
out	<i>cells</i>	The tokenized cell values
out	<i>status_ok</i>	Status flag

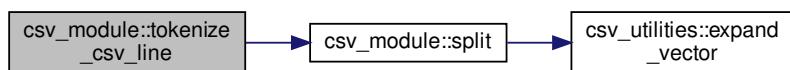
Warning

It does not account for delimiters in quotes (these are treated as a new cell). Need to fix!

Definition at line 1572 of file `csv_module.F90`.

References `split()`.

Here is the call graph for this function:



```
18.1.1.28 variable_types() subroutine csv_module::variable_types (
    class(csv\_file), intent(inout) me,
    integer, dimension(:), intent(out), allocatable itypes,
    logical, intent(out) status_ok ) [private]
```

Returns an array indicating the variable type of each columns.

Parameters

in,out	<i>me</i>	The input/output csv_file class
out	<i>itypes</i>	The type of the cell values 1: a character string cell 2: a 'real(wp)' cell 3: an 'integer(ip)' cell 4: a logical cell
out	<i>status_ok</i>	Status flag

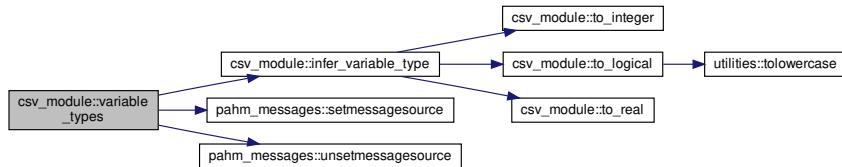
Note

The first element in the column is used to determine the type.

Definition at line 1088 of file [csv_module.F90](#).

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

Here is the call graph for this function:



18.1.2 Variable Documentation

18.1.2.1 csv_type_double integer, parameter, public csv_module::csv_type_double = 2

Definition at line 31 of file [csv_module.F90](#).

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

18.1.2.2 csv_type_integer integer, parameter, public csv_module::csv_type_integer = 3

Definition at line 32 of file [csv_module.F90](#).

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

18.1.2.3 csv_type_logical integer, parameter, public csv_module::csv_type_logical = 4

Definition at line 33 of file [csv_module.F90](#).

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

18.1.2.4 csv_type_string integer, parameter, public csv_module::csv_type_string = 1

Definition at line 30 of file [csv_module.F90](#).

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

18.1.2.5 zero real(wp), parameter csv_module::zero = 0.0_wp [private]

Definition at line 35 of file [csv_module.F90](#).

Referenced by [get_column\(\)](#), and [to_real\(\)](#).

18.2 csv_parameters Module Reference

Variables

- integer(ip), parameter, public [max_real_str_len](#) = 27
- character(len= *), parameter, public [default_real_fmt](#) = '(E27.17E4)'
- integer(ip), parameter, public [max_integer_str_len](#) = 256
- character(len= *), parameter, public [default_int_fmt](#) = '(I256)'

18.2.1 Variable Documentation

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

Definition at line 32 of file [csv_parameters.F90](#).

Referenced by [csv_module::add_cell\(\)](#), and [csv_module::to_integer\(\)](#).

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

Definition at line 26 of file [csv_parameters.F90](#).

Referenced by [csv_module::add_cell\(\)](#).

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

Definition at line 29 of file [csv_parameters.F90](#).

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

Definition at line 23 of file [csv_parameters.F90](#).

18.3 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)
Finds the unique elements in a vector of integers.
- subroutine, public [sortAscending](#) (ivec)
Sorts an integer array ivec in increasing order.
- pure elemental subroutine [swap](#) (i1, i2)
Swap two integer values.

Variables

- integer, parameter [max_size_for_insertion_sort](#) = 20

18.3.1 Function/Subroutine Documentation

18.3.1.1 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.

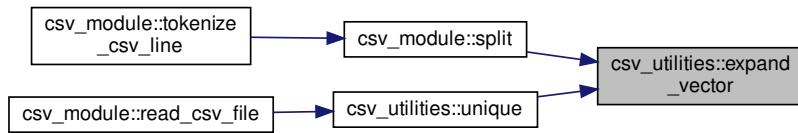
Parameters

in,out	<i>vec</i>	The input integer vector (input/output)
in,out	<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)
in	<i>chunk_size</i>	Allocate <i>vec</i> in blocks of this size (>0)
in	<i>val</i>	The value to add to <i>vec</i> (optional)
in	<i>finished</i>	Set to true to return <i>vec</i> as its correct size (<i>n</i>) (optional)

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

Referenced by [csv_module::split\(\)](#), and [unique\(\)](#).

Here is the caller graph for this function:



18.3.1.2 sortAscending() subroutine, public csv_utilities::sortAscending (

```
    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).

Parameters

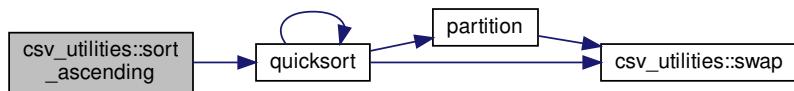
in, out	ivec	A vector of integers
---------	------	----------------------

Definition at line 167 of file [csv_utilities.F90](#).

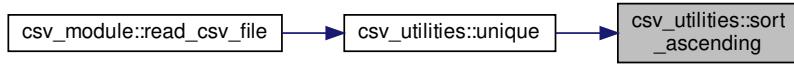
References [quicksort\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.3.1.3 swap() pure elemental subroutine `csv_utilities::swap` (
integer, intent(inout) `i1`,
integer, intent(inout) `i2`) [private]

Swap two integer values.

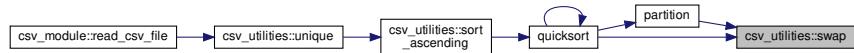
Parameters

in, out	<code>i1</code>	The first integer value to swap
in, out	<code>i2</code>	The second integer value to swap

Definition at line 259 of file [csv_utilities.F90](#).

Referenced by [partition\(\)](#), and [quicksort\(\)](#).

Here is the caller graph for this function:



18.3.1.4 unique() integer function, dimension(:), allocatable, public csv_utilities::unique (
 integer, dimension(:), intent(in) vec,
 integer, intent(in) chunk_size)

Finds the unique elements in a vector of integers.

Parameters

in	<i>vec</i>	A vector of integers
in	<i>chunk_size</i>	Chunk size for adding to arrays

Returns

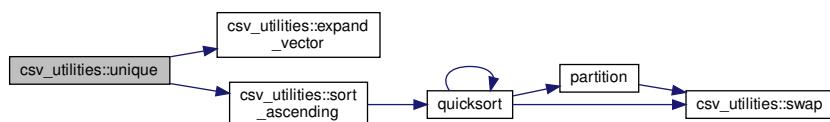
ivec_unique The unique elements of the vector "vec"

Definition at line 119 of file [csv_utilities.F90](#).

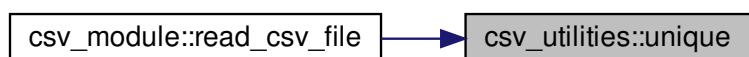
References [expand_vector\(\)](#), and [sortAscending\(\)](#).

Referenced by [csv_module::read_csv_file\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.3.2 Variable Documentation

18.3.2.1 max_size_for_insertion_sort integer, parameter csv_utilities::max_size_for_insertion_sort = 20 [private]

Definition at line 23 of file [csv_utilities.F90](#).

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

18.4 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](#)

18.4.1 Function/Subroutine Documentation

18.4.1.1 getprogramcmdlargs() subroutine pahm_drivermod::getprogramcmdlargs

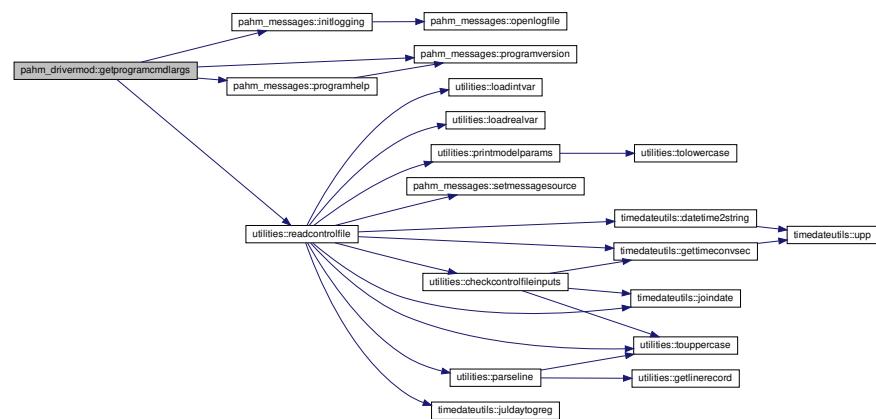
Prints on the screen the help system of the PaHM program.

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

References [pahm_global::controlfilename](#), [pahm_messages::initlogging\(\)](#), [pahm_messages::programhelp\(\)](#), [pahm_messages::programver\(\)](#) and [utilities::readcontrolfile\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:

**18.4.1.2 pahm_finalize()** subroutine pahm_drivermod::pahm_finalize

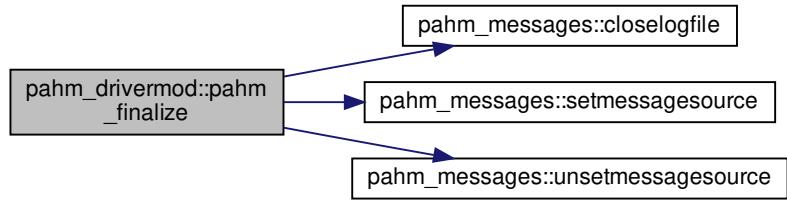
Subroutine to finalize a PaHM run.

Definition at line 211 of file [driver_mod.F90](#).

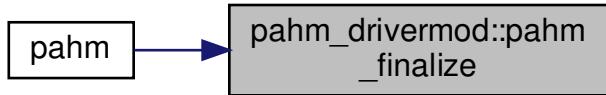
References [pahm_messages::closelogfile\(\)](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.4.1.3 `pahm_init()` subroutine `pahm_drivermod::pahm_init`

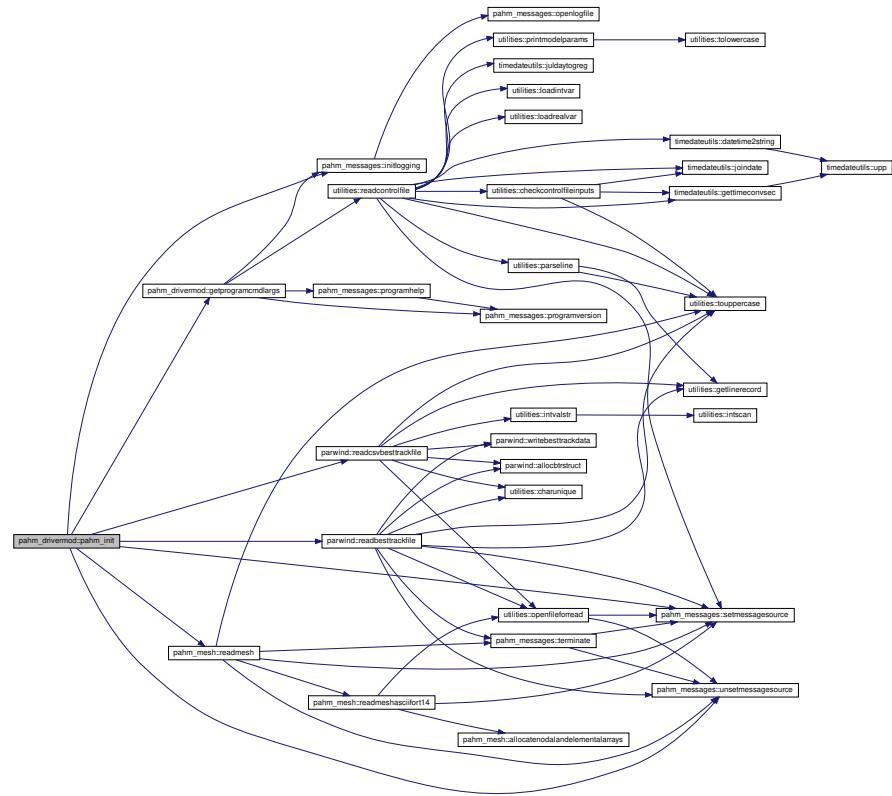
Subroutine to initialize a PaHM run.

Definition at line 94 of file [driver_mod.F90](#).

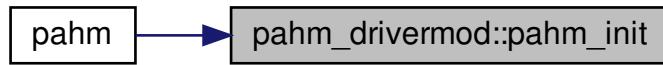
References `cnttimebegin`, `cnttimeend`, `getprogramcmdlargs()`, `pahm_messages::initlogging()`, `pahm_global::noutdt`, `parwind::readbesttrackfile()`, `parwind::readcsvbesttrackfile()`, `pahm_mesh::readmesh()`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Subroutine to run PaHM (timestepping).

Parameters

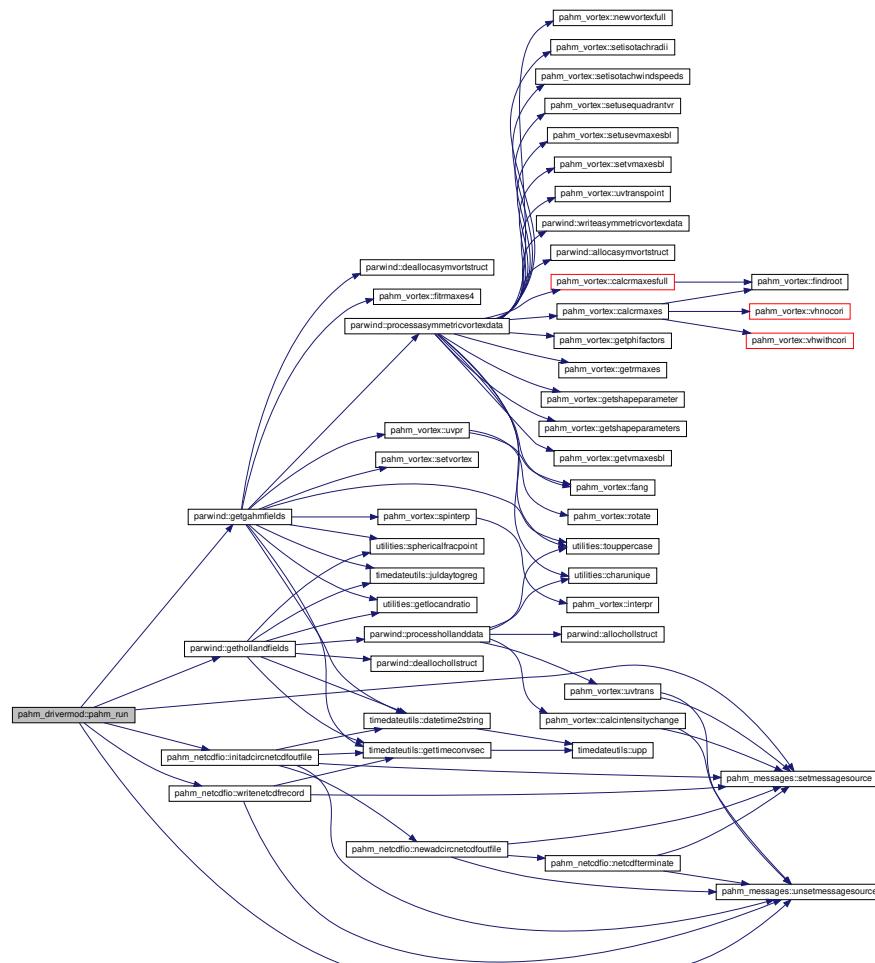
in *nTimeSTP* The timestep index (optional)

Definition at line 139 of file [driver_mod.F90](#).

References `cnttimebegin`, `cnttimeend`, `pahm_messages::error`, `parwind::getgahmfields()`, `parwind::gethollandfields()`, `pahm_netcdfio::initadcircnetcdfoutfile()`, `pahm_global::modeltype`, `pahm_global::outfilename`, `pahm_global::outfilenamespecified`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, `pahm_messages::unsetmessagesource()`, and `pahm_netcdfio::writenetcdfrecord()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.4.2 Variable Documentation

18.4.2.1 cnttimebegin integer, save pahm_drivermod::cnttimebegin

Definition at line 23 of file [driver_mod.F90](#).

Referenced by [pahm_init\(\)](#), and [pahm_run\(\)](#).

18.4.2.2 cnttimeend integer, save pahm_drivermod::cnttimeend

Definition at line 23 of file [driver_mod.F90](#).

Referenced by [pahm_init\(\)](#), and [pahm_run\(\)](#).

18.5 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`name = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) `controlfile`name = 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_windreduction` = 0.90_SZ
- real(sz) `windreduction` = DEFV_WINDREDUCTION
- 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 `ncshuffle` = 0
- integer `ncdeflate` = 0
- integer `ncplevel` = 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`
- character(19), dimension(:), allocatable `datestimes`

18.5.1 Function/Subroutine Documentation

18.5.1.1 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

in	<i>atmT</i>	Air temperature ($^{\circ}C$)
in	<i>atmP</i>	Atmospheric pressure (<i>mbar</i>)
in	<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](#).

18.5.2 Variable Documentation

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

Definition at line 117 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [utilities::printmodelparams\(\)](#), and [parwind::processasymmetricvortexdata\(\)](#).

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

Definition at line 78 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), and [parwind::gethollandfields\(\)](#).

18.5.2.3 begdate integer pahm_global::begdate = IMISSV

Definition at line 141 of file [global.F90](#).

18.5.2.4 begdatespecified logical pahm_global::begdatespecified = .FALSE.

Definition at line 149 of file [global.F90](#).

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

18.5.2.5 begdatetime character(len=64) pahm_global::begdatetime = BLANK

Definition at line 140 of file [global.F90](#).

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

18.5.2.6 begday integer pahm_global::begday = 0

Definition at line 145 of file [global.F90](#).

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

18.5.2.7 beghour integer pahm_global::beghour = 0

Definition at line 146 of file [global.F90](#).

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

18.5.2.8 begmin integer pahm_global::begmin = 0

Definition at line 147 of file [global.F90](#).

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

18.5.2.9 begmonth integer pahm_global::begmonth = 0

Definition at line 144 of file [global.F90](#).

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

18.5.2.10 begsec integer pahm_global::begsec = 0

Definition at line 148 of file [global.F90](#).

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

18.5.2.11 begsimspecified logical pahm_global::begsimspecified = .FALSE.

Definition at line 166 of file [global.F90](#).

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

18.5.2.12 begsimtime real(sz) pahm_global::begsimtime = RMISSV

Definition at line 164 of file [global.F90](#).

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

18.5.2.13 begtime integer pahm_global::begtime = IMISSV

Definition at line 142 of file [global.F90](#).

18.5.2.14 begyear integer pahm_global::begyear = IMISSV

Definition at line 143 of file [global.F90](#).

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

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

Definition at line 108 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [utilities::printmodelparams\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [parwind::readbesttrackfile\(\)](#), and [parwind::readcsvbesttrackfile\(\)](#).

18.5.2.16 besttrackfilenamespecified logical pahm_global::besttrackfilenamespecified = .FALSE.

Definition at line 106 of file [global.F90](#).

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

18.5.2.17 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::getprogramcmdargs\(\)](#).

18.5.2.18 datestimes character(19), dimension(:), allocatable pahm_global::datestimes

Definition at line 217 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), and [parwind::gethollandfields\(\)](#).

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

Definition at line 189 of file [global.F90](#).

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

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

Definition at line 189 of file [global.F90](#).

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

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

Definition at line 189 of file [global.F90](#).

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

18.5.2.22 defv_atmpress real(sz), parameter pahm_global::defv_atmpress = 1013.25_sz

Definition at line 43 of file [global.F90](#).

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

18.5.2.23 defv_gravity real(sz), parameter pahm_global::defv_gravity = 9.80665_sz

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

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

18.5.2.24 defv_rhoair real(sz), parameter pahm_global::defv_rhoair = 1.1478_sz

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

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

18.5.2.25 defv_rhowater real(sz), parameter pahm_global::defv_rhowater = 1000.0000

Definition at line 49 of file [global.F90](#).

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

18.5.2.26 defv_windreduction real(sz), parameter pahm_global::defv_windreduction = 0.90_sz

Definition at line 120 of file [global.F90](#).

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

18.5.2.27 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::cpptogeо::cpptogeо_1d\(\)](#), [utilities::cpptogeо::cpptogeо_scalar\(\)](#), [utilities::geotocpp::geotocpp_1d\(\)](#), [utilities::geotocpp::geotocpp_scalar\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_vortex::newvortex\(\)](#), [pahm_vortex::newvortexfull\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [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\(\)](#).

18.5.2.28 enddate integer pahm_global::enddate = IMISSV

Definition at line 153 of file [global.F90](#).

18.5.2.29 enddatespecified logical pahm_global::enddatespecified = .FALSE.

Definition at line 161 of file [global.F90](#).

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

18.5.2.30 enddatetime character(len=64) pahm_global::enddatetime = BLANK

Definition at line 152 of file [global.F90](#).

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

18.5.2.31 endday integer pahm_global::endday = 0

Definition at line 157 of file [global.F90](#).

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

18.5.2.32 endhour integer pahm_global::endhour = 0

Definition at line 158 of file [global.F90](#).

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

18.5.2.33 endmin integer pahm_global::endmin = 0

Definition at line 159 of file [global.F90](#).

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

18.5.2.34 endmonth integer pahm_global::endmonth = 0

Definition at line 156 of file [global.F90](#).

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

18.5.2.35 endsec integer pahm_global::endsec = 0

Definition at line 160 of file [global.F90](#).

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

18.5.2.36 endsimspecified logical pahm_global::endsimspecified = .FALSE.

Definition at line 167 of file [global.F90](#).

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

18.5.2.37 endsimtime real(sz) pahm_global::endsimtime = RMISSV

Definition at line 165 of file [global.F90](#).

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

18.5.2.38 endtime integer pahm_global::endtime = IMISSV

Definition at line 154 of file [global.F90](#).

18.5.2.39 endyear integer pahm_global::endyear = IMISSV

Definition at line 155 of file [global.F90](#).

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

18.5.2.40 gravity real(sz) pahm_global::gravity = DEFV_GRAVITY

Definition at line 114 of file [global.F90](#).

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

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

Definition at line 83 of file [global.F90](#).

Referenced by [pahm_vortex::calcrmaxesfull\(\)](#), [parwind::getgahmfields\(\)](#), [pahm_vortex::newvortex\(\)](#), [pahm_vortex::newvortexfull\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [parwind::processhollanddata\(\)](#), [pahm_vortex::uvp\(\)](#), [pahm_vortex::uvpr\(\)](#), [pahm_vortex::vhncori\(\)](#), [pahm_vortex::vhwithcori\(\)](#), and [pahm_vortex::vhwithcorifull\(\)](#).

18.5.2.42 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\(\)](#).

18.5.2.43 lun_btrk integer, parameter pahm_global::lun_btrk = 22

Definition at line 30 of file [global.F90](#).

Referenced by [csv_module::read_csv_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [parwind::writeasymmetricvortexdata\(\)](#), and [parwind::writebesttrackdata\(\)](#).

18.5.2.44 lun_btrk1 integer, parameter pahm_global::lun_btrk1 = 23

Definition at line 31 of file [global.F90](#).

Referenced by [parwind::readbesttrackfile\(\)](#), [parwind::writeasymmetricvortexdata\(\)](#), and [parwind::writebesttrackdata\(\)](#).

18.5.2.45 lun_ctrl integer, parameter pahm_global::lun_ctrl = 10

Definition at line 26 of file [global.F90](#).

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

18.5.2.46 lun_inp integer, parameter pahm_global::lun_inp = 14

Definition at line 27 of file [global.F90](#).

Referenced by [pahm_mesh::readmeshasciifort14\(\)](#).

18.5.2.47 lun_inp1 integer, parameter pahm_global::lun_inp1 = 15

Definition at line 28 of file [global.F90](#).

18.5.2.48 lun_log integer, parameter pahm_global::lun_log = 35

Definition at line 29 of file [global.F90](#).

Referenced by [pahm_messages::closelogfile\(\)](#), [pahm_messages::logmessage::logmessage_1\(\)](#), [pahm_messages::logmessage::logmess](#) and [pahm_messages::openlogfile\(\)](#).

18.5.2.49 lun_out integer, parameter pahm_global::lun_out = 25

Definition at line 32 of file [global.F90](#).

18.5.2.50 lun_out1 integer, parameter pahm_global::lun_out1 = 26

Definition at line 33 of file [global.F90](#).

18.5.2.51 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\(\)](#).

18.5.2.52 m2nm real(sz), parameter pahm_global::m2nm = 1.0_sz / NM2M

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

Referenced by [parwind::getgahmfields\(\)](#), and [pahm_vortex::uvp\(\)](#).

18.5.2.53 mb2kpa real(sz), parameter pahm_global::mb2kpa = 0.1_sz

Definition at line 87 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), and [parwind::gethollandfields\(\)](#).

18.5.2.54 mb2pa real(sz), parameter pahm_global::mb2pa = 100.0_sz

Definition at line 86 of file [global.F90](#).

Referenced by [pahm_vortex::calcrmaxesfull\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_vortex::newvortex\(\)](#), [pahm_vortex::newvortexfull\(\)](#), [pahm_vortex::uvp\(\)](#), and [pahm_vortex::uvpr\(\)](#).

18.5.2.55 mdbegsimtime real(sz) pahm_global::mdbegsimtime = RMISSV

Definition at line 177 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.56 mdendsimtime real(sz) pahm_global::mdendsimtime = RMISSV

Definition at line 178 of file [global.F90](#).

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

18.5.2.57 mdoutdt real(sz) pahm_global::mdoutdt = RMISSV

Definition at line 176 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.58 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\(\)](#).

18.5.2.59 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\(\)](#).

18.5.2.60 meshfilenamespecified logical pahm_global::meshfilenamespecified = .FALSE.

Definition at line 101 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), and [pahm_mesh::readmesh\(\)](#).

18.5.2.61 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\(\)](#).

18.5.2.62 modeltype integer pahm_global::modeltype = IMISSV

Definition at line 197 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_drivermod::pahm_run\(\)](#), and [utilities::printmodelparams\(\)](#).

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

Definition at line 84 of file [global.F90](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#), [pahm_vortex::vhnocori\(\)](#), [pahm_vortex::vhwithcori\(\)](#), and [pahm_vortex::vhwithcorifull\(\)](#).

```
18.5.2.64 nbtrfiles integer pahm_global::nbtrfiles = IMISSV
```

Definition at line 107 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [utilities::printmodelparams\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [parwind::processhollanddata\(\)](#), [parwind::readbesttrackfile\(\)](#), and [parwind::readcsvbesttrackfile\(\)](#).

```
18.5.2.65 ncdeflate integer pahm_global::ncdeflate = 0
```

Definition at line 183 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

```
18.5.2.66 ncdlevel integer pahm_global::ncdlevel = 0
```

Definition at line 184 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

```
18.5.2.67 ncshuffle integer pahm_global::ncshuffle = 0
```

Definition at line 182 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

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

Definition at line 193 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.69 ncvarnam_wndx character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAM_WNDX

Definition at line 193 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.70 ncvarnam_wndy character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAM_WNDY

Definition at line 193 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

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

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

Referenced by [pahm_vortex::calcmaxesfull\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [parwind::processhollanddata\(\)](#), [pahm_vortex::uvp\(\)](#), [pahm_vortex::uvpr\(\)](#), [pahm_vortex::vhwithcori\(\)](#), and [pahm_vortex::vhwithcorifull\(\)](#).

18.5.2.72 noutdt integer pahm_global::noutdt = IMISSV

Definition at line 175 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_drivermod::pahm_init\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.73 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::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_vortex::newvortex\(\)](#), [pahm_vortex::newvortexfull\(\)](#), and [pahm_vortex::setvortex\(\)](#).

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

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

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_vortex::uvp\(\)](#), and [pahm_vortex::uvpr\(\)](#).

18.5.2.75 outdt real(sz) pahm_global::outdt = RMISSV

Definition at line 174 of file [global.F90](#).

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

18.5.2.76 outfilename character(len=fnamelen) pahm_global::outfilename = BLANK

Definition at line 181 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_drivermod::pahm_run\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.77 outfilenamespecified logical pahm_global::outfilenamespecified = .FALSE.

Definition at line 180 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), and [pahm_drivermod::pahm_run\(\)](#).

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

Definition at line 75 of file [global.F90](#).

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

Definition at line 77 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [utilities::sphericalfracpoint\(\)](#), and [pahm_vortex::uvp\(\)](#).

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

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

Referenced by [utilities::cpptogeо::cpptogeо_1d\(\)](#), [utilities::cpptogeо::cpptogeо_scalar\(\)](#), [utilities::geotocpp::geotocpp_1d\(\)](#), [utilities::geotocpp::geotocpp_scalar\(\)](#), [parwind::getgahmfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::sphericaldistance::sphericaldistance_2d\(\)](#), [utilities::sphericaldistance::sphericaldistance_scalar\(\)](#), [utilities::sphericaldistanceharv\(\)](#), [utilities::sphericalfracpoint\(\)](#), and [pahm_vortex::uvp\(\)](#).

18.5.2.81 refdate integer pahm_global::refdate = IMISSV

Definition at line 129 of file [global.F90](#).

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

18.5.2.82 refdatespecified logical pahm_global::refdatespecified = .FALSE.

Definition at line 137 of file [global.F90](#).

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

18.5.2.83 refdatetime character(len=64) pahm_global::refdatetime = BLANK

Definition at line 128 of file [global.F90](#).

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

18.5.2.84 refday integer pahm_global::refday = 0

Definition at line 133 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.85 refhour integer pahm_global::refhour = 0

Definition at line 134 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.86 refmin integer pahm_global::refmin = 0

Definition at line 135 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.87 refmonth integer pahm_global::refmonth = 0

Definition at line 132 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.88 refsec integer pahm_global::refsec = 0

Definition at line 136 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.89 reftime integer pahm_global::reftime = IMISSV

Definition at line 130 of file [global.F90](#).

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

18.5.2.90 refyear integer pahm_global::refyear = IMISSV

Definition at line 131 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

18.5.2.91 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\(\)](#).

18.5.2.92 rhowater real(sz) pahm_global::rhowater = DEFV_RHOWATER

Definition at line 115 of file [global.F90](#).

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

18.5.2.93 tenZone real(sz), parameter pahm_global::tenZone = 1.0_sz / 0.8928_sz

Definition at line 73 of file [global.F90](#).

18.5.2.94 times real(sz), dimension(:), allocatable pahm_global::times

Definition at line 216 of file [global.F90](#).

Referenced by [pahm_vortex::calcintensitychange\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdf\(\)](#), [pahm_vortex::uvtrans\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

18.5.2.95 title character(len=512) pahm_global::title = BLANK

Definition at line 112 of file [global.F90](#).

Referenced by [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::printmodelparams\(\)](#).

18.5.2.96 unittime character(len=1) pahm_global::unittime = 'S'

Definition at line 169 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), [utilities::printmodelparams\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

18.5.2.97 windreduction real(sz) pahm_global::windreduction = DEFV_WINDREDUCTION

Definition at line 121 of file [global.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [parwind::gethollandfields\(\)](#), [utilities::printmodelparams\(\)](#), [parwind::processasymmetricvort\(\)](#), [pahm_vortex::uvp\(\)](#), and [pahm_vortex::uvpr\(\)](#).

18.5.2.98 wpress real(sz), dimension(:), allocatable pahm_global::wpress

Definition at line 215 of file [global.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

18.5.2.99 writeparams logical pahm_global::writeparams = .FALSE.

Definition at line 204 of file [global.F90](#).

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

18.5.2.100 wvelx real(sz), dimension(:), allocatable pahm_global::wvelx

Definition at line 215 of file [global.F90](#).

Referenced by [parwind::getgahmfIELDS\(\)](#), [parwind::gethollandFIELDS\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

18.5.2.101 wvely real(sz), dimension(:), allocatable pahm_global::wvely

Definition at line 215 of file [global.F90](#).

Referenced by [parwind::getgahmfIELDS\(\)](#), [parwind::gethollandFIELDS\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

18.6 pahm_mesh Module Reference

Functions/Subroutines

- subroutine [readmesh](#) ()

Reads an input mesh file for the specified supported model type.
- subroutine [readmeshascifort14](#) ()

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.

18.6.1 Function/Subroutine Documentation

18.6.1.1 `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:



18.6.1.2 `readmesh()` subroutine pahm_mesh::readmesh

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

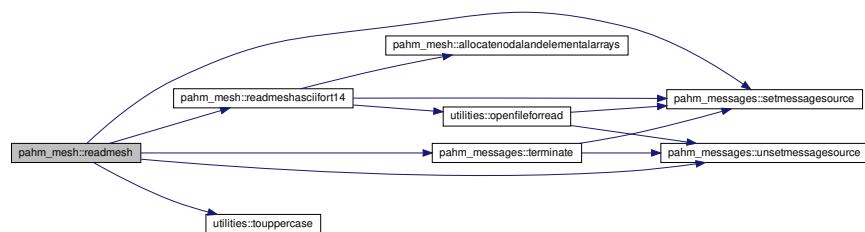
Read the mesh file for the specified model type (meshFileType) and in ASCII or NetCDF format (if applicable).

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

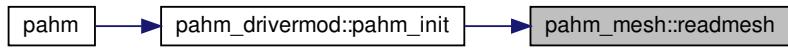
References [pahm_messages::error](#), [pahm_global::meshfileform](#), [pahm_global::meshfilename](#), [pahm_global::meshfilenamespecified](#), [pahm_global::meshfiletype](#), [readmeshasciifort14\(\)](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), [pahm_messages::terminate\(\)](#),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



18.6.1.3 readmeshasciifort14() subroutine pahm_mesh::readmeshasciifort14

Reads the ADCIRC fort.14 mesh file.

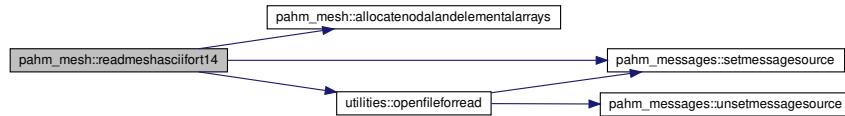
Reads the ADCIRC fort.14 mesh file and sets all mesh variables and arrays.

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

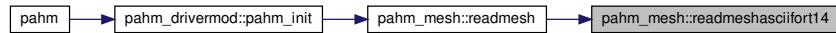
References `agrid`, `allocatenodalandallementalarrays()`, `pahm_messages::info`, `ismeshok`, `pahm_global::lun_inp`, `maxfacenodes`, `pahm_global::meshfilename`, `ne`, `np`, `utilities::openfileforread()`, `pahm_messages::setmessagesource()`, `sfea`, `sfea0`, `slam`, `slam0`, `xclam`, and `ycsfea`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.6.2 Variable Documentation

18.6.2.1 agrid character(len=80) pahm_mesh::agrid

Definition at line 32 of file [mesh.F90](#).

Referenced by [pahm_ncdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.2 dp real(sz), dimension(:), allocatable pahm_mesh::dp

Definition at line 36 of file [mesh.F90](#).

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

18.6.2.3 ics integer pahm_mesh::ics

Definition at line 35 of file [mesh.F90](#).

18.6.2.4 ismeshok logical pahm_mesh::ismeshok = .FALSE.

Definition at line 51 of file [mesh.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.5 maxfacenodes integer, parameter pahm_mesh::maxfacenodes = 5

Definition at line 48 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.6 ne integer pahm_mesh::ne = IMISSV

Definition at line 34 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), [pahm_ncdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.7 nfn integer, dimension(:), allocatable pahm_mesh::nfn

Definition at line 37 of file [mesh.F90](#).

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

18.6.2.8 nm integer, dimension(:, :), allocatable pahm_mesh::nm

Definition at line 38 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), and [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#).

18.6.2.9 np integer pahm_mesh::np = IMISSV

Definition at line 33 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#) and [readmeshasciifort14\(\)](#).

18.6.2.10 sfea real(sz), dimension(:), allocatable pahm_mesh::sfea

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

Referenced by [allocatenodalandelementalarrays\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#) and [readmeshasciifort14\(\)](#).

18.6.2.11 sfea0 real(sz) pahm_mesh::sfea0 = RMISSV

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

Referenced by [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.12 slam real(sz), dimension(:), allocatable pahm_mesh::slam

Definition at line 39 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#) and [readmeshasciifort14\(\)](#).

18.6.2.13 slam0 real(sz) pahm_mesh::slam0 = RMISSV

Definition at line 44 of file [mesh.F90](#).

Referenced by [pahm_ncdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.14 xcslam real(sz), dimension(:), allocatable pahm_mesh::xcslam

Definition at line 41 of file [mesh.F90](#).

Referenced by [allocatenodalandelementalarrays\(\)](#), [pahm_ncdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.6.2.15 ycsfea real(sz), dimension(:), allocatable pahm_mesh::ycsfea

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

Referenced by [allocatenodalandelementalarrays\(\)](#), [pahm_ncdfio::initadcircnetcdfoutfile\(\)](#), and [readmeshasciifort14\(\)](#).

18.7 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) `loglevelname`
- character(len=50), dimension(100) `messagesources`
- character(len=1024) `scratchmessage`
- character(len=1024) `scratchformat`
- integer `sourcenumber`
- logical `logfileopened` = .FALSE.
- logical `loginitcalled` = .FALSE.

18.7.1 Function/Subroutine Documentation

18.7.1.1 `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.

Parameters

<code>in</code>	<code>message</code>	The message to display
-----------------	----------------------	------------------------

Definition at line 318 of file `messages.F90`.

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

Definition at line 330 of file [messages.F90](#).

18.7.1.3 closelogfile() subroutine pahm_messages::closelogfile

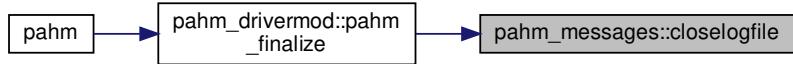
Closes an opened log file.

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:



18.7.1.4 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](#), [loglevelname](#)s, [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:



18.7.1.5 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.

Parameters

in	message	The message to display
----	---------	------------------------

Definition at line 251 of file [messages.F90](#).

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

Definition at line 275 of file [messages.F90](#).

18.7.1.7 openlogfile() subroutine pahm_messages::openlogfile

Opens the log file for writing.

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

References [error](#), [pahm_global::logfilename](#), [logfileopened](#), [pahm_global::lun_log](#), and [scratchmessage](#).

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

Here is the caller graph for this function:



18.7.1.8 programhelp() subroutine pahm_messages::programhelp

Prints on the screen the help system of the program.

Definition at line 439 of file [messages.F90](#).

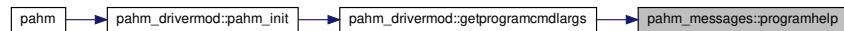
References [pahm_global::lun_screen](#), and [programversion\(\)](#).

Referenced by [pahm_drivermod::getprogramcmdlargs\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**18.7.1.9 programversion()** subroutine pahm_messages::programversion

Prints on the screen the versioning information of the program.

Definition at line 409 of file [messages.F90](#).

References [pahm_global::lun_screen](#).

Referenced by [pahm_drivermod::getprogramcmdlargs\(\)](#), and [programhelp\(\)](#).

Here is the caller graph for this function:

**18.7.1.10 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.

Parameters

in	<i>message</i>	The message to display
----	----------------	------------------------

Definition at line 180 of file [messages.F90](#).

18.7.1.11 screenmessage_2() subroutine pahm_messages::screenmessage_2 (

```
    integer, intent(in) level,
    character(len=*), intent(in) message )
```

Definition at line 204 of file [messages.F90](#).

18.7.1.12 setmessagesource() subroutine pahm_messages::setmessagesource (

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

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

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

Parameters

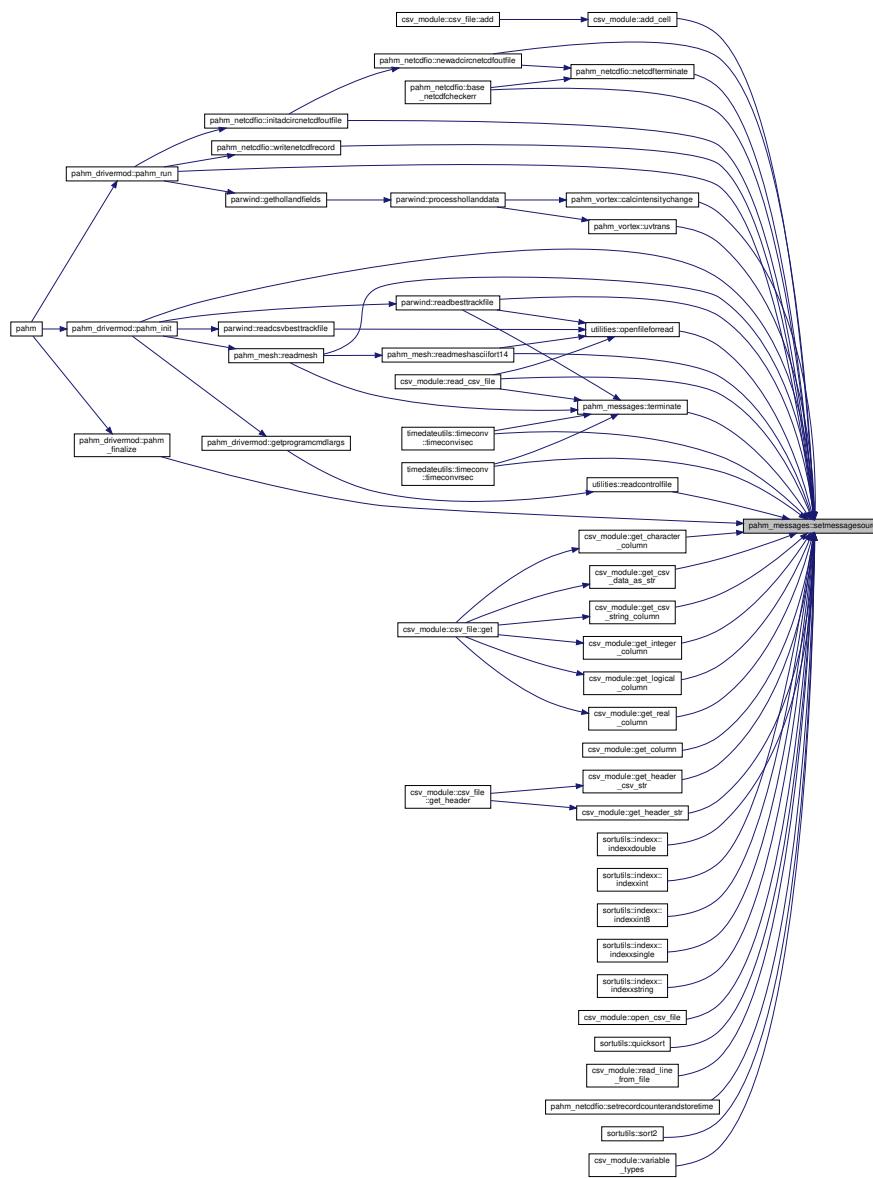
in	<i>source</i>	The name of the calling procedure
----	---------------	-----------------------------------

Definition at line 361 of file [messages.F90](#).

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

Referenced by [csv_module::add_cell\(\)](#), [pahm_netcdffio::base_ncdfcheckerr\(\)](#), [pahm_vortex::calcintensitychange\(\)](#), [csv_module::get_character_column\(\)](#), [csv_module::get_column\(\)](#), [csv_module::get_csv_data_as_str\(\)](#), [csv_module::get_csv_string_column\(\)](#), [csv_module::get_header_csv_str\(\)](#), [csv_module::get_header_str\(\)](#), [csv_module::get_integer_column\(\)](#), [csv_module::get_logical_column\(\)](#), [csv_module::get_real_column\(\)](#), [sortutils::indexx::indexxdouble\(\)](#), [sortutils::indexx::indexxitint\(\)](#), [sortutils::indexx::indexxit8\(\)](#), [sortutils::indexx::indexxsingle\(\)](#), [sortutils::indexx::indexxstring\(\)](#), [pahm_netcdffio::initadcircnetcdfoutfile\(\)](#), [pahm_netcdffio::netcdfterminate\(\)](#), [pahm_netcdffio::newadcircnetcdfoutfile\(\)](#), [csv_module::open_csv_file\(\)](#), [utilities::openfileforread\(\)](#), [pahm_drivermod::pahm_finalize\(\)](#), [pahm_drivermod::pahm_init\(\)](#), [pahm_drivermod::pahm_run\(\)](#), [sortutils::quicksort\(\)](#), [csv_module::read_csv_file\(\)](#), [csv_module::read_line_from_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [utilities::readcontrolfile\(\)](#), [pahm_mesh::readmesh\(\)](#), [pahm_mesh::readmeshasciifort14\(\)](#), [pahm_netcdffio::setrecordcounterandstoretime\(\)](#), [sortutils::sort2\(\)](#), [terminate\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), [timedateutils::timeconv::timeconvrsec\(\)](#), [pahm_vortex::uvtrans\(\)](#), [csv_module::variable_types\(\)](#), and [pahm_netcdffio::writenetcdfrecord\(\)](#).

Here is the caller graph for this function:



18.7.1.13 terminate() subroutine pahm messages:::terminate

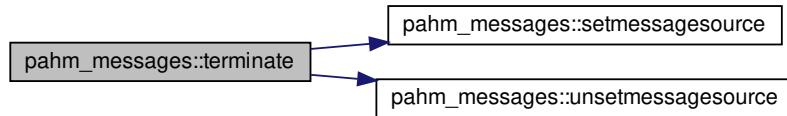
Terminates the calling program when a fatal error is encountered.

Definition at line 464 of file `messages.F90`.

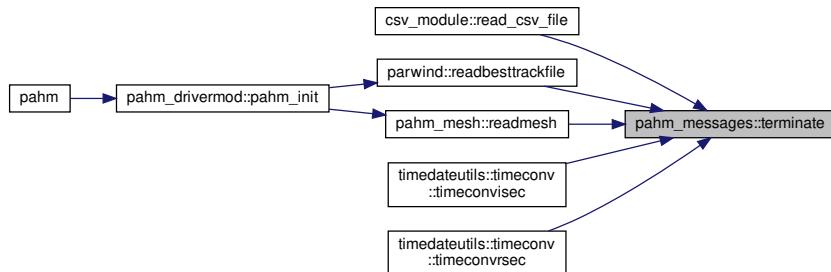
References [error](#), [setmessagesource\(\)](#), and [unsetmessagesource\(\)](#).

Referenced by `csv_module::read_csv_file()`, `parwind::readbesttrackfile()`, `pahm_mesh::readmesh()`, `timedateutils::timeconv::timeconvise()` and `timedateutils::timeconv::timeconvrsec()`.

Here is the call graph for this function:



Here is the caller graph for this function:



18.7.1.14 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.

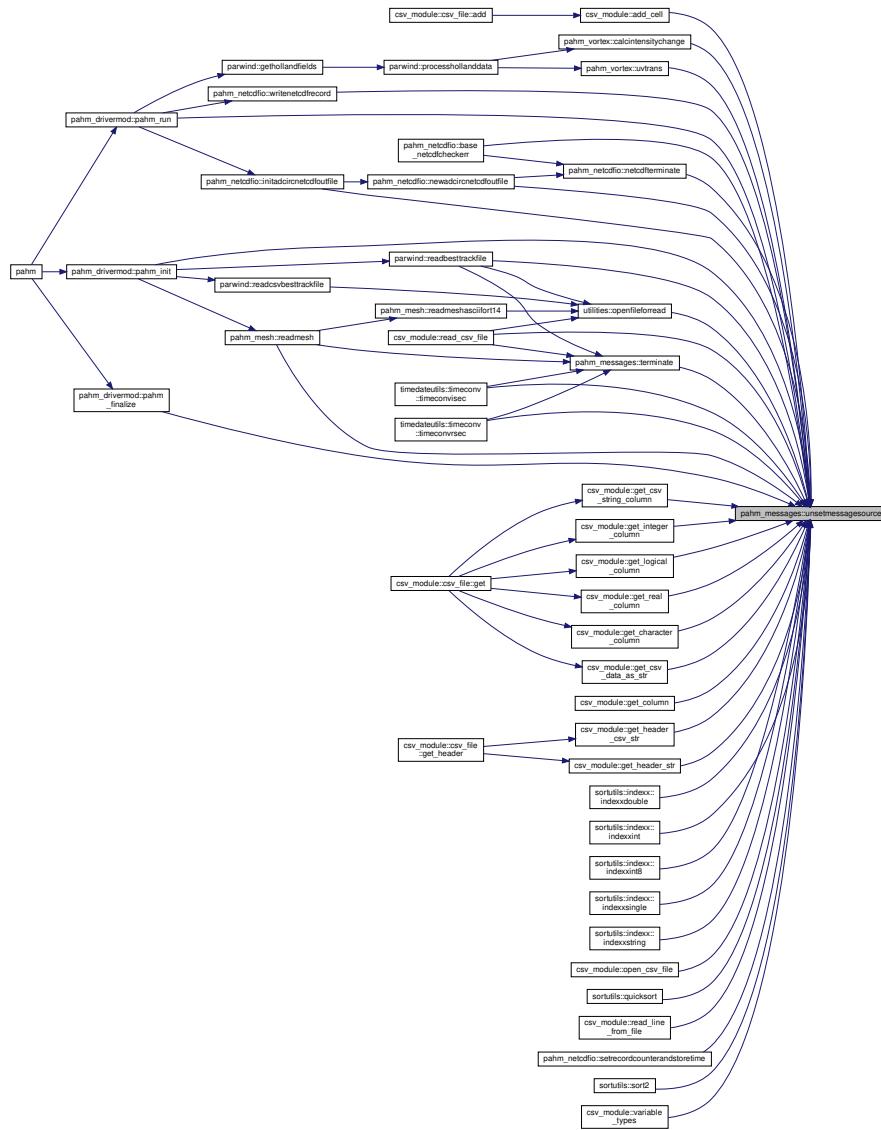
Definition at line 388 of file `messages.F90`.

References `sourcenumber`.

Referenced by `csv_module::add_cell()`, `pahm_netcdfio::base_ncdfcheckerr()`, `pahm_vortex::calcintensitychange()`, `csv_module::get_character_column()`, `csv_module::get_column()`, `csv_module::get_csv_data_as_str()`, `csv_module::get_csv_string_column()`, `csv_module::get_header_csv_str()`, `csv_module::get_header_str()`, `csv_module::get_integer_column()`, `csv_module::get_logical_column()`, `csv_module::get_real_column()`, `sortutils::indexx::indexxdouble()`, `sortutils::indexx::indexxxint()`, `sortutils::indexx::indexxxint8()`, `sortutils::indexx::indexxsingle()`, `sortutils::indexx::indexxstring()`, `pahm_netcdfio::initadcircnetcdfoutfile()`, `pahm_netcdfio::ncdfterminate()`, `pahm_netcdfio::newadcircnetcdfoutfile()`, `csv_module::open_csv_file()`, `utilities::openfileforread()`, `pahm_drivermod::pahm_finalize()`, `pahm_drivermod::pahm_init()`, `pahm_drivermod::pahm_run()`, `sortutils::quicksort()`, `csv_module::read_csv_file()`,

`csv_module::read_line_from_file()`, `parwind::readbesttrackfile()`, `pahm_mesh::readmesh()`, `pahm_ncdfio::setrecordcounterandstoretime()`, `sortutils::sort2()`, `terminate()`, `timedateutils::timeconv::timeconvisec()`, `timedateutils::timeconv::timeconvrsec()`, `pahm_vortex::uvtrans()`, `csv_module::variable_types()`, and `pahm_ncdfio::writenetcdfrecord()`.

Here is the caller graph for this function:



18.7.2 Variable Documentation

18.7.2.1 debug integer, parameter pahm_messages::debug = -1

Definition at line 30 of file [messages.F90](#).

18.7.2.2 echo integer, parameter pahm_messages::echo = 0

Definition at line 31 of file [messages.F90](#).

18.7.2.3 error integer, parameter pahm_messages::error = 3

Definition at line 34 of file [messages.F90](#).

Referenced by [csv_module::add_cell\(\)](#), [pahm_netcdfio::base_ncdfcheckerr\(\)](#), [pahm_vortex::calcintensitychange\(\)](#), [csv_module::get_character_column\(\)](#), [csv_module::get_column\(\)](#), [csv_module::get_csv_data_as_str\(\)](#), [csv_module::get_csv_string_column\(\)](#), [csv_module::get_header_csv_str\(\)](#), [csv_module::get_header_str\(\)](#), [csv_module::get_integer_column\(\)](#), [csv_module::get_logical_column\(\)](#), [csv_module::get_real_column\(\)](#), [sortutils::indexx::indexxdouble\(\)](#), [sortutils::indexx::indexxit\(\)](#), [sortutils::indexx::indexxit8\(\)](#), [sortutils::indexx::indexxsingle\(\)](#), [sortutils::indexx::indexxstring\(\)](#), [pahm_netcdfio::newadcircnetcdfoutfile\(\)](#), [csv_module::open_csv_file\(\)](#), [utilities::openfileforread\(\)](#), [openlogfile\(\)](#), [pahm_drivermod::pahm_run\(\)](#), [sortutils::quicksort\(\)](#), [csv_module::read_csv_file\(\)](#), [csv_module::read_line_from_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [pahm_mesh::readmesh\(\)](#), [sortutils::sort2\(\)](#), [terminate\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), [timedateutils::timeconv::timeconvrsec\(\)](#), [pahm_vortex::uvtrans\(\)](#), and [csv_module::variable_types\(\)](#).

18.7.2.4 info integer, parameter pahm_messages::info = 1

Definition at line 32 of file [messages.F90](#).

Referenced by [pahm_netcdfio::base_ncdfcheckerr\(\)](#), [pahm_netcdfio::netcdfterminate\(\)](#), [pahm_netcdfio::newadcircnetcdfoutfile\(\)](#), [utilities::openfileforread\(\)](#), [csv_module::read_csv_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [pahm_mesh::readmeshasciifort14\(\)](#), and [pahm_netcdfio::setrecordcounterandstoretime\(\)](#).

18.7.2.5 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\(\)](#).

18.7.2.6 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\(\)](#).

18.7.2.7 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\(\)](#).

18.7.2.8 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\(\)](#).

18.7.2.9 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\(\)](#).

18.7.2.10 scratchformat character(len=1024) pahm_messages::scratchformat

Definition at line 39 of file [messages.F90](#).

Referenced by [pahm_netcdfio::setrecordcounterandstoretime\(\)](#).

18.7.2.11 scratchmessage character(len=1024) pahm_messages::scratchmessage

Definition at line 38 of file [messages.F90](#).

Referenced by [csv_module::add_cell\(\)](#), [pahm_vortex::calcintensitychange\(\)](#), [csv_module::get_character_column\(\)](#), [csv_module::get_column\(\)](#), [csv_module::get_csv_data_as_str\(\)](#), [csv_module::get_csv_string_column\(\)](#), [csv_module::get_header_csv_st](#), [csv_module::get_header_str\(\)](#), [csv_module::get_integer_column\(\)](#), [csv_module::get_logical_column\(\)](#), [csv_module::get_real_column\(\)](#), [sortutils::indexx::indexxdouble\(\)](#), [sortutils::indexx::indexxint\(\)](#), [sortutils::indexx::indexxint8\(\)](#), [sortutils::indexx::indexxsingle\(\)](#), [sortutils::indexx::indexxstring\(\)](#), [pahm_netcdfio::newadcircnetcdfoutfile\(\)](#), [csv_module::open_csv_file\(\)](#), [utilities::openfileforread\(\)](#), [openlogfile\(\)](#), [pahm_drivermod::pahm_run\(\)](#), [sortutils::quicksort\(\)](#), [csv_module::read_csv_file\(\)](#), [csv_module::read_line_from_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [pahm_mesh::readmesh\(\)](#), [pahm_netcdfio::setrecordcounterandstoretime\(\)](#), [sortutils::sort2\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), [timedateutils::timeconv::timeconvrsec\(\)](#), [pahm_vortex::uvtrans\(\)](#), and [csv_module::variable_types\(\)](#).

18.7.2.12 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\(\)](#).

18.7.2.13 warning integer, parameter pahm_messages::warning = 2

Definition at line 33 of file [messages.F90](#).

18.8 pahm_netcdfio Module Reference

Data Types

- type [adcirccoorddata_t](#)
- type [adcircvardata3d_t](#)
- type [adcircvardata_t](#)
- type [filedata_t](#)
- type [timedata_t](#)

Functions/Subroutines

- subroutine [initadcircnetcdfoutfile](#) (adcircOutFile)
Initializes a new NetCDF data file and puts it in define mode.
- subroutine [newadcircnetcdfoutfile](#) (nclD, 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.
- subroutine [netcdfterminate](#) ()
Terminates the program on NetCDF error.
- subroutine [writenetcdfrecord](#) (adcircOutFile, timeLoc)
Writes data to the NetCDF file.
- subroutine [setrecordcounterandstoretime](#) (nclD, f, t)
Sets the record counter.

Variables

- integer, private `ncformat`
- integer, parameter, private `nc4form` = IOR(NF90_NETCDF4, NF90_CLASSIC_MODEL)
- integer, parameter, private `nc3form` = IOR(NF90_CLOBBER, 0)
- integer, private `nodedimid`
- integer, private `vertdimid`
- integer, private `elemdimid`
- integer, private `meshdimid`
- integer, private `meshvarid`
- integer, private `projvarid`
- type(`filedata_t`), save `myfile`
- type(`timedata_t`), save `mytime`
- type(`adcirccoorddata_t`), save, private `crdtime`
- type(`adcirccoorddata_t`), save, private `crdlons`
- type(`adcirccoorddata_t`), save, private `crdlats`
- type(`adcirccoorddata_t`), save, private `crdxcs`
- type(`adcirccoorddata_t`), save, private `crdycs`
- type(`adcircvardata_t`), save, private `datelements`
- type(`adcircvardata_t`), save, private `datatmpres`
- type(`adcircvardata_t`), save, private `datwindx`
- type(`adcircvardata_t`), save, private `datwindy`

18.8.1 Function/Subroutine Documentation

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

Checks the return value from netCDF calls.

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

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

Definition at line 704 of file `netcdfio.F90`.

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

Here is the call graph for this function:



18.8.1.2 initadcircnetcdfoutfile() subroutine pahm_ncdfio::initadcircnetcdfoutfile (
character(len=*), intent(inout) *adcircOutFile*)

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

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

Parameters

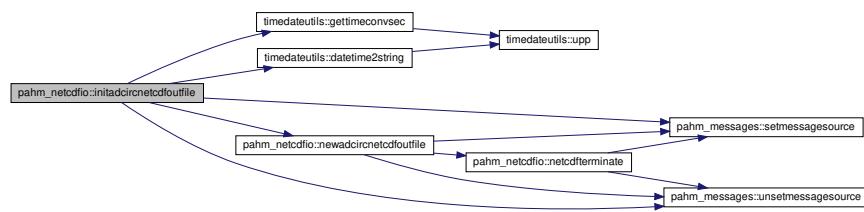
in, out	<i>adcircOutFile</i>	The name of the file to be initialized. The file is first created by calling NewAdcircNetCDFOutFile.
---------	----------------------	--

Definition at line 123 of file [netcdfio.F90](#).

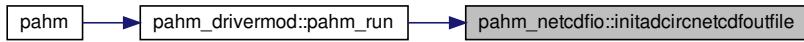
References [pahm_mesh::agrid](#), [crdlats](#), [crdlons](#), [crdtime](#), [crdxcs](#), [crdycs](#), [datatmpres](#), [datelements](#), [timedateutils::datetime2string\(\)](#), [datwindx](#), [datwindy](#), [elemdimid](#), [timedateutils::gettimeconvsec\(\)](#), [pahm_sizes::imissv](#), [meshdimid](#), [meshvarid](#), [myfile](#), [nc4form](#), [pahm_global::ncdeflate](#), [pahm_global::nclevel](#), [pahm_global::ncformat](#), [pahm_global::ncshuffle](#), [pahm_global::ncvarnam_pres](#), [pahm_global::ncvarnam_wndx](#), [pahm_global::ncvarnam_wndy](#), [pahm_mesh::ne](#), [newadcircnetcdfoutfile\(\)](#), [pahm_mesh::nm](#), [nodedimid](#), [pahm_mesh::np](#), [projvarid](#), [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\(\)](#), [vertdimid](#), [pahm_global::wpress](#), [pahm_global::wvelx](#), [pahm_global::wwely](#), [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:



18.8.1.3 netcdfterminate() subroutine pahm_ncdfio::netcdfterminate

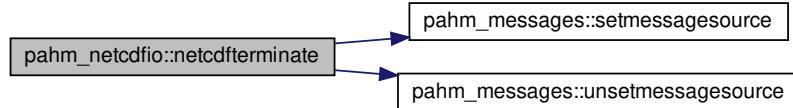
Terminates the program on NetCDF error.

Definition at line 740 of file [netcdfio.F90](#).

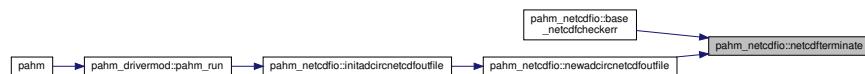
References [pahm_messages::info](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [base_ncdfcheckerr\(\)](#), and [newadcircnetcdfoutfile\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.8.1.4 newadcircnetcdfoutfile() subroutine pahm_ncdfio::newadcircnetcdfoutfile (

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

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

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

out	<i>ncID</i>	The NetCDF ID of the file to be created (output)
in, out	<i>adcircOutFile</i>	The name of the file to be created (input/output)

Definition at line 606 of file [netcdfio.F90](#).

References [pahm_messages::error](#), [pahm_messages::info](#), [nc3form](#), [nc4form](#), [ncformat](#), [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:



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

Sets the record counter.

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.

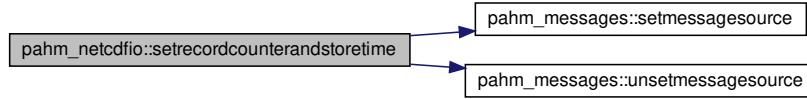
Parameters

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

Definition at line 850 of file [netcdfio.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:



18.8.1.6 writenetcdffrecord() subroutine pahm_ncdfio::writenetcdffrecord (character(len=*) , intent(in) adcircOutFile, integer, intent(in) timeLoc)

Writes data to the NetCDF file.

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

Parameters

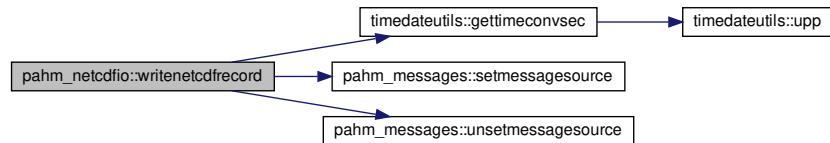
in	<i>adcircOutFile</i>	The name of the NetCDF file
in	<i>timeLoc</i>	The time record to write

Definition at line 775 of file [netcdfio.F90](#).

References [crdtime](#), [datatmpres](#), [datwindx](#), [datwindy](#), [timedateutils::gettimeconvsec\(\)](#), [nodedimid](#), [pahm_messages::setmessagesource\(\)](#), [pahm_global::times](#), [pahm_global::unittime](#), [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:



18.8.2 Variable Documentation

18.8.2.1 crdlats type([adcirccoorddata_t](#)), save, private pahm_netcdfio::crdlats [private]

Definition at line 94 of file [netcdfio.F90](#).

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

18.8.2.2 crdlons type([adcirccoorddata_t](#)), save, private pahm_netcdfio::crdlons [private]

Definition at line 93 of file [netcdfio.F90](#).

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

18.8.2.3 crdtime type([adcirccoorddata_t](#)), save, private pahm_netcdfio::crdtime [private]

Definition at line 92 of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [writenetcdffrecord\(\)](#).

18.8.2.4 crdxcs type([adcirccoorddata_t](#)), save, private pahm_netcdfio::crdxcs [private]

Definition at line 95 of file [netcdfio.F90](#).

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

18.8.2.5 `crdycs` type([adcirccoorddata_t](#)), save, private pahm_netcdfio::crdycs [private]

Definition at line [96](#) of file [netcdfio.F90](#).

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

18.8.2.6 `datatmpres` type([adcircvardata_t](#)), save, private pahm_netcdfio::datatmpres [private]

Definition at line [99](#) of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [writenetcdfreCORD\(\)](#).

18.8.2.7 `datelements` type([adcircvardata_t](#)), save, private pahm_netcdfio::datelements [private]

Definition at line [98](#) of file [netcdfio.F90](#).

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

18.8.2.8 `datwindx` type([adcircvardata_t](#)), save, private pahm_netcdfio::datwindx [private]

Definition at line [100](#) of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [writenetcdfreCORD\(\)](#).

18.8.2.9 `datwindy` type([adcircvardata_t](#)), save, private pahm_netcdfio::datwindy [private]

Definition at line [101](#) of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [writenetcdfreCORD\(\)](#).

18.8.2.10 `elemdimid` integer, private pahm_netcdfio::elemdimid [private]

Definition at line [36](#) of file [netcdfio.F90](#).

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

18.8.2.11 meshdimid integer, private pahm_ncdfio::meshdimid [private]

Definition at line 36 of file [netcdfio.F90](#).

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

18.8.2.12 meshvarid integer, private pahm_ncdfio::meshvarid [private]

Definition at line 37 of file [netcdfio.F90](#).

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

18.8.2.13 myfile type([filedata_t](#)), save pahm_ncdfio::myfile [private]

Definition at line 89 of file [netcdfio.F90](#).

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

18.8.2.14 mytime type([timedata_t](#)), save pahm_ncdfio::mytime

Definition at line 90 of file [netcdfio.F90](#).

18.8.2.15 nc3form integer, parameter, private pahm_ncdfio::nc3form = IOR(NF90_CLOBBER, 0) [private]

Definition at line 34 of file [netcdfio.F90](#).

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

18.8.2.16 nc4form integer, parameter, private pahm_ncdfio::nc4form = IOR(NF90_NETCDF4, NF90_↔ CLASSIC_MODEL) [private]

Definition at line 33 of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [newadcircnetcdfoutfile\(\)](#).

18.8.2.17 ncformat integer, private pahm_ncdfio::ncformat [private]

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

Referenced by [initadcircnetcdfoutfile\(\)](#), and [newadcircnetcdfoutfile\(\)](#).

18.8.2.18 nodedimid integer, private pahm_ncdfio::nodedimid [private]

Definition at line 36 of file [netcdfio.F90](#).

Referenced by [initadcircnetcdfoutfile\(\)](#), and [writenetcdfrecord\(\)](#).

18.8.2.19 projvarid integer, private pahm_ncdfio::projvarid [private]

Definition at line 37 of file [netcdfio.F90](#).

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

18.8.2.20 vertdimid integer, private pahm_ncdfio::vertdimid [private]

Definition at line 36 of file [netcdfio.F90](#).

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

18.9 pahm_sizes Module Reference

Data Types

- interface [comparereals](#)
- interface [fixnearwholereal](#)

Functions/Subroutines

- integer function [comparedoublereals](#) (rVal1, rVal2, eps)
Compares two double precision numbers.
- integer function [comparesinglereals](#) (rVal1, rVal2, eps)
Compares two single precision numbers.
- real([hp](#)) function [fixnearwholedoublereal](#) (rVal, eps)
Rounds a double precision real number to its nearest whole number.
- real([sp](#)) function [fixnearwholesinglereal](#) (rVal, eps)
Rounds a single precision real number to its nearest whole number.

Variables

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

18.9.1 Function/Subroutine Documentation

```
18.9.1.1 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

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

```
18.9.1.2 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

in	<code>rVal1</code>	The first value (single precision number) in the comparison
in	<code>rVal2</code>	The second value (single precision number) in the comparison
in	<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 168 of file [sizes.F90](#).

```
18.9.1.3 fixnearwholedoubblereal() real(hp) function pahm_sizes::fixnearwholedoubblereal (
    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

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

Returns

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

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

Definition at line 235 of file [sizes.F90](#).

```
18.9.1.4 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

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

Returns

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

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

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

18.9.2 Variable Documentation

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

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

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

18.9.2.2 fnamelen integer, parameter pahm_sizes::fnamelen = 1024

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

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

Definition at line 35 of file [sizes.F90](#).

Referenced by [timedateutils::gregtojulday::gregtojuldayisec\(\)](#).

18.9.2.4 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\(\)](#).

18.9.2.5 int1 integer, parameter pahm_sizes::int1 = SELECTED_INT_KIND(2)

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

18.9.2.6 int16 integer, parameter pahm_sizes::int16 = SELECTED_INT_KIND(38)

Definition at line 38 of file [sizes.F90](#).

18.9.2.7 int2 integer, parameter pahm_sizes::int2 = SELECTED_INT_KIND(4)

Definition at line 41 of file [sizes.F90](#).

18.9.2.8 int4 integer, parameter pahm_sizes::int4 = SELECTED_INT_KIND(9)

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

18.9.2.9 int8 integer, parameter pahm_sizes::int8 = SELECTED_INT_KIND(18)

Definition at line 39 of file [sizes.F90](#).

18.9.2.10 ip integer, parameter pahm_sizes::ip = INT8

Definition at line 47 of file [sizes.F90](#).

18.9.2.11 llong integer, parameter pahm_sizes::llong = INT16

Definition at line 44 of file [sizes.F90](#).

18.9.2.12 long integer, parameter pahm_sizes::long = INT8

Definition at line 43 of file [sizes.F90](#).

18.9.2.13 nbyte integer, parameter pahm_sizes::nbyte = 8

Definition at line 57 of file [sizes.F90](#).

18.9.2.14 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::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

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

Definition at line 34 of file [sizes.F90](#).

18.9.2.16 sz integer, parameter pahm_sizes::sz = HP

Definition at line 56 of file [sizes.F90](#).

18.9.2.17 wp integer, parameter pahm_sizes::wp = HP

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

Referenced by [csv_module::add_cell\(\)](#), [csv_module::csv_get_value\(\)](#), and [csv_module::get_column\(\)](#).

18.10 pahm_vortex Module Reference

Functions/Subroutines

- subroutine, public [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, public [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, public [uvtranspoint](#) (lat1, lon1, lat2, lon2, time1, time2, u, v)

This subroutine calculates the translational velocity of a moving hurricane.

- subroutine, public [newvortex](#) (pinf, p0, lat, lon, vm)

Creates a new Vortex object.

- subroutine, public [newvortexfull](#) (pinf, p0, lat, lon, vm)

Creates a new Vortex object.

- subroutine, public [setvortex](#) (pinf, p0, lat, lon)

Sets basic parameters for a new Vortex object.

- subroutine, public [setrmaxes](#) (rMaxW)

- subroutine, public [getrmaxes](#) (rMaxW)

- subroutine, public [calcrmaxes](#) ()

Calculates the radius of maximum winds for all storm quadrants.

- subroutine, public [calcrmaxesfull](#) ()

Calculates the radius of maximum winds for all storm quadrants.

- subroutine, public [fitrmaxes](#) ()

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

- subroutine, public [fitrmaxes4](#) ()

- subroutine, public [setvmaxesbl](#) (vMaxW)

- subroutine, public [getvmaxesbl](#) (vMaxW)

- subroutine, public [setusevmaxesbl](#) (u)

- subroutine, public [setshapeparameter](#) (param)

- real(sz) function, public [getshapeparameter](#) ()

- real(sz) function, dimension(4), public [getshapeparameters](#) ()

- real(sz) function, dimension(4), public [getphifactors](#) ()

- subroutine, public [setisotachradii](#) (ir)

- subroutine, public [setisotachwindspeeds](#) (vrQ)

- subroutine, public `setisotachwindspeed` (`sp`)
- subroutine, public `setusequadrantvr` (`u`)
- logical function, public `getusequadrantvr` ()
- real(`sz`) function, public `sinterp` (`angle, dist, opt`)

Spatial Interpolation function based on angle and r.

- real(`sz`) function, public `interp` (`quadVal, quadSel, quadDis`)
- real(`sz`) function, public `rmw` (`angle`)

Calculate the radius of maximum winds.

- subroutine, public `uvp` (`lat, lon, uTrans, vTrans, u, v, p`)

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

- subroutine, public `uvpr` (`iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, uTrans, vTrans, geof, u, v, p`)

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

- real(`sz`) function, public `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, public `getlatestrmax` ()
- real(`sz`) function, public `getlatestangle` ()
- real(`sz`) function `vhwthcorifull` (`testRMax`)

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

- real(`sz`) function `vhwthcori` (`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), public `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), public `bs4`
- real(`sz`), dimension(`npoints`) `vmbl`
- real(`sz`), dimension(`npoints`, 4), public `vmbl4`
- integer, dimension(`npoints`, 4), public `quadflag4`

- real(sz), dimension(npoints, 4), public quadir4
- real(sz), dimension(nquads) vrquadrant
- real(sz), dimension(nquads) radius
- integer quad
- real(sz) latestrmax
- real(sz) latestangle
- logical usequadrantvr
- logical usevmaxesbl

18.10.1 Function/Subroutine Documentation

18.10.1.1 calcintensitychange() subroutine, public 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.

Parameters

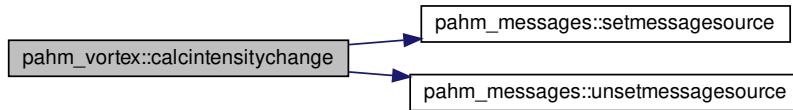
in	<i>var</i>	The input variable (vector)
in	<i>times</i>	Time values (vector) at the center locations
out	<i>calcInt</i>	The calculated intensity change (df/dt)
out	<i>status</i>	Error status (0 means no error)
in	<i>order</i>	The accuracy order required for the calculations (1, 2) order <= 1: first order approximation for finite differences order >= 2: second order approximation for finite differences

Definition at line 108 of file [vortex.F90](#).

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

Referenced by [parwind::processhollanddata\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.10.1.2 calcrmaxes() subroutine, public pahm_vortex::calcrmaxes

Calculates the radius of maximum winds for all storm quadrants.

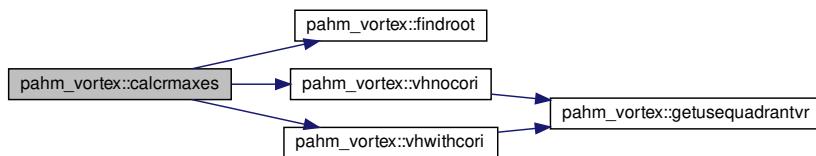
Calculates rMax, the radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity

Definition at line 743 of file [vortex.F90](#).

References [b](#), [bs](#), [findroot\(\)](#), [nquads](#), [quad](#), [radius](#), [rmaxes](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), [vmax](#), and [vmbi](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.10.1.3 `calcrmaxesfull()` subroutine, public pahm_vortex::calcrmaxesfull

Calculates the radius of maximum winds for all storm quadrants.

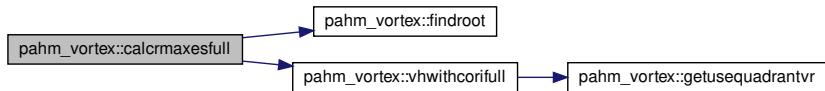
Solves the full gradient wind equation without the assumption of cyclostrophic balance. Calculates rMax, the radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity.

Definition at line 823 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 [vml](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.10.1.4 `fang()` real(sz) function, public pahm_vortex::fang

```

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

Compute a wind angle to parameterize frictional inflow across isobars.

Parameters

in	<i>r</i>	Distance from center of storm
in	<i>rmx</i>	Radius of maximum winds

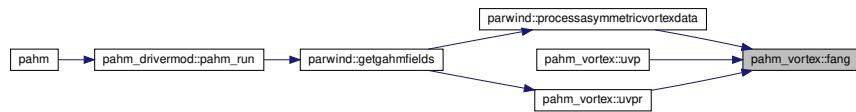
Returns

myValOut Frictional inflow angle (degrees)

Definition at line 1742 of file [vortex.F90](#).

Referenced by `parwind::processasymmetricvortexdata()`, `uvp()`, and `uvpr()`.

Here is the caller graph for this function:



```
18.10.1.5 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*].

Parameters

in	<i>func</i>	Function $f(x)=0$ for which root is sought
in	<i>x1</i>	Left side of the interval
in	<i>x2</i>	Right side of the interval
in	<i>dx</i>	x increment for march
out	<i>a</i>	Left side of interval that brackets the root
out	<i>b</i>	Right side of interval that brackets the root

Returns

myRoot The value of the root

Definition at line 2006 of file `vortex.F90`.

References [b](#).

Referenced by `calcrmaxes()`, and `calcrmaxesfull()`.

Here is the caller graph for this function:



18.10.1.6 fitrmaxes() subroutine, public pahm_vortex::fitrmaxes

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

Generates 2 additional (theta, rMax) points for curve fitting.

Definition at line 956 of file [vortex.F90](#).

References [rmaxes](#).

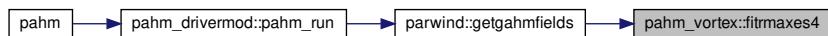
18.10.1.7 fitrmaxes4() subroutine, public pahm_vortex::fitrmaxes4

Definition at line 971 of file [vortex.F90](#).

References [bs4](#), [phis4](#), [quadflag4](#), [quadir4](#), [rmaxes4](#), and [vmb4](#).

Referenced by [parwind::getgahmfields\(\)](#).

Here is the caller graph for this function:

**18.10.1.8 getlatestangle()** real(sz) function, public pahm_vortex::getlatestangle

Definition at line 1838 of file [vortex.F90](#).

References [latestangle](#).

18.10.1.9 getlatestrmax() real(sz) function, public pahm_vortex::getlatestrmax

Definition at line 1825 of file [vortex.F90](#).

References [latestrmax](#).

18.10.1.10 getphifactors() real(sz) function, dimension(4), public pahm_vortex::getphifactors

Definition at line 1103 of file [vortex.F90](#).

References [phis](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:

**18.10.1.11 getrmaxes()** subroutine, public pahm_vortex::getrmaxes (real(sz), dimension(4), intent(out) rMaxW)

Definition at line 715 of file [vortex.F90](#).

References [rmaxes](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:

**18.10.1.12 getshapeparameter()** real(sz) function, public pahm_vortex::getshapeparameter

Definition at line 1067 of file [vortex.F90](#).

References [b](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



18.10.1.13 getshapeparameters() real(sz) function, dimension(4), public pahm_vortex::getshapeparameters

Definition at line 1082 of file [vortex.F90](#).

References [bs](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:

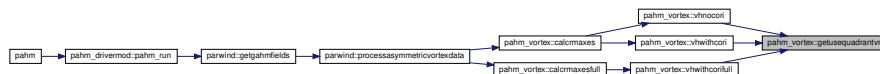
**18.10.1.14 getusequadrantvr()** logical function, public pahm_vortex::getusequadrantvr

Definition at line 1184 of file [vortex.F90](#).

References [usequadrantvr](#).

Referenced by [vhncori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

Here is the caller graph for this function:

**18.10.1.15 getvmaxesbl()** subroutine, public pahm_vortex::getvmaxesbl (real(sz), dimension(4), intent(out) vMaxW)

Definition at line 1020 of file [vortex.F90](#).

References [vmbi](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Definition at line 1283 of file [vortex.F90](#).

References [quadflag4](#), and [quadir4](#).

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

Here is the caller graph for this function:



```
18.10.1.17 newvortex() subroutine, public 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 )
```

Creates a new Vortex object.

A new vortex is created with the essential parameters calculated.

Parameters

in	<i>pinf</i>	Ambient surface pressure (mb)
in	<i>p0</i>	Surface pressure at center of storm (mb)
in	<i>lat</i>	Latitude of storm center (degrees north)
in	<i>lon</i>	Longitude of storm center (degrees east)
in	<i>vm</i>	Max sustained wind velocity in storm (knots)

Definition at line 563 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 [vmlb](#).

```
18.10.1.18 newvortexfull() subroutine, public 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 )
```

Creates a new Vortex object.

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

Parameters

in	<i>pinf</i>	Ambient surface pressure (mb)
in	<i>p0</i>	Surface pressure at center of storm (mb)
in	<i>lat</i>	Latitude of storm center (degrees north)
in	<i>lon</i>	Longitude of storm center (degrees east)
in	<i>vm</i>	Max sustained wind velocity in storm (knots)

Definition at line 617 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 [vml](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Calculate the radius of maximum winds.

Parameters

in	<i>angle</i>	Azimuthal angle (degrees)
----	--------------	---------------------------

Returns

myValOut Rmw: Radius of maximum winds (meters) from curve fit

Definition at line 1362 of file [vortex.F90](#).

References [rmaxes](#).

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

Here is the caller graph for this function:



```
18.10.1.20 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 ) [private]
```

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

Parameters

in	<i>x</i>	x component of vector
in	<i>y</i>	y component of vector
in	<i>angle</i>	Angle to rotate the vector (degrees)
in	<i>whichWay</i>	direction of the rotation whichWay < 0: clockwise whichWay > 0: counter-clockwise
out	<i>xr</i>	x component of rotated vector
out	<i>yr</i>	y component of rotated vector

Definition at line 1795 of file [vortex.F90](#).

References [pahm_global::deg2rad](#).

Referenced by [uvp\(\)](#), and [uvpr\(\)](#).

Here is the caller graph for this function:



18.10.1.21 setisotachraddir() subroutine, public pahm_vortex::setisotachraddir (real(sz), dimension(4), intent(in) ir)

Definition at line 1124 of file [vortex.F90](#).

References [radius](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



18.10.1.22 setisotachwindspeed() subroutine, public pahm_vortex::setisotachwindspeed (real(sz), intent(in) sp)

Definition at line 1154 of file [vortex.F90](#).

References [vr](#).

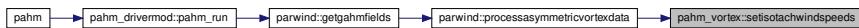
18.10.1.23 setisotachwindspeeds() subroutine, public pahm_vortex::setisotachwindspeeds (real(sz), dimension(4), intent(in) vrQ)

Definition at line 1139 of file [vortex.F90](#).

References [vrquadrant](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Definition at line 697 of file [vortex.F90](#).

References [rmaxes](#).

```
18.10.1.25 setshapeparameter() subroutine, public pahm_vortex::setshapeparameter (
    real(sz) param )
```

Definition at line 1052 of file [vortex.F90](#).

References [b](#).

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

Definition at line 1169 of file [vortex.F90](#).

References [usequadrantvr](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Definition at line 1039 of file [vortex.F90](#).

References [usevmaxesbl](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Definition at line 1001 of file [vortex.F90](#).

References [vmb1](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Sets basic parameters for a new Vortex object.

Aim is to define pn, pc, and corio.

Parameters

in	<i>pinf</i>	Hurricane Ambient pressure (mb)
in	<i>p0</i>	Hurricane central pressure (mb)
in	<i>lat</i>	Latitude of storm center (degrees north)
in	<i>lon</i>	Longitude of storm center (degrees east)

Definition at line 670 of file [vortex.F90](#).

References [clat](#), [clon](#), [corio](#), [pahm_global::deg2rad](#), [pahm_global::omega](#), [pc](#), and [pn](#).

Referenced by [parwind::getgahmfields\(\)](#).

Here is the caller graph for this function:



```
18.10.1.30 spinterp() real(sz) function, public 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.

Aim is to define pn, pc, and corio.

Parameters

in	<i>angle</i>	Azimuthal angle (degrees)
in	<i>dist</i>	Distance to storm Center (nm)
in	<i>opt</i>	Flag to calculate one of rMax/vMax/B

Returns

myValOut The interpolated value for rMax/vMax/B

Definition at line 1222 of file [vortex.F90](#).

References [bs4](#), [interpr\(\)](#), [rmaxes4](#), and [vmb14](#).

Referenced by [parwind::getgahmfields\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.10.1.31 uvp() subroutine, public 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 )
```

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

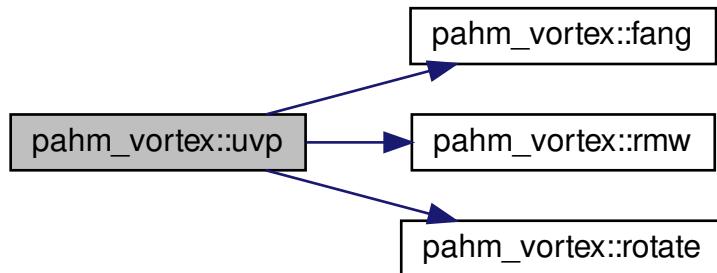
Parameters

in	<i>lat</i>	Latitude of nodal point (degrees north)
in	<i>lon</i>	Longitude of nodal point (degrees east)
in	<i>uTrans</i>	x component of translational velocity (knts)
in	<i>vTrans</i>	y component of translational velocity (knts)
out	<i>u</i>	x component of wind velocity at nodal point (m/s)
out	<i>v</i>	y component of wind velocity at nodal point (m/s)
out	<i>p</i>	Surface pressure at nodal point (Pa)

Definition at line 1441 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:



```
18.10.1.32 uvpr() subroutine, public 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 )
```

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

Parameters

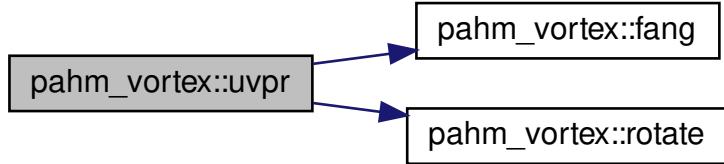
in	<i>iDist</i>	Distance to hurricane center in nautical miles
in	<i>iAngle</i>	Azimuthal angle (degrees)
in	<i>iRmx</i>	Radius of maximum wind (Rmw)
in	<i>iRmxTrue</i>	
in	<i>iB</i>	Holland B parameter
in	<i>iVm</i>	Vortex maximum velocity at upper boundary
in	<i>iPhi</i>	Vortex correction factor
in	<i>uTrans</i>	x component of translational velocity (knts)
in	<i>vTrans</i>	y component of translational velocity (knts)
in	<i>geof</i>	Factor to calculate wind parameters from the asymmetric hurricane vortex (geof = 1)
out	<i>u</i>	x component of wind velocity at nodal point (m/s)
out	<i>v</i>	y component of wind velocity at nodal point (m/s)
out	<i>p</i>	Surface pressure at nodal point (Pa)

Definition at line 1608 of file [vortex.F90](#).

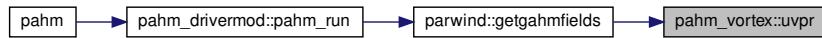
References [b](#), [clat](#), [corio](#), [pahm_global::deg2rad](#), [fang\(\)](#), [pahm_global::kt2ms](#), [pahm_global::mb2pa](#), [pahm_global::nm2m](#), [pahm_global::one2ten](#), [pc](#), [phi](#), [pn](#), [rotate\(\)](#), [vmax](#), and [pahm_global::windreduction](#).

Referenced by [parwind::getgahmfields\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.10.1.33 uvtrans() subroutine, public 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.

Parameters

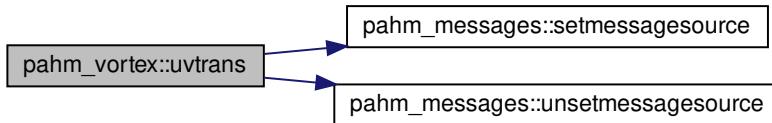
in	<i>lat</i>	Latitude values (vector) of the center (degrees north)
in	<i>lon</i>	Longitude values (vector) of the center (degrees east)
in	<i>times</i>	Time values (vector) at the center locations (seconds)
out	<i>u</i>	x component of the translational velocities (m/s)
out	<i>v</i>	y component of the translational velocities (m/s)
out	<i>status</i>	Error status (0 means no error)
in	<i>order</i>	The accuracy order required for the calculations (1, 2) order <= 1: first order approximation for finite differences order >= 2: second order approximation for finite differences

Definition at line 282 of file [vortex.F90](#).

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

Referenced by [parwind::processhollanddata\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.10.1.34 uvtranspoint() subroutine, public 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.

Parameters

in	<i>lat1</i>	Previous latitude of center (degrees north)
in	<i>lon1</i>	Previous longitude of center (degrees east)
in	<i>lat2</i>	Current latitude of center (degrees north)
in	<i>lon2</i>	Current longitude of center (degrees east)
in	<i>time1</i>	Previous time (seconds)
out	<i>time2</i>	Current time (seconds)
out	<i>u</i>	x component of translational velocity (m/s)
out	<i>v</i>	y component of translational velocity (m/s)

Definition at line 509 of file [vortex.F90](#).

References [pahm_global::deg2rad](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#).

Here is the caller graph for this function:



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

Definition at line 1954 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:



18.10.1.36 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.

Parameters

in	<i>testRMax</i>	Iterative values which converge to root
----	-----------------	---

Returns

myValOut The function's result

Definition at line 1918 of file [vortex.F90](#).

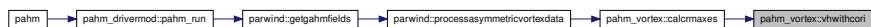
References [b](#), [corio](#), [getusequadrantvr\(\)](#), [pahm_global::kt2ms](#), [pahm_global::ms2kt](#), [pahm_global::nm2m](#), [quad](#), [radius](#), [vmax](#), [vr](#), and [vrquadrant](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.10.1.37 vhwithcorifull() `real(sz) function pahm_vortex::vhwithcorifull (real(sz), intent(in) testRMax) [private]`

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

Parameters

in	<i>testRMax</i>	Iterative values which converge to root
----	-----------------	---

Returns

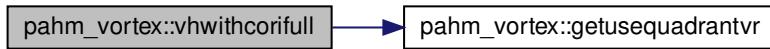
myValOut The function's result

Definition at line 1866 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:



18.10.2 Variable Documentation

18.10.2.1 **b** real(sz) pahm_vortex::b [private]

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

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [findroot\(\)](#), [getshapeparameter\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setshapeparameter\(\)](#), [uvp\(\)](#), [uvpr\(\)](#), [vhncori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.2 **bs** real(sz), dimension(npoints) pahm_vortex::bs [private]

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

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [getshapeparameters\(\)](#), [newvortex\(\)](#), and [newvortexfull\(\)](#).

18.10.2.3 bs4 real(sz), dimension(npoints, 4), public pahm_vortex::bs4

Definition at line 62 of file [vortex.F90](#).

Referenced by [firmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [spinterp\(\)](#).

18.10.2.4 clat real(sz) pahm_vortex::clat [private]

Definition at line 50 of file [vortex.F90](#).

Referenced by [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

18.10.2.5 clon real(sz) pahm_vortex::clon [private]

Definition at line 51 of file [vortex.F90](#).

Referenced by [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), and [uvp\(\)](#).

18.10.2.6 corio real(sz) pahm_vortex::corio [private]

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

Referenced by [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), [uvpr\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.7 latestangle real(sz) pahm_vortex::latestangle [private]

Definition at line 73 of file [vortex.F90](#).

Referenced by [getlatestangle\(\)](#), and [uvp\(\)](#).

18.10.2.8 latestrmax real(sz) pahm_vortex::latestrmax [private]

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

Referenced by [getlatestrmax\(\)](#), and [uvp\(\)](#).

18.10.2.9 npoints integer, parameter pahm_vortex::npoints = NQUADS + 2 [private]

Definition at line 44 of file [vortex.F90](#).

18.10.2.10 nquads integer, parameter pahm_vortex::nquads = 4 [private]

Definition at line 43 of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), and [calcrmaxesfull\(\)](#).

18.10.2.11 pc real(sz) pahm_vortex::pc [private]

Definition at line 49 of file [vortex.F90](#).

Referenced by [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

18.10.2.12 phi real(sz) pahm_vortex::phi [private]

Definition at line 57 of file [vortex.F90](#).

Referenced by [calcrmaxesfull\(\)](#), [newvortexfull\(\)](#), [uvpr\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.13 phis real(sz), dimension([npoints](#)) pahm_vortex::phis [private]

Definition at line 58 of file [vortex.F90](#).

Referenced by [calcrmaxesfull\(\)](#), [getphifactors\(\)](#), and [newvortexfull\(\)](#).

18.10.2.14 phis4 real(sz), dimension([npoints](#), 4) pahm_vortex::phis4 [private]

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

Referenced by [firrmaxes4\(\)](#).

18.10.2.15 pn real(sz) pahm_vortex::pn [private]

Definition at line 48 of file [vortex.F90](#).

Referenced by [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

18.10.2.16 quad integer pahm_vortex::quad [private]

Definition at line 70 of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.17 quadflag4 integer, dimension([npoints](#), 4), public pahm_vortex::quadflag4

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

Referenced by [firmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [interpr\(\)](#).

18.10.2.18 quadir4 real(sz), dimension([npoints](#), 4), public pahm_vortex::quadir4

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

Referenced by [firmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [interpr\(\)](#).

18.10.2.19 radius real(sz), dimension([nquads](#)) pahm_vortex::radius [private]

Definition at line 68 of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [setisotachradii\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.20 rmaxes real(sz), dimension([npoints](#)) pahm_vortex::rmaxes [private]

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

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [firmaxes\(\)](#), [getrmaxes\(\)](#), [rmw\(\)](#), and [setrmaxes\(\)](#).

18.10.2.21 rmaxes4 real(sz), dimension([npoints](#), 4), public pahm_vortex::rmaxes4

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

Referenced by [firmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [spinterp\(\)](#).

18.10.2.22 usequadrantvr logical pahm_vortex::usequadrantvr [private]

Definition at line [74](#) of file [vortex.F90](#).

Referenced by [getusequadrantvr\(\)](#), and [setusequadrantvr\(\)](#).

18.10.2.23 usevmaxesbl logical pahm_vortex::usevmaxesbl [private]

Definition at line [75](#) of file [vortex.F90](#).

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

18.10.2.24 vmax real(sz) pahm_vortex::vmax [private]

Definition at line [52](#) of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [uvp\(\)](#), [uvpr\(\)](#), [vhncori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.25 vmb1 real(sz), dimension([npoints](#)) pahm_vortex::vmb1 [private]

Definition at line [63](#) of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [getvmaxesbl\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), and [setvmaxesbl\(\)](#).

18.10.2.26 vmb14 real(sz), dimension([npoints](#), 4), public pahm_vortex::vmb14

Definition at line [64](#) of file [vortex.F90](#).

Referenced by [firmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [spinterp\(\)](#).

18.10.2.27 vr real(sz) pahm_vortex::vr [private]

Definition at line 56 of file [vortex.F90](#).

Referenced by [setisotachwindspeed\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.10.2.28 vrquadrant real(sz), dimension(nquads) pahm_vortex::vrquadrant [private]

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

Referenced by [setisotachwindspeeds\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

18.11 parwind Module Reference

Data Types

- type [asymmetricvortexdata_t](#)
- 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(s) (GetHollandFields).
- subroutine [processasymmetricvortexdata](#) (idTrFile, strOut, status)
Subroutine to support the asymmetric vortex model(s) (Holland or otherwise).
- subroutine [gethollandfields](#) (timeIDX)
Calculates wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.
- subroutine [getgahmfields](#) (timeIDX)
Calculates wind velocities and MSL pressures at the mesh nodes from the GAHM Wind model.
- subroutine [writebesttrackdata](#) (inpFile, trackStruc, suffix)
Outputs the post-processed best track data to file.
- subroutine [writeasymmetricvortexdata](#) (inpFile, trackStruc, suffix)
Outputs the post-processed best track data to file.
- subroutine [allocbtrstruct](#) (str, nRec)
Subroutine to allocate memory for a best track structure.
- subroutine [deallocbtrstruct](#) (str)
Subroutine to deallocate the memory allocated for a best track structure.
- subroutine [allochollstruct](#) (str, nRec)
Subroutine to allocate memory for a holland structure.
- subroutine [deallochollstruct](#) (str)
Subroutine to deallocate memory of an allocated holland structure.
- subroutine [allocasymvortstruct](#) (str, nRec)
Subroutine to allocate memory for an asymmetric vortex structure.
- subroutine [deallocasymvortstruct](#) (str)
Subroutine to deallocate memory of an allocated asymmetric vortex structure.

Variables

- logical `geostrophicswitch` = .TRUE.
- integer `geofactor` = 1
- integer `method` = 4
- integer `approach` = 2
- integer, parameter, private `stormnamelen` = 10
- type(`besttrackdata_t`), dimension(:), allocatable `besttrackdata`
- type(`hollanddata_t`), dimension(:), allocatable `holstru`
- type(`asymmetricvortexdata_t`), dimension(:), allocatable `asyvortstru`

18.11.1 Function/Subroutine Documentation

18.11.1.1 allocasymvortstruct() subroutine `parwind::allocasymvortstruct` (

```
type(asymmetricvortexdata_t), intent(inout) str,
integer, intent(in) nRec )
```

Subroutine to allocate memory for an asymmetric vortex structure.

Parameters

in, out	<code>str</code>	The asymmetric vortex structure of type AsymmetricVortexData_T
in	<code>nRec</code>	The number of records in the structure

Definition at line 3392 of file `parwind.F90`.

Referenced by `processasymmetricvortexdata()`.

Here is the caller graph for this function:



18.11.1.2 allocbtrstruct() subroutine `parwind::allocbtrstruct` (

```
type(besttrackdata_t), intent(inout) str,
integer, intent(in) nRec )
```

Subroutine to allocate memory for a best track structure.

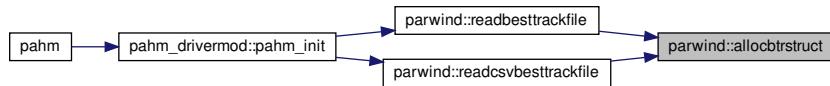
Parameters

in, out	<i>str</i>	The best track structure of type BestTrackData_T
in	<i>nRec</i>	The number of records in the structure

Definition at line 3124 of file [parwind.F90](#).

Referenced by [readbesttrackfile\(\)](#), and [readcsvbesttrackfile\(\)](#).

Here is the caller graph for this function:



18.11.1.3 allochollstruct() subroutine `parwind::allochollstruct` (

```

type(hollanddata\_t), intent(inout) str,
integer, intent(in) nRec )

```

Subroutine to allocate memory for a holland structure.

Parameters

in, out	<i>str</i>	The holland structure of type HollandData_T
in	<i>nRec</i>	The number of records in the structure

Definition at line 3263 of file [parwind.F90](#).

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

Here is the caller graph for this function:



18.11.1.4 deallocasymvortstruct() subroutine `parwind::dallocasymvortstruct` (

```

type(asymmetricvortexdata\_t), intent(inout) str )

```

Subroutine to deallocate memory of an allocated asymmetric vortex structure.

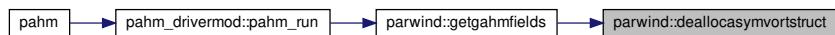
Parameters

in, out	str	The asymmetric vortex structure of type AsymmetricVortexData_T
---------	-----	--

Definition at line 3485 of file [parwind.F90](#).

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

Here is the caller graph for this function:



18.11.1.5 deallocbtrstruct() subroutine parwind::deallocbtrstruct (type([besttrackdata_t](#)), intent(inout) str)

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

Parameters

in, out	str	The best track structure of type BestTrackData_T
---------	-----	--

Definition at line 3193 of file [parwind.F90](#).

18.11.1.6 deallochollstruct() subroutine parwind::deallochollstruct (type([hollanddata_t](#)), intent(inout) str)

Subroutine to deallocate memory of an allocated holland structure.

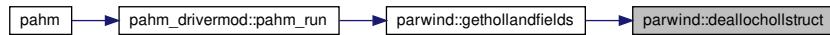
Parameters

in, out	str	The holland structure of type HollandData_T
---------	-----	---

Definition at line 3327 of file [parwind.F90](#).

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

Here is the caller graph for this function:



18.11.1.7 `getgahmfields()` subroutine `parwind::getgahmfields` (

<code>integer, intent(in) timeIDX</code>
--

Calculates wind velocities and MSL pressures at the mesh nodes from the GAHM Wind model.

This subroutine takes a wind file in best track format and uses the GHAM GAHM Wind model to calculate the wind fields (10-m wind speed and MSLP).

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

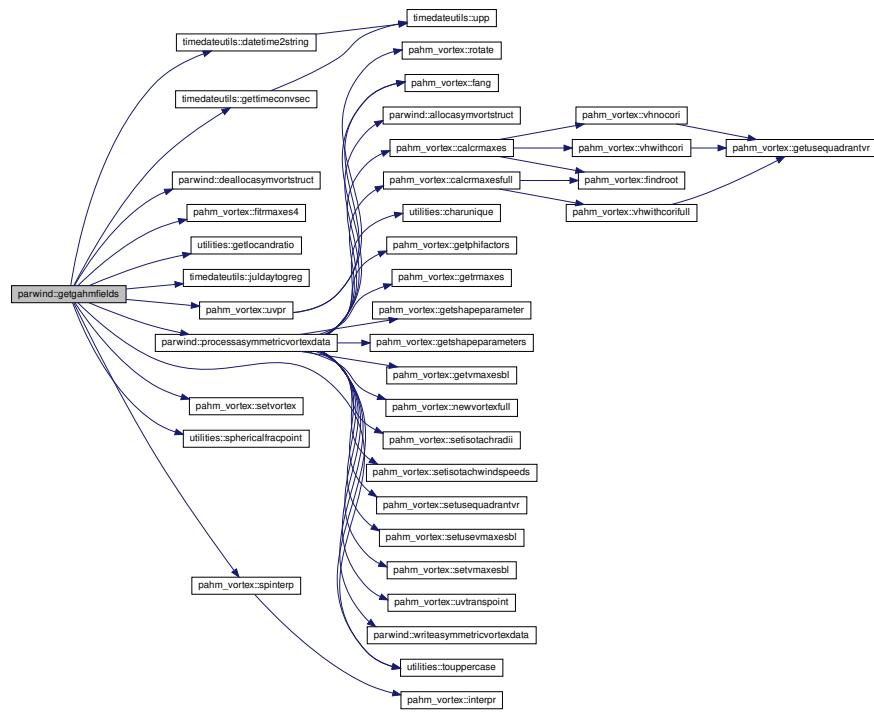
in	<code>timeIDX</code>	The time location to generate the fields for
----	----------------------	--

Definition at line 2356 of file [parwind.F90](#).

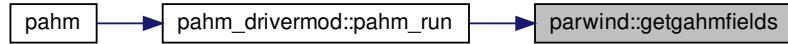
References `asyvortstru`, `pahm_global::backgroundatmpress`, `pahm_global::basee`, `pahm_global::besttrackfilename`, `pahm_vortex::bs4`, `pahm_global::datestimes`, `timedateutils::datetime2string()`, `deallocasymvortstruct()`, `pahm_global::deg2rad`, `pahm_vortex::firmaxes4()`, `geofactor`, `utilities::getlocandratio()`, `timedateutils::gettimeconvsec()`, `pahm_mesh::ismeshok`, `timedateutils::juldaytoreg()`, `pahm_global::kt2ms`, `pahm_global::m2nm`, `pahm_global::mb2kpa`, `pahm_global::mb2pa`, `pahm_global::mdbegsimtime`, `pahm_global::mdoutdt`, `pahm_global::nbtrfiles`, `pahm_global::nm2m`, `pahm_global::noutdt`, `pahm_mesh::np`, `pahm_global::omega`, `pahm_global::one2ten`, `processasymmetricvortexdata()`, `pahm_vortex::quadflag4`, `pahm_vortex::quadir4`, `pahm_global::rad2deg`, `pahm_global::rearth`, `pahm_global::refday`, `pahm_global::refhour`, `pahm_global::refmin`, `pahm_global::refmonth`, `pahm_global::refsec`, `pahm_global::refyear`, `pahm_vortex::rmaxes4`, `pahm_vortex::setvortex()`, `pahm_mesh::sfea`, `pahm_mesh::slam`, `utilities::sphericalfracpoint()`, `pahm_vortex::spinterp()`, `pahm_global::times`, `utilities::touppercase()`, `pahm_vortex::uvpr()`, `pahm_vortex::vmb14`, `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:



18.11.1.8 gethollandfields() 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

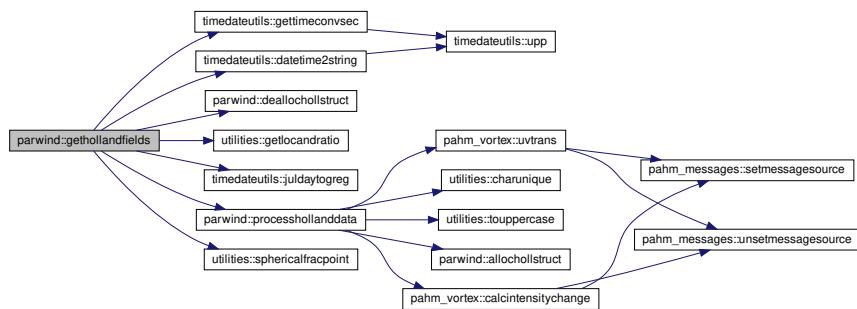
in	<i>timeIDX</i>	The time location to generate the fields for
----	----------------	--

Definition at line 1954 of file [parwind.F90](#).

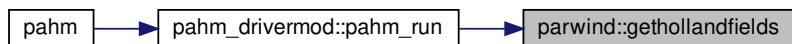
References [pahm_global::backgroundatmpress](#), [pahm_global::basee](#), [pahm_global::besttrackfilename](#), [pahm_global::datestimes](#), [timedateutils::datetime2string\(\)](#), [deallochollstruct\(\)](#), [pahm_global::deg2rad](#), [utilities::getlocandratio\(\)](#), [timedateutils::gettimeconvsec\(\)](#), [holstru](#), [pahm_mesh::ismeshok](#), [timedateutils::juldaytogreg\(\)](#), [pahm_global::mb2kpa](#), [pahm_global::mb2pa](#), [pahm_global::mdbegsimtime](#), [pahm_global::mdoutdt](#), [pahm_global::nbtrfiles](#), [pahm_global::noutdt](#), [pahm_mesh::np](#), [pahm_global::omega](#), [pahm_global::one2ten](#), [processhollanddata\(\)](#), [pahm_global::rad2deg](#), [pahm_global::refday](#), [pahm_global::refhour](#), [pahm_global::refmin](#), [pahm_global::refmonth](#), [pahm_global::refsec](#), [pahm_global::refyear](#), [pahm_global::rhoair](#), [pahm_mesh::sfea](#), [pahm_mesh::slam](#), [utilities::sphericalfracpoint\(\)](#), [pahm_global::times](#), [pahm_global::windreduction](#), [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:



```
18.11.1.9 processasymmetricvortexdata() subroutine parwind::processasymmetricvortexdata (
    integer, intent(in) idTrFile,
    type(asymmetricvortexdata_t), intent(out) strOut,
    integer, intent(out) status )
```

Subroutine to support the asymmetric vortex model(s) (Holland or otherwise).

Subroutine to support asymmetric vortex models. 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

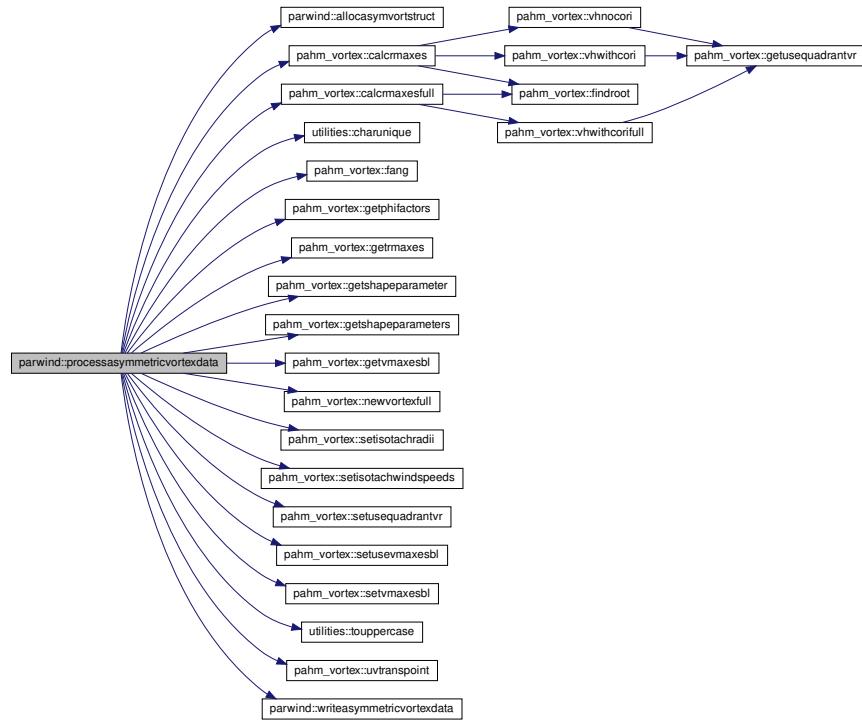
in	<i>idTrFile</i>	The ID of the input track file (1, 2, ...)
out	<i>strOut</i>	The AsymmetricVortexData_T structure that stores all model generated data (output)
out	<i>status</i>	Error status, 0 = no error (output)

Definition at line 1118 of file [parwind.F90](#).

References [allocasymvortstruct\(\)](#), [approach](#), [pahm_global::backgroundatmpress](#), [besttrackdata](#), [pahm_global::besttrackfilename](#), [pahm_vortex::calcrmaxes\(\)](#), [pahm_vortex::calcrmaxesfull\(\)](#), [utilities::charunique\(\)](#), [pahm_global::deg2rad](#), [pahm_vortex::fang\(\)](#), [geostrophicswitch](#), [pahm_vortex::getphifactors\(\)](#), [pahm_vortex::getrmaxes\(\)](#), [pahm_vortex::getshapeparameter\(\)](#), [pahm_vortex::getshapeparameters\(\)](#), [pahm_vortex::getvmaxesbl\(\)](#), [pahm_global::kt2ms](#), [method](#), [pahm_global::ms2kt](#), [pahm_global::nbtrfiles](#), [pahm_vortex::newvortexfull\(\)](#), [pahm_global::nm2m](#), [pahm_global::rad2deg](#), [pahm_vortex::setisotachradii\(\)](#), [pahm_vortex::setisotachwindspeeds\(\)](#), [pahm_vortex::setusequadrantvr\(\)](#), [pahm_vortex::setusevmaxesbl\(\)](#), [pahm_vortex::setvmaxesbl\(\)](#), [utilities::touppercase\(\)](#), [pahm_vortex::uvtranspoint\(\)](#), [pahm_global::windreduction](#), and [writeasymmetricvortexdata\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



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

Subroutine to support the Holland model(s) (GetHollandFields).

Subroutine to support the Holland model (GetHollandFields). 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

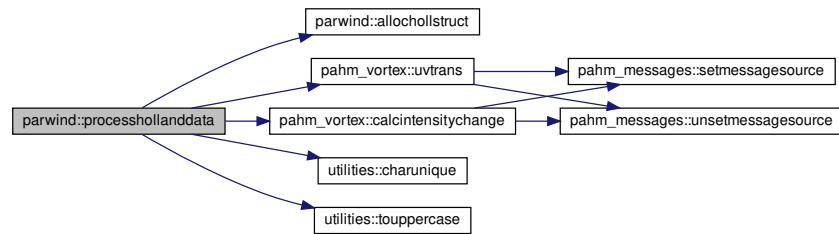
in	<i>idTrFile</i>	The ID of the input track file (1, 2, ...)
out	<i>strOut</i>	The HollandData_T structure that stores all Holland model generated data (output)
out	<i>status</i>	Error status, 0 = no error (output)

Definition at line 894 of file [parwind.F90](#).

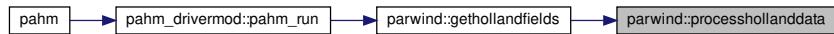
References [allochollstruct\(\)](#), [besttrackdata](#), [pahm_vortex::calcintensitychange\(\)](#), [utilities::charunique\(\)](#), [pahm_global::kt2ms](#), [pahm_global::nbtrfiles](#), [pahm_global::nm2m](#), [utilities::touppercase\(\)](#), and [pahm_vortex::uvtrans\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.11.1.11 `readbesttrackfile()` subroutine `parwind:::readbesttrackfile`

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

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.

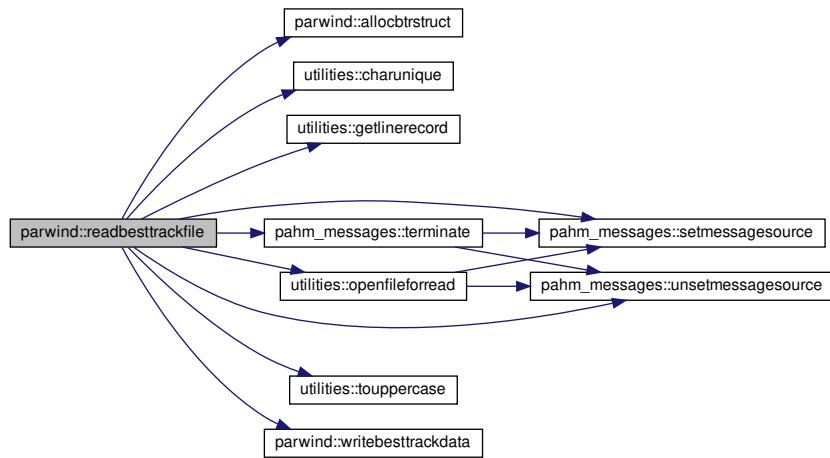
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 266 of file [parwind.F90](#).

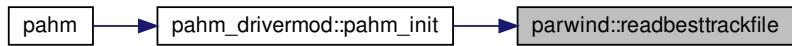
References [allocbtrstruct\(\)](#), [besttrackdata](#), [pahm_global::besttrackfilename](#), [utilities::charunique\(\)](#), [pahm_messages::error](#), [utilities::getline\(\)](#), [pahm_messages::info](#), [pahm_global::lun_btrk](#), [pahm_global::lun_btrk1](#), [pahm_global::nbtrfiles](#), [utilities::openfileforread\(\)](#),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

Referenced by [pahm_drivermod::pahm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.11.1.12 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:

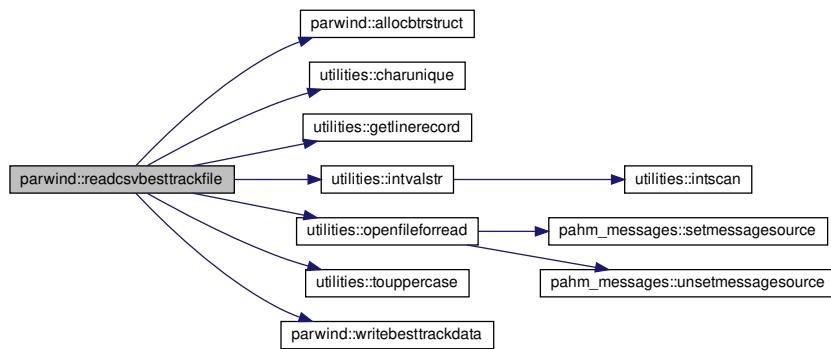
- a-deck: guidance information
- b-deck: best track information
- Skips lines that are time repeats. ???PV check
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 582 of file [parwind.F90](#).

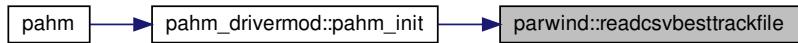
References [allocbtrstruct\(\)](#), [besttrackdata](#), [pahm_global::besttrackfilename](#), [utilities::charunique\(\)](#), [utilities::getlineRecord\(\)](#), [utilities::intvalstr\(\)](#), [pahm_global::nbtrfiles](#), [utilities::openfileforread\(\)](#), [utilities::touppercase\(\)](#), and [writebesttrackdata\(\)](#).

Referenced by [pahm_drivermod::pahm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```

18.11.1.13 writeasymmetricvortexdata() subroutine parwind::writeasymmetricvortexdata (
    character(len=*), intent(in)  inpFile,
    type(asymmetricvortexdata\_t), intent(in)  trackStruc,
    character(len=*), intent(in), optional suffix )
  
```

Outputs the post-processed best track data to file.

Writes the generated asymmetric vortex data in addition to the adjusted best track data into the "extended" best track output file.

Parameters

in	<i>inpFile</i>	The name of the input best track file
in	<i>trackStruc</i>	The "extended" best track data structure that corresponds to the <i>inpFile</i>
in	<i>suffix</i>	The suffix (optional) to be appended to the <i>inpFile</i> (default '_asymvort')

Definition at line 3017 of file [parwind.F90](#).

References [pahm_global::lun_btrk](#), and [pahm_global::lun_btrk1](#).

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

Here is the caller graph for this function:



18.11.1.14 writebesttrackdata() subroutine [parwind::writebesttrackdata](#) (

```

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

Outputs the post-processed best track data to file.

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

Parameters

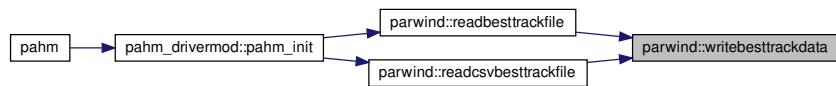
in	<i>inpFile</i>	The name of the input best track file
in	<i>trackStruc</i>	The "adjusted" best track data structure that corresponds to the <i>inpFile</i>
in	<i>suffix</i>	The suffix (optional) to be appended to the <i>inpFile</i> (default '_adj')

Definition at line 2912 of file [parwind.F90](#).

References [pahm_global::lun_btrk](#), and [pahm_global::lun_btrk1](#).

Referenced by [readbesttrackfile\(\)](#), and [readcsvbesttrackfile\(\)](#).

Here is the caller graph for this function:



18.11.2 Variable Documentation

18.11.2.1 approach integer parwind::approach = 2

Definition at line [26](#) of file [parwind.F90](#).

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

18.11.2.2 asyvortstru type([asymmetricvortexdata_t](#)), dimension(:), allocatable parwind::asyvortstru

Definition at line [243](#) of file [parwind.F90](#).

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

18.11.2.3 besttrackdata type([besttrackdata_t](#)), dimension(:), allocatable parwind::besttrackdata

Definition at line [118](#) of file [parwind.F90](#).

Referenced by [processasymmetricvortexdata\(\)](#), [processhollanddata\(\)](#), [readbesttrackfile\(\)](#), and [readcsvbesttrackfile\(\)](#).

18.11.2.4 geofactor integer parwind::geofactor = 1

Definition at line [25](#) of file [parwind.F90](#).

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

18.11.2.5 geostrophicswitch logical parwind::geostrophicswitch = .TRUE.

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

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

18.11.2.6 holstru type([hollanddata_t](#)), dimension(:), allocatable parwind::holstru

Definition at line [158](#) of file [parwind.F90](#).

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

18.11.2.7 method integer parwind::method = 4

Definition at line [26](#) of file [parwind.F90](#).

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

18.11.2.8 stormnamelen integer, parameter, private parwind::stormnamelen = 10 [private]

Definition at line [28](#) of file [parwind.F90](#).

18.12 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 **arraycopsisingle** (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).

- subroutine **swapsingle** (a, b, mask)

Swaps the contents of a and b (single precision).

- subroutine **swapdouble** (a, b, mask)

Swaps the contents of a and b (double precision).

- subroutine **swapintvec** (a, b, mask)

Swaps the contents of a and b (integer).

- subroutine **swapsinglevec** (a, b, mask)

Swaps the contents of a and b (single precision).

- subroutine **swapdoublevec** (a, b, mask)

Swaps the contents of a and b (double precision).

- 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) **artsingle** (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").

18.12.1 Function/Subroutine Documentation

18.12.1.1 **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

in	<i>src</i>	The one-dimensional array to be copied (double precision)
out	<i>dest</i>	The copied array (output)
out	<i>nCP</i>	The number of elements of "src" array that copied (output)
out	<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 1244 of file [sortutils.F90](#).

```
18.12.1.2 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

in	<i>src</i>	The one-dimensional array to be copied (integer)
out	<i>dest</i>	The copied array (output)
out	<i>nCP</i>	The number of elements of "src" array that copied (output)
out	<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 1166 of file [sortutils.F90](#).

```
18.12.1.3 arraycopsingle() subroutine sortutils::arraycopsingle (
    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

in	<i>src</i>	The one-dimensional array to be copied (single precision)
out	<i>dest</i>	The copied array (output)
out	<i>nCP</i>	The number of elements of "src" array that copied (output)
out	<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 1205 of file [sortutils.F90](#).

```
18.12.1.4 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

in	<i>arr1</i>	The first array in the comparison (double precision)
in	<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 1381 of file [sortutils.F90](#).

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

Compares two one-dimensional integer arrays for equality.

Parameters

in	<i>arr1</i>	The first array in the comparison (integer)
in	<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 1281 of file [sortutils.F90](#).

```
18.12.1.6 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

in	<i>arr1</i>	The first array in the comparison (single precision)
in	<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 1326 of file [sortutils.F90](#).

```
18.12.1.7 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

in	<i>first</i>	The value of the first term (double precision)
in	<i>increment</i>	The value of the increment (double precision)
in	<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 1943 of file [sortutils.F90](#).

```
18.12.1.8 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

in	<i>first</i>	The value of the first term (integer)
in	<i>increment</i>	The value of the increment (integer)
in	<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 1809 of file [sortutils.F90](#).

```
18.12.1.9 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

in	<i>first</i>	The value of the first term (single precision)
in	<i>increment</i>	The value of the increment (single precision)
in	<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 1876 of file [sortutils.F90](#).

18.12.1.10 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

in	<i>arr1D</i>	The array to be indexed (double precision)
out	<i>idx1D</i>	The array of "indexed" indexes of arr1D (output)
out	<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](#).

18.12.1.11 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

in	<i>arr1D</i>	The array to be indexed (integer)
out	<i>idx1D</i>	The array of "indexed" indexes of arr1D (output)
out	<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](#).

```
18.12.1.12 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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 257 of file [sortutils.F90](#).

```
18.12.1.13 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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 607 of file [sortutils.F90](#).

```
18.12.1.14 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

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

Definition at line 430 of file [sortutils.F90](#).

```
18.12.1.15 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

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

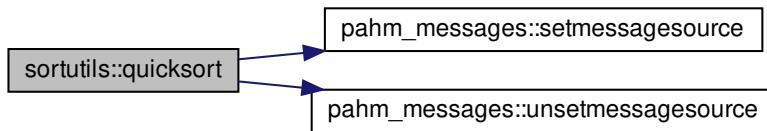
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 951 of file [sortutils.F90](#).

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

Here is the call graph for this function:



```
18.12.1.16 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

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

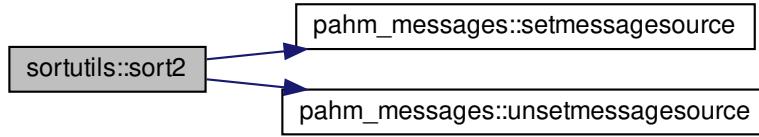
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1095 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:



```
18.12.1.17 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

in	<i>str1</i>	The first string in the comparison
in	<i>str2</i>	The second string in the comparison
in	<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 1440 of file [sortutils.F90](#).

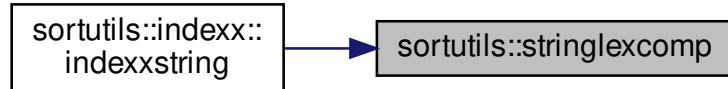
References [utilities::touppercase\(\)](#).

Referenced by [sortutils::indexx::indexxstring\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.12.1.18 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1607 of file [sortutils.F90](#).

```
18.12.1.19 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1760 of file [sortutils.F90](#).

```
18.12.1.20 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1505 of file [sortutils.F90](#).

18.12.1.21 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1658 of file [sortutils.F90](#).

18.12.1.22 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1556 of file [sortutils.F90](#).

```
18.12.1.23 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1709 of file [sortutils.F90](#).

18.13 timedateutils Module Reference

Data Types

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

Functions/Subroutines

- subroutine `timeconviseC` (iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)
Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- subroutine `timeconvrsec` (iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)
Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- logical function `leapyear` (iYear)
Checks for a leap year.
- integer function `yeardays` (iYear)
Determines the days of the year.
- integer function `monthdays` (iYear, iMonth)
Determines the days in the month of the year.
- integer function `dayofyear` (iYear, iMonth, iDay)
Determines the day of the year.
- real(sz) function `gregtojulayisec` (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
Determines the Julian date from a Gregorian date.
- real(sz) function `gregtojulaysec` (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 `splittatetimestring` (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
Splits a date string into components.
- subroutine `splittatetimestring2` (inDateTime, iDate, iTime)
Splits a date string into two components.
- character(len=len(indatetim)) 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.
- character(len=inpstring)) function, private `upp` (inpString)
Convert a string to upper-case.

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 `mdftime` = UNIXTIME
- real(hp), parameter `mdjoffset` = OFFUNIXJULDAY

18.13.1 Function/Subroutine Documentation

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

Constructs a NetCDF time string.

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

Parameters

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

Returns

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

Definition at line 1335 of file [timedateutils.F90](#).

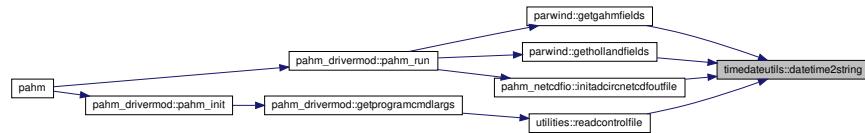
References [upp\(\)](#).

Referenced by [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdfio::initadcircnetcdfoutfile\(\)](#), and [utilities::readcontrolfile\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



18.13.1.2 dayofyear() integer function `timedateutils::dayofyear` (

<code>integer, intent(in) iYear,</code>
<code>integer, intent(in) iMonth,</code>
<code>integer, intent(in) iDay)</code>

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

<code>in</code>	<code>iYear</code>	The year (YYYY, integer, 1582 \leq YYYY)
<code>in</code>	<code>iMonth</code>	The month of the year (MM, integer, 1 \leq MM \leq 12)
<code>in</code>	<code>iDay</code>	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](#).

18.13.1.3 `dayofyeartogreg()` subroutine `timedateutils::dayofyeartogreg` (

```
    integer, intent(in) inYR,
    integer, intent(in) inDY,
    integer, intent(out) iYear,
    integer, intent(out) iMonth,
    integer, intent(out) iDay )
```

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

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

Parameters

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

Definition at line 1010 of file [timedateutils.F90](#).

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

Here is the call graph for this function:



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

Calculates the elapsed time in seconds.

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

Parameters

in	<i>inTime1</i>	The start time (real)
in	<i>inTime2</i>	The end time (real)
in	<i>inUnits</i>	The units (string, optional) of the time variables. Available options: For converting days to seconds : inUnits = ['DAYS', 'DAY', 'DA', 'D'] For converting hours to seconds: inUnits = ['HOURS', 'HOUR', 'HOU', 'HO', 'H'] For converting seconds to seconds: inUnits = ['SEC', 'SE', 'SC', 'S'] Default: inUnits = ['SEC', 'SE', 'SC', 'S']

Returns

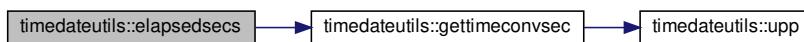
myVal The elapsed time in seconds (real). If this value is very close, within a tolerance, to the nearest whole number, it is set equal to that number.

Definition at line 1516 of file [timedateutils.F90](#).

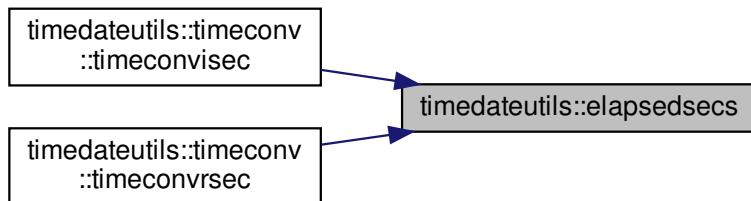
References [gettimeconvsec\(\)](#).

Referenced by [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvrsec\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.13.1.5 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

in	<i>units</i>	The time unit used in the calculations (string: S, M, H, D, W)
in	<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 1430 of file [timedateutils.F90](#).

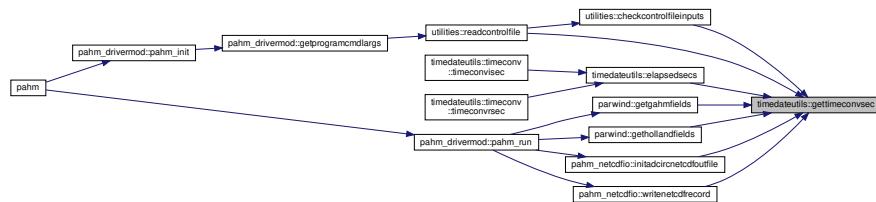
References [upp\(\)](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [elapsedsecs\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), [pahm_netcdffio::initadcircnetcdfoutfile\(\)](#), [utilities::readcontrolfile\(\)](#), and [pahm_netcdffio::writenetcdffrecord\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.13.1.6 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 useful to compute differences between dates.

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

Parameters

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

Returns

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

Note

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

Definition at line 775 of file [timedateutils.F90](#).

```
18.13.1.7 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

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

Returns

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

Note

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

Definition at line 536 of file [timedateutils.F90](#).

```
18.13.1.8 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

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

Returns

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

Note

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

Definition at line 654 of file [timedateutils.F90](#).

```
18.13.1.9 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

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

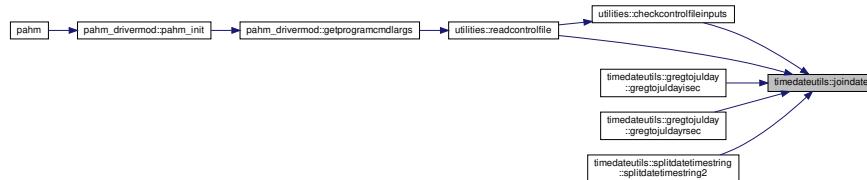
Returns

myValOut The integer date (YYYYMMDD)

Definition at line 1240 of file [timedateutils.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::gregtojuldaysec\(\)](#), [utilities::readcontrolfile\(\)](#), and [timedateutils::splitdatetimestring::splitdatetimestring2\(\)](#).

Here is the caller graph for this function:



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

Determines the Julian date from a Gregorian date.

This subroutine computes the calendar year, month, day, hour, minute and second corresponding to a given Julian date. The inverse of this procedure is the function GregToJulDay. In case of error, year is set equal to IMISSV (-999999). Considers Gregorian dates (after 10/05/1582) only.

The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is usefull to compute differences between dates.

Parameters

in	<i>julDay</i>	The Julian day number (double).
in	<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.</p> <p>Subtracting the large number shifts the zero day to a more recent time (November 17, 1858, midnight) allowing smaller numbers to represent time.</p>
out	<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY, output)
out	<i>iMonth</i>	The month of the year (MM, integer, 1 <= MM <=12, output)
out	<i>iDay</i>	The day of the month (DD, integer, 1 <= DD <=31, output)
out	<i>iHour</i>	The hour of the day (hh, integer, 0 <= hh <= 23, output)
out	<i>iMin</i>	The minute of the hour (mm, integer, 0 <= mm <= 59, output)
out	<i>iSec</i>	The second of the minute (ss, integer, 0 <= ss <= 59, output)

Note

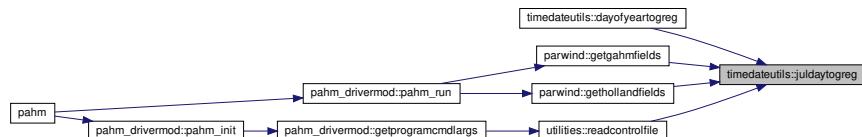
The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

Definition at line 898 of file [timedateutils.F90](#).

References [mdjoffset](#), [offfirstgregday](#), and [usemodjulday](#).

Referenced by [dayofyeartogreg\(\)](#), [parwind::getgahmfields\(\)](#), [parwind::gethollandfields\(\)](#), and [utilities::readcontrolfile\(\)](#).

Here is the caller graph for this function:



18.13.1.11 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

in	<i>iYear</i>	The year (YYYY, integer, 1582 <= YYYY)
----	--------------	--

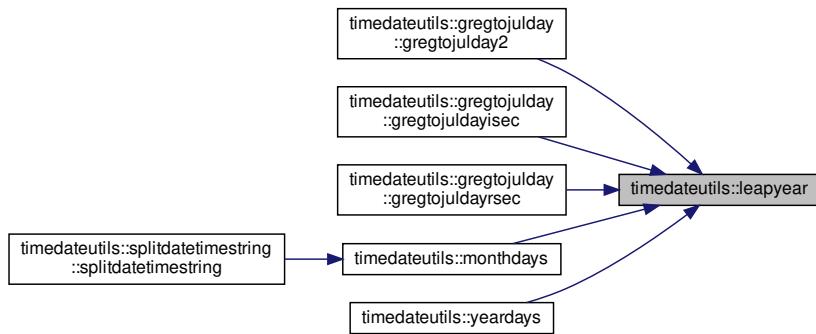
Returns

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

Definition at line 315 of file [timedateutils.F90](#).

Referenced by [timedateutils::gregtojulday::gregtojulday2\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::gregmonthdays\(\)](#), and [yeardays\(\)](#).

Here is the caller graph for this function:



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

Determines the days in the month of the year.

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

Parameters

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

Returns

myVal The days of the month

Definition at line 403 of file [timedateutils.F90](#).

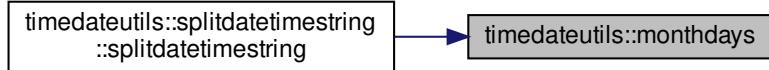
References [pahm_sizes::imissv](#), and [leapyear\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.13.1.13 preprocessdatetimestring() character(len=len(indatetime)) function timedateutils::preprocessdatetimestri  
(  
    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

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

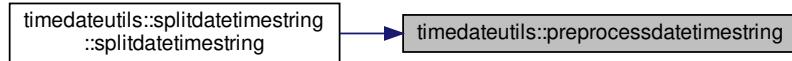
Returns

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

Definition at line 1185 of file [timedateutils.F90](#).

Referenced by [timedateutils::splittimestring::splittimestring\(\)](#).

Here is the caller graph for this function:



18.13.1.14 `splitdate()` subroutine [timedateutils::splitdate](#) (
 integer, intent(in) *inDate*,
 integer, intent(out) *iYear*,
 integer, intent(out) *iMonth*,
 integer, intent(out) *iDay*)

Pre-processes an arbitrary date string.

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

Parameters

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

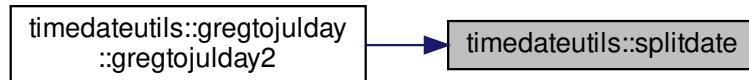
Note

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

Definition at line 1280 of file [timedateutils.F90](#).

Referenced by [timedateutils::gregtojulday::gregtojulday2\(\)](#).

Here is the caller graph for this function:



18.13.1.15 `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

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

Definition at line 1072 of file `timedateutils.F90`.

18.13.1.16 `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

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

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

```
18.13.1.17 timeconvise() subroutine timedateutils::timeconvise (
```

```
    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

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

Definition at line 125 of file [timedateutils.F90](#).

```
18.13.1.18 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

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

Definition at line 202 of file [timedateutils.F90](#).

```
18.13.1.19 upp() character(len(inpstring)) function, private timedateutils::upp (
    character(*), intent(in) inpString ) [private]
```

Convert a string to upper-case.

Parameters

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

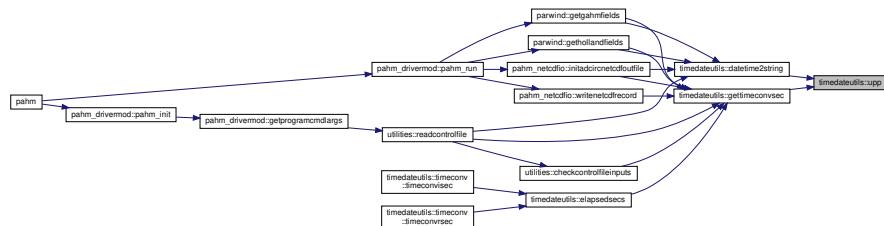
Returns

outString The input string converted to upper case string

Definition at line 1566 of file [timedateutils.F90](#).

Referenced by [datetime2string\(\)](#), and [gettimeconvsec\(\)](#).

Here is the caller graph for this function:



```
18.13.1.20 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

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

Returns

myVal The days of the year (365 or 366)

Definition at line 366 of file [timedateutils.F90](#).

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

Here is the call graph for this function:



18.13.2 Variable Documentation

```
18.13.2.1 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::gregtojuldaysec\(\)](#).

```
18.13.2.2 firstgregtime integer, parameter timedateutils::firstgregtime = 0 * 10000 + 0 * 100 + 0
```

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

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

18.13.2.3 mdjdate integer, parameter timedateutils::mdjdate = UNIXDATE

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

18.13.2.4 mdjoffset real(hp), parameter timedateutils::mdjoffset = OFFUNIXJULDAY

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

Referenced by [timedateutils::gregtojulday::gregtojulday2\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::greg](#) and [juldaytogram\(\)](#).

18.13.2.5 mdjtime integer, parameter timedateutils::mdjtime = UNIXTIME

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

18.13.2.6 modeldate integer, parameter timedateutils::modeldate = 1990 * 10000 + 1 * 100 + 1

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

18.13.2.7 modeltime integer, parameter timedateutils::modeltime = 0 * 10000 + 0 * 100 + 0

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

18.13.2.8 modjuldate integer, parameter timedateutils::modjuldate = 1858 * 10000 + 11 * 100 + 17

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

18.13.2.9 modjultime integer, parameter timedateutils::modjultime = 0 * 10000 + 0 * 100 + 0

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

18.13.2.10 offfirstgregday real(hp), parameter timedateutils::offfirstgregday = 2299150.5_HP

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

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

18.13.2.11 offmodeljulday real(hp), parameter timedateutils::offmodeljulday = 2447892.5_HP

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

18.13.2.12 offmodjulday real(hp), parameter timedateutils::offmodjulday = 2400000.5_HP

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

18.13.2.13 offunixjulday real(hp), parameter timedateutils::offunixjulday = 2440587.5_HP

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

18.13.2.14 unixdate integer, parameter timedateutils::unixdate = 1970 * 10000 + 1 * 100 + 1

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

18.13.2.15 unixtime integer, parameter timedateutils::unixtime = 0 * 10000 + 0 * 100 + 0

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

18.13.2.16 usemodjulday integer, parameter timedateutils::usemodjulday = 0

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

Referenced by [timedateutils::gregtojulday::gregtojulday2\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#) and [juldaytogram\(\)](#).

18.14 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 **sphericaldistancehav** (lat1, lon1, lat2, lon2)

Calculates the distance of two points along the great circle using the Haversine formula.

- subroutine **sphericalfracpoint** (lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)

Calculates the coordinates of an intermediate point between two points along the great circle.

- subroutine **getlocandratio** (val, arrVal, idx1, idx2, wtRatio)

Calculates the location of a value in an 1D array of values.

- integer function **charunique** (inpVec, outVec, idxVec)

Find the unique non-blank elements in 1D character array.

- real(sp) function **valstr** (String)

Returns the value of the leading double precision real numeric string.

- real(hp) function **dvalstr** (String)

Returns the value of the leading double precision real numeric string.

- integer function **intvalstr** (String)

Returns the value of the leading integer numeric string.

- integer function **realscan** (String, Pos, Value)

Scans string looking for the leading single precision real numeric string.

- integer function **drealscan** (String, Pos, Value)

Scans string looking for the leading double precision real numeric string.

- integer function **intscan** (String, Pos, Signed, Value)

Scans string looking for the leading integer numeric string.

Variables

- integer, private **numbtfiles** = 0
- real(sz), parameter **closetol** = 0.001_SZ

18.14.1 Function/Subroutine Documentation

```
18.14.1.1 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

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

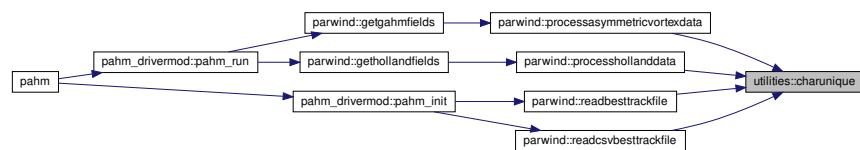
Returns

myRec: The number of the uniques elements in the input array

Definition at line 2553 of file [utilities.F90](#).

Referenced by [parwind::processasymmetricvortexdata\(\)](#), [parwind::processhollanddata\(\)](#), [parwind::readbesttrackfile\(\)](#), and [parwind::readcsvbesttrackfile\(\)](#).

Here is the caller graph for this function:



18.14.1.2 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

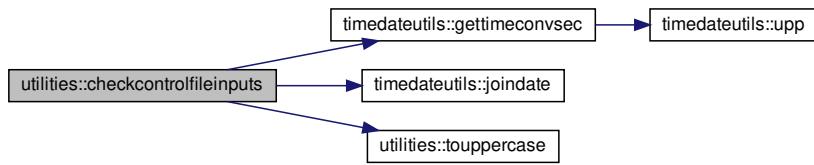
myStatus: The error status, no error: status = 0

Definition at line 1147 of file [utilities.F90](#).

References [pahm_global::backgroundatmpress](#), [pahm_global::begsimspecified](#), [pahm_global::begsimtime](#), [pahm_global::besttrackfilename](#), [pahm_global::besttrackfilenamespecified](#), [closetol](#), [pahm_global::def_ncnam_pres](#), [pahm_global::def_ncnam_wndx](#), [pahm_global::def_ncnam_wndy](#), [pahm_global::defv_atmpress](#), [pahm_global::defv_gravity](#), [pahm_global::defv_rhoair](#), [pahm_global::defv_rhowater](#), [pahm_global::defv_windreduction](#), [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](#), [numbtfiles](#), [pahm_global::outdt](#), [pahm_global::outfile](#), [pahm_global::outfilenamespecified](#), [pahm_global::refdate](#), [pahm_global::refdatetime](#), [pahm_global::reftime](#), [pahm_global::rhoair](#), [pahm_global::rhowater](#), [touppercase\(\)](#), [pahm_global::unitime](#), and [pahm_global::windreduction](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.14.1.3 convlon() `real(sz) function utilities::convlon (real(sz), intent(in) inpLon)`

Convert longitude values from the (0, 360) to the (-180, 180) notation.

Parameters

in	<i>inpLon</i>	The longitude value to be converted
----	---------------	-------------------------------------

Returns

myValOut: The converted longitude value

Definition at line 1767 of file [utilities.F90](#).

18.14.1.4 cptestgeo_1d() `subroutine utilities::cptestgeo_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

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

Definition at line 1947 of file [utilities.F90](#).

18.14.1.5 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

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

Definition at line 1900 of file [utilities.F90](#).

18.14.1.6 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

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

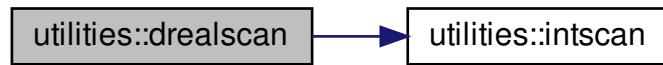
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib_win.html

Definition at line 2919 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:



18.14.1.7 dvalstr() `real(hp) function utilities::dvalstr (character(len=*) , intent(in) String)`

Returns the value of the leading double precision real numeric string.

Parameters

in	<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

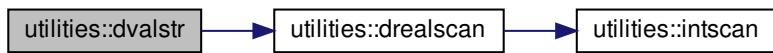
See also

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

Definition at line 2661 of file [utilities.F90](#).

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

Here is the call graph for this function:



```
18.14.1.8 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

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

Definition at line 1854 of file [utilities.F90](#).

```
18.14.1.9 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

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

Definition at line 1807 of file [utilities.F90](#).

```
18.14.1.10 getlinerecord() integer function utilities::getlinerecord (
    character(len=*), intent(in) inpLine,
    character(len=len(inpline)), intent(out) outLine,
    integer, optional lastCommFlag )
```

Gets a line from a file.

This function reads a line record, which is neither a commented or a blank line, from a file for further processing. Commented lines are those with a first character either "#" or "!".

Parameters

in	<i>inpLine</i>	The input text line
in	<i>lastCommFlag</i>	Optional flag to check/remove commented portion at the right of the text line <i>lastCommFlag</i> <= 0 do nothing <i>lastCommFlag</i> > 0 check for "#!" symbols at the right of the text line and remove that portion of the line
out	<i>outLine</i>	The output line (the left adjusted input line)

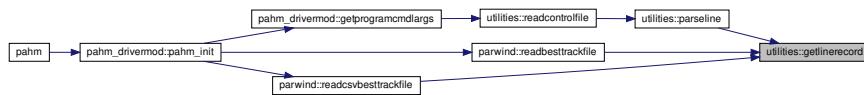
Returns

myLen: The length of *outLine* (end blanks removed)

Definition at line 780 of file [utilities.F90](#).

Referenced by [parseline\(\)](#), [parwind::readbesttrackfile\(\)](#), and [parwind::readcsvbesttrackfile\(\)](#).

Here is the caller graph for this function:



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

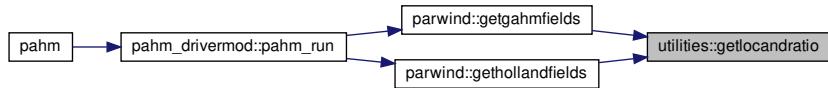
Parameters

in	<i>val</i>	The value to search for, such that $\text{arrVal}(\text{idx1}) \leq \text{val} \leq \text{arrVal}(\text{idx2})$
in	<i>arrVal</i>	The one-dimensional array to search (PV ordered in ascending order?)
out	<i>idx1</i>	The index of the lowest array bound such that: $\text{arrVal}(\text{idx1}) \leq \text{val}$ (output)
out	<i>idx2</i>	The index of the highest array bound such that: $\text{arrVal}(\text{idx2}) \geq \text{val}$ (output)
out	<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 2437 of file [utilities.F90](#).

Referenced by [parwind::getgahmfields\(\)](#), and [parwind::gethollandfields\(\)](#).

Here is the caller graph for this function:



18.14.1.12 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

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

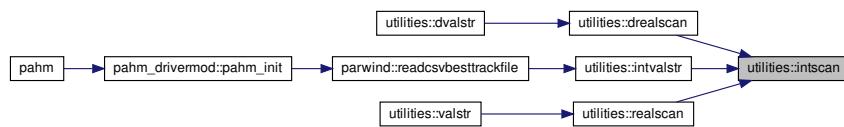
See also

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

Definition at line 3072 of file [utilities.F90](#).

Referenced by [drealscan\(\)](#), [intvalstr\(\)](#), and [realscan\(\)](#).

Here is the caller graph for this function:



18.14.1.13 intvalstr() integer function utilities::intvalstr (
character(len=*), intent(in) String)

Returns the value of the leading integer numeric string.

Parameters

in	String	The input string
----	--------	------------------

Returns

myVal: The value of the integer number as extracted from the input string

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

See also

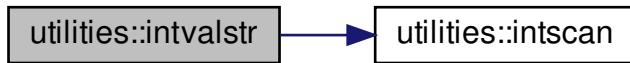
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdfplib/nsdfplib_win.html

Definition at line 2703 of file [utilities.F90](#).

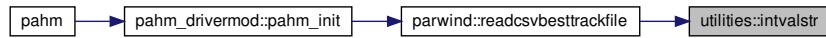
References [intscan\(\)](#).

Referenced by [parwind::readcsvbesttrackfile\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.14.1.14 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

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

Returns

`nValsOut`: Number of processed output values

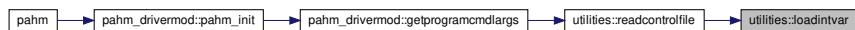
Note

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

Definition at line 1485 of file [utilities.F90](#).

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

Here is the caller graph for this function:



```

18.14.1.15 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

in	<code>nInp</code>	Number of input values
in	<code>vInp</code>	Array of input values
in	<code>nOut</code>	Number of output values
out	<code>vOut</code>	Array of output values (logical, output)

Returns

`nValsOut`: Number of processed output values

Note

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

Definition at line 1551 of file [utilities.F90](#).

```
18.14.1.16 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

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

Returns

nValsOut: Number of processed output values

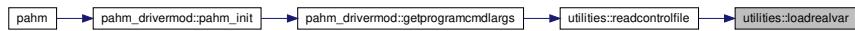
Note

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

Definition at line 1629 of file [utilities.F90](#).

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

Here is the caller graph for this function:



```
18.14.1.17 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

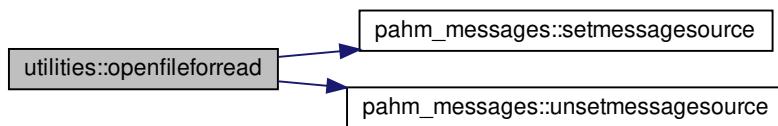
in	<i>lun</i>	The logical unit number (LUN) to use
in	<i>fileName</i>	The full pathname of the input file
out	<i>errorIO</i>	The error status, no error: status = 0 (output)

Definition at line 68 of file [utilities.F90](#).

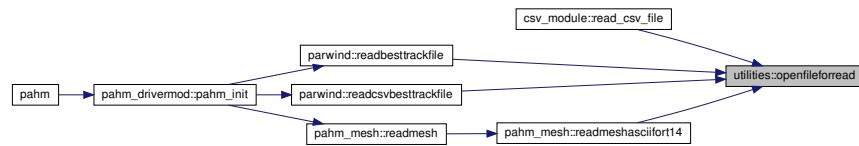
References [pahm_messages::error](#), [pahm_messages::info](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::read_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:



```

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

Returns

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

Note

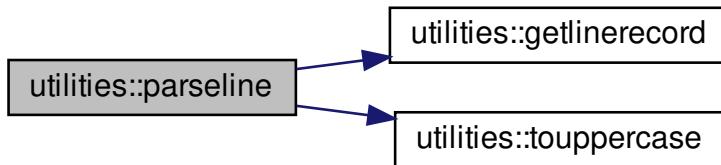
Adopted from the ROMS source (Utility/inp_par.F, decode_line)

Definition at line 876 of file [utilities.F90](#).

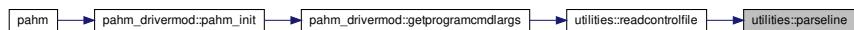
References [getlineRecord\(\)](#), and [touppercase\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



18.14.1.19 printmodelparams() subroutine [utilities::printmodelparams](#)

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

Definition at line 651 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::begsimtime](#), [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](#),

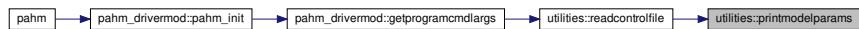
`pahm_global::meshfilename, pahm_global::meshfiletype, pahm_global::modeltype, pahm_global::nbtrfiles, pahm_global::ncdeflate, pahm_global::ncdlevel, pahm_global::ncshuffle, pahm_global::ncvarnam_pres, pahm_global::ncvarnam_wndx, pahm_global::ncvarnam_wndy, pahm_global::noutdt, pahm_global::outdt, pahm_global::outfilename, 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, pahm_global::windreduction, and pahm_global::writeparams.`

Referenced by `readcontrolfile()`.

Here is the call graph for this function:



Here is the caller graph for this function:



18.14.1.20 `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

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

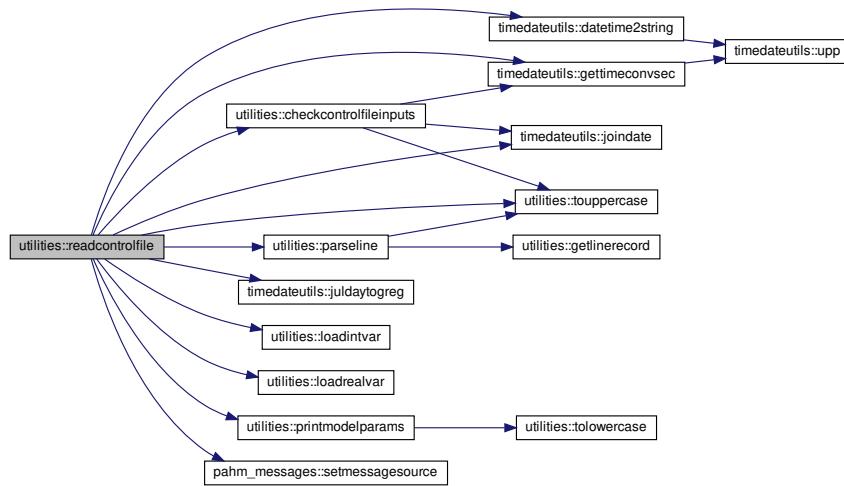
Definition at line 153 of file `utilities.F90`.

References `pahm_sizes::blank, checkcontrolfileinputs(), closetol, timedateutils::datetime2string(), timedateutils::gettimeconvsec(), pahm_sizes::imissv, timedateutils::joindate(), timedateutils::juldaytoreg(), loadintvar(), loadrealvar(), pahm_global::lun_ctrl,`

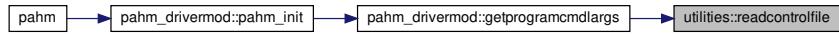
`pahm_global::lun_screen`, `numbtfiles`, `parseline()`, `printmodelparams()`, `pahm_messages::setmessagesource()`, and `touppercase()`.

Referenced by `pahm_drivermod::getprogramcmdargs()`.

Here is the call graph for this function:



Here is the caller graph for this function:



18.14.1.21 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

in	<i>String</i>	The input string
in	<i>Pos</i>	The position in the input string where the scanning begins
out	<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 2765 of file [utilities.F90](#).

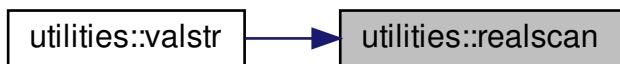
References [intscan\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



```
18.14.1.22 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%27s_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

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

Returns

myValOut: The great-circle distance in meters, 1D array

Definition at line 2068 of file [utilities.F90](#).

```
18.14.1.23 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 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

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

Returns

myValOut: The great-circle distance in meters, 2D array

Definition at line 2167 of file [utilities.F90](#).

```
18.14.1.24 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 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

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

Returns

myValOut: The great-circle distance in meters

Definition at line 2000 of file [utilities.F90](#).

```
18.14.1.25 sphericaldistancehav() real(sz) function utilities::sphericaldistancehav (  
    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

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

Returns

myValOut: The great-circle distance in meters

Definition at line 2268 of file [utilities.F90](#).

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

```
18.14.1.26 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

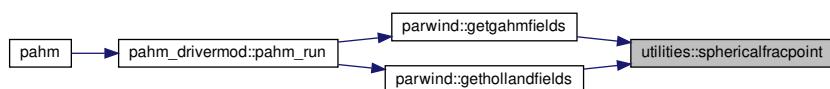
in	<i>lat1</i>	Latitude of the first point (degrees north)
in	<i>lon1</i>	Longitude of the first point (degrees east)
in	<i>lat2</i>	Latitude of the second point (degrees north)
in	<i>lon2</i>	Longitude of the second point (degrees east)
in	<i>fraction</i>	The fraction of the distance between points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

Definition at line 2339 of file [utilities.F90](#).

References [pahm_global::deg2rad](#), [pahm_global::rad2deg](#), and [pahm_global::rearth](#).

Referenced by [parwind::getgahmfields\(\)](#), and [parwind::gethollandfields\(\)](#).

Here is the caller graph for this function:



```
18.14.1.27 tolowercase() pure character(len(inpstring)) function utilities::tolowercase (
    character(*), intent(in) inpString )
```

Convert a string to lower-case.

Parameters

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

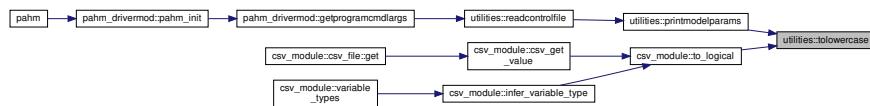
Returns

outString: The ouput string in lower case

Definition at line 1687 of file [utilities.F90](#).

Referenced by [printmodelparams\(\)](#), and [csv_module::to_logical\(\)](#).

Here is the caller graph for this function:



```
18.14.1.28 touppercase() pure character(len(inpstring)) function utilities::touppercase (
    character(*), intent(in) inpString )
```

Convert a string to upper-case.

Parameters

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

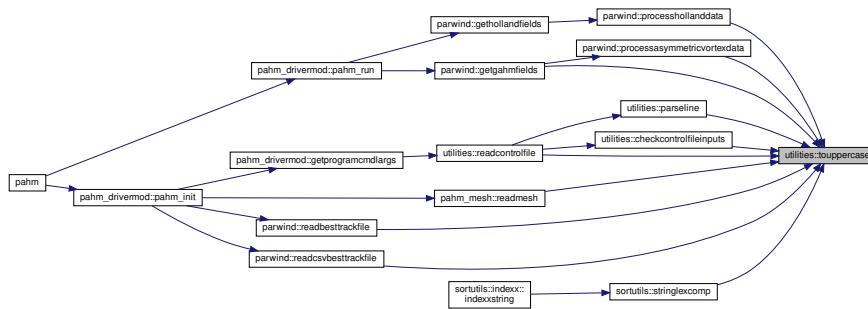
Returns

outString: The ouput string in upper case

Definition at line 1727 of file [utilities.F90](#).

Referenced by [checkcontrolfileinputs\(\)](#), [parwind::getgahmfields\(\)](#), [parseline\(\)](#), [parwind::processasymmetricvortexdata\(\)](#), [parwind::processhollanddata\(\)](#), [parwind::readbesttrackfile\(\)](#), [readcontrolfile\(\)](#), [parwind::readcsvbesttrackfile\(\)](#), [pahm_mesh::readmesh\(\)](#), and [sortutils::stringlexcomp\(\)](#).

Here is the caller graph for this function:



18.14.1.29 valstr() `real(sp) function utilities::valstr (character(len=*), intent(in) String)`

Returns the value of the leading double precision real numeric string.

Parameters

in	<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

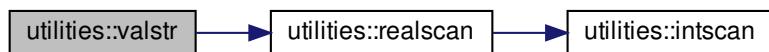
See also

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

Definition at line 2619 of file [utilities.F90](#).

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

Here is the call graph for this function:



18.14.2 Variable Documentation

18.14.2.1 closetol real(sz), parameter utilities::closetol = 0.001_SZ

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

Referenced by [checkcontrolfileinputs\(\)](#), and [readcontrolfile\(\)](#).

18.14.2.2 numbtfiles integer, private utilities::numbtfiles = 0 [private]

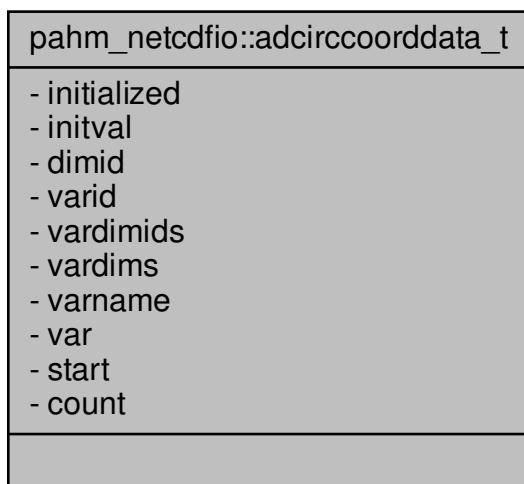
Definition at line 23 of file [utilities.F90](#).

Referenced by [checkcontrolfileinputs\(\)](#), and [readcontrolfile\(\)](#).

19 Data Type Documentation

19.1 pahm_netcdfio::adcirccoorddata_t Type Reference

Collaboration diagram for pahm_netcdfio::adcirccoorddata_t:



Private Attributes

- logical `initialized` = .FALSE.
- real(sz) `initval`
- integer `dimid`
- integer `varid`
- integer `vardimids`
- integer `vardims`
- character(50) `varname`
- real(sz), dimension(:,), allocatable `var`
- integer, dimension(1) `start`
- integer, dimension(1) `count`

19.1.1 Detailed Description

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

19.1.2 Member Data Documentation

19.1.2.1 count integer, dimension(1) pahm_netcdfio::adcirccoorddata_t::count [private]

Definition at line 64 of file [netcdfio.F90](#).

19.1.2.2 dimid integer pahm_netcdfio::adcirccoorddata_t::dimid [private]

Definition at line 58 of file [netcdfio.F90](#).

19.1.2.3 initialized logical pahm_netcdfio::adcirccoorddata_t::initialized = .FALSE. [private]

Definition at line 56 of file [netcdfio.F90](#).

19.1.2.4 initval real(sz) pahm_netcdfio::adcirccoorddata_t::initval [private]

Definition at line 57 of file [netcdfio.F90](#).

19.1.2.5 `start` integer, dimension(1) pahm_ncdfio::adcirccoorddata_t::start [private]

Definition at line 64 of file [netcdfio.F90](#).

19.1.2.6 `var` real(sz), dimension(:), allocatable pahm_ncdfio::adcirccoorddata_t::var [private]

Definition at line 63 of file [netcdfio.F90](#).

19.1.2.7 `vardimids` integer pahm_ncdfio::adcirccoorddata_t::vardimids [private]

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

19.1.2.8 `vardims` integer pahm_ncdfio::adcirccoorddata_t::vardims [private]

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

19.1.2.9 `varid` integer pahm_ncdfio::adcirccoorddata_t::varid [private]

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

19.1.2.10 `varname` character(50) pahm_ncdfio::adcirccoorddata_t::varname [private]

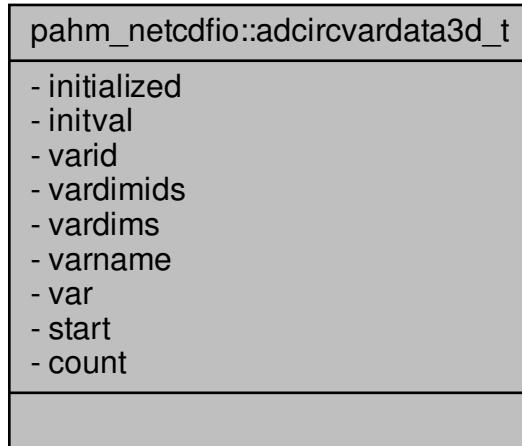
Definition at line 62 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

19.2 pahm_netcdfio::adcircvardata3d_t Type Reference

Collaboration diagram for pahm_netcdfio::adcircvardata3d_t:



Private Attributes

- logical `initialized` = .FALSE.
- real(sz) `initval`
- integer `varid`
- integer, dimension(3) `vardimids`
- integer, dimension(3) `vardims`
- character(50) `varname`
- real(sz), dimension(:, :, :), allocatable `var`
- integer, dimension(3) `start`
- integer, dimension(3) `count`

19.2.1 Detailed Description

Definition at line 78 of file [netcdfio.F90](#).

19.2.2 Member Data Documentation

19.2.2.1 count integer, dimension(3) pahm_netcdfio::adcircvardata3d_t::count [private]

Definition at line 86 of file [netcdfio.F90](#).

19.2.2.2 initialized logical pahm_netcdfio::adcircvardata3d_t::initialized = .FALSE. [private]

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

19.2.2.3 initval real(sz) pahm_netcdfio::adcircvardata3d_t::initval [private]

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

19.2.2.4 start integer, dimension(3) pahm_netcdfio::adcircvardata3d_t::start [private]

Definition at line 86 of file [netcdfio.F90](#).

19.2.2.5 var real(sz), dimension(:, :, :), allocatable pahm_netcdfio::adcircvardata3d_t::var [private]

Definition at line 85 of file [netcdfio.F90](#).

19.2.2.6 vardimids integer, dimension(3) pahm_netcdfio::adcircvardata3d_t::vardimids [private]

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

19.2.2.7 vardims integer, dimension(3) pahm_netcdfio::adcircvardata3d_t::vardims [private]

Definition at line 83 of file [netcdfio.F90](#).

19.2.2.8 varid integer pahm_netcdfio::adcircvardata3d_t::varid [private]

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

19.2.2.9 varname character(50) pahm_ncdfio::adcircvardata3d_t::varname [private]

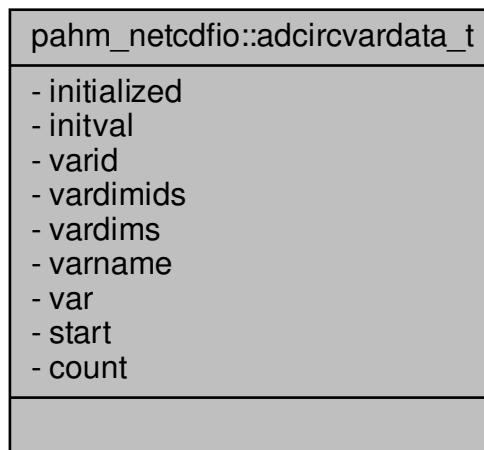
Definition at line 84 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

19.3 pahm_ncdfio::adcircvardata_t Type Reference

Collaboration diagram for pahm_ncdfio::adcircvardata_t:



Private Attributes

- logical `initialized` = .FALSE.
- real(sz) `initval`
- integer `varid`
- integer, dimension(2) `vardimids`
- integer, dimension(2) `vardims`
- character(50) `varname`
- real(sz), dimension(:, :,), allocatable `var`
- integer, dimension(2) `start`
- integer, dimension(2) `count`

19.3.1 Detailed Description

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

19.3.2 Member Data Documentation

19.3.2.1 count integer, dimension(2) pahm_netcdfio::adcircvardata_t::count [private]

Definition at line 75 of file [netcdfio.F90](#).

19.3.2.2 initialized logical pahm_netcdfio::adcircvardata_t::initialized = .FALSE. [private]

Definition at line 68 of file [netcdfio.F90](#).

19.3.2.3 initval real(sz) pahm_netcdfio::adcircvardata_t::initval [private]

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

19.3.2.4 start integer, dimension(2) pahm_netcdfio::adcircvardata_t::start [private]

Definition at line 75 of file [netcdfio.F90](#).

19.3.2.5 var real(sz), dimension(:, :,), allocatable pahm_netcdfio::adcircvardata_t::var [private]

Definition at line 74 of file [netcdfio.F90](#).

19.3.2.6 vardimids integer, dimension(2) pahm_netcdfio::adcircvardata_t::vardimids [private]

Definition at line 71 of file [netcdfio.F90](#).

19.3.2.7 vardims integer, dimension(2) pahm_netcdfio::adcircvardata_t::vardims [private]

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

19.3.2.8 varid integer pahm_netcdfio::adcircvardata_t::varid [private]

Definition at line 70 of file [netcdfio.F90](#).

19.3.2.9 varname character(50) pahm_netcdfio::adcircvardata_t::varname [private]

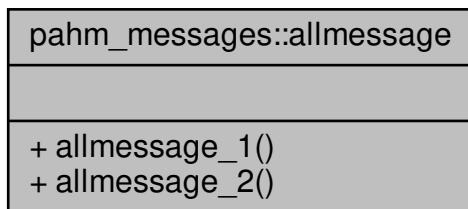
Definition at line 73 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

19.4 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)

19.4.1 Detailed Description

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

19.4.2 Member Function/Subroutine Documentation

19.4.2.1 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.

Parameters

in	<i>message</i>	The message to display
----	----------------	------------------------

Definition at line 318 of file [messages.F90](#).

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

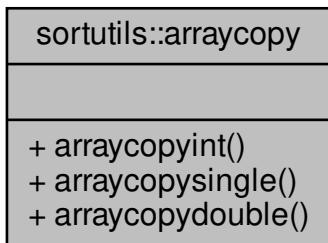
Definition at line 330 of file [messages.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[messages.F90](#)

19.5 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 [arraycopsingle](#) (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".

19.5.1 Detailed Description

Definition at line 38 of file [sortutils.F90](#).

19.5.2 Member Function/Subroutine Documentation

```
19.5.2.1 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

in	<i>src</i>	The one-dimensional array to be copied (double precision)
out	<i>dest</i>	The copied array (output)
out	<i>nCP</i>	The number of elements of "src" array that copied (output)
out	<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 1244 of file [sortutils.F90](#).

```
19.5.2.2 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

in	<i>src</i>	The one-dimensional array to be copied (integer)
out	<i>dest</i>	The copied array (output)
out	<i>nCP</i>	The number of elements of "src" array that copied (output)
out	<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 1166 of file [sortutils.F90](#).

```
19.5.2.3 arraycopsingle() subroutine sortutils::arraycopy::arraycopsingle (
    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

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

Note

Adopted from Numerical Recipes for Fortran 90

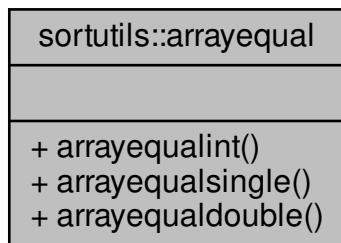
Definition at line 1205 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

19.6 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 `arrayeqaldouble (arr1, arr2)`
Compares two one-dimensional double precision arrays for equality.

19.6.1 Detailed Description

Definition at line 44 of file `sortutils.F90`.

19.6.2 Member Function/Subroutine Documentation

19.6.2.1 arrayeqaldouble() logical function `sortutils::arrayequal::arrayeqaldouble (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

in	<code>arr1</code>	The first array in the comparison (double precision)
in	<code>arr2</code>	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 1381 of file `sortutils.F90`.

19.6.2.2 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

in	<i>arr1</i>	The first array in the comparison (integer)
in	<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 1281 of file [sortutils.F90](#).

19.6.2.3 arrayqualsingle() logical function sortutils::arrayequal::arrayqualsingle (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

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

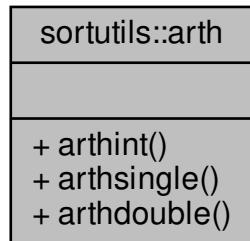
Definition at line 1326 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

19.7 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").

19.7.1 Detailed Description

Definition at line 32 of file [sortutils.F90](#).

19.7.2 Member Function/Subroutine Documentation

```
19.7.2.1 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

in	<i>first</i>	The value of the first term (double precision)
in	<i>increment</i>	The value of the increment (double precision)
in	<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 1943 of file [sortutils.F90](#).

```
19.7.2.2 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

in	<i>first</i>	The value of the first term (integer)
in	<i>increment</i>	The value of the increment (integer)
in	<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 1809 of file [sortutils.F90](#).

```
19.7.2.3 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

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

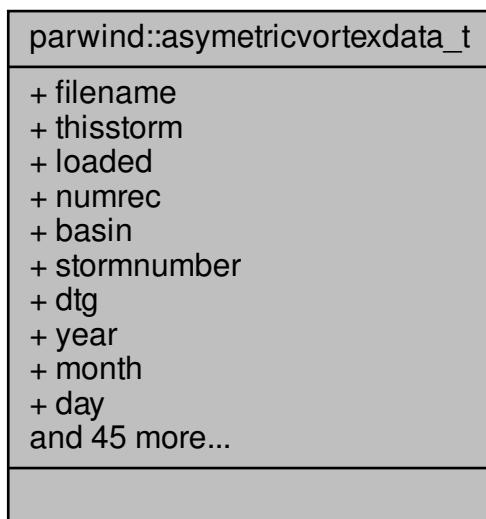
Definition at line 1876 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

19.8 parwind::asymmetricvortexdata_t Type Reference

Collaboration diagram for parwind::asymmetricvortexdata_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`
- integer, dimension(:), allocatable `casttypenum`
- 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`
- character(len=1), dimension(:), allocatable `ns`
- character(len=1), dimension(:), allocatable `ew`
- integer, dimension(:), allocatable `ispeed`
- real(sz), dimension(:), allocatable `speed`
- integer, dimension(:), allocatable `icpress`
- real(sz), dimension(:), allocatable `cpress`
- character(len=2), dimension(:), allocatable `ty`
- integer, dimension(:), allocatable `ivr`
- character(len=3), dimension(:), allocatable `windcode`
- integer, dimension(:, :), allocatable `ir`
- integer, dimension(:), allocatable `iprp`
- real(sz), dimension(:), allocatable `prp`
- integer, dimension(:), allocatable `irrp`
- real(sz), dimension(:), allocatable `rrp`
- integer, dimension(:), allocatable `irmw`
- real(sz), dimension(:), allocatable `rmw`
- integer, dimension(:), allocatable `gusts`
- integer, dimension(:), allocatable `eye`
- character(len=3), dimension(:), allocatable `subregion`
- integer, dimension(:), allocatable `maxseas`
- character(len=3), dimension(:), allocatable `initials`
- real(sz), dimension(:), allocatable `trvx`
- real(sz), dimension(:), allocatable `tryv`
- integer, dimension(:), allocatable `idir`
- real(sz), dimension(:), allocatable `dir`
- integer, dimension(:), allocatable `istormspeed`
- real(sz), dimension(:), allocatable `stormspeed`
- character(len=`stormnamelen`), dimension(:), allocatable `stormname`
- integer `ncycles`
- integer, dimension(:), allocatable `numcycle`

- integer, dimension(:), allocatable `isotachspercycle`
- integer, dimension(:, :), allocatable `quadflag`
- real(sz), dimension(:, :, :), allocatable `rmaxw`
- real(sz), dimension(:, :, :), allocatable `hollb`
- real(sz), dimension(:, :, :), allocatable `holls`
- real(sz), dimension(:, :, :), allocatable `vmaxesbl`

19.8.1 Detailed Description

Definition at line 164 of file [parwind.F90](#).

19.8.2 Member Data Documentation

19.8.2.1 basin character(len=2), dimension(:), allocatable `parwind::asymmetricvortexdata_t::basin`

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

19.8.2.2 casttime real(sz), dimension(:), allocatable `parwind::asymmetricvortexdata_t::casttime`

Definition at line 174 of file [parwind.F90](#).

19.8.2.3 casttype character(len=4), dimension(:), allocatable `parwind::asymmetricvortexdata_t::casttype`

Definition at line 176 of file [parwind.F90](#).

19.8.2.4 casttypenum integer, dimension(:), allocatable `parwind::asymmetricvortexdata_t::casttypenum`

Definition at line 175 of file [parwind.F90](#).

19.8.2.5 cpress real(sz), dimension(:), allocatable `parwind::asymmetricvortexdata_t::cpress`

Definition at line 187 of file [parwind.F90](#).

19.8.2.6 day integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::day

Definition at line 173 of file [parwind.F90](#).

19.8.2.7 dir real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::dir

Definition at line 224 of file [parwind.F90](#).

19.8.2.8 dtg character(len=10), dimension(:), allocatable parwind::asymmetricvortexdata_t::dtg

Definition at line 172 of file [parwind.F90](#).

19.8.2.9 ew character(len=1), dimension(:), allocatable parwind::asymmetricvortexdata_t::ew

Definition at line 181 of file [parwind.F90](#).

19.8.2.10 eye integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::eye

Definition at line 206 of file [parwind.F90](#).

19.8.2.11 fcstinc integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::fcstinc

Definition at line 177 of file [parwind.F90](#).

19.8.2.12 filename character(len=fnamelen) parwind::asymmetricvortexdata_t::filename

Definition at line 165 of file [parwind.F90](#).

19.8.2.13 gusts integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::gusts

Definition at line 205 of file [parwind.F90](#).

19.8.2.14 `hollb` `real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::hollb`

Definition at line 238 of file [parwind.F90](#).

19.8.2.15 `holls` `real(sz), dimension(:, :), allocatable parwind::asymmetricvortexdata_t::holls`

Definition at line 239 of file [parwind.F90](#).

19.8.2.16 `hour` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::hour`

Definition at line 173 of file [parwind.F90](#).

19.8.2.17 `icpress` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::icpress`

Definition at line 186 of file [parwind.F90](#).

19.8.2.18 `idir` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::idir`

Definition at line 223 of file [parwind.F90](#).

19.8.2.19 `ilat` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ilat`

Definition at line 179 of file [parwind.F90](#).

19.8.2.20 `ilon` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ilon`

Definition at line 179 of file [parwind.F90](#).

19.8.2.21 `initials` `character(len=3), dimension(:), allocatable parwind::asymmetricvortexdata_t::initials`

Definition at line 218 of file [parwind.F90](#).

19.8.2.22 iprp integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::iprp

Definition at line 196 of file [parwind.F90](#).

19.8.2.23 ir integer, dimension(:, :), allocatable parwind::asymmetricvortexdata_t::ir

Definition at line 193 of file [parwind.F90](#).

19.8.2.24 irmw integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::irmw

Definition at line 202 of file [parwind.F90](#).

19.8.2.25 irrp integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::irrp

Definition at line 199 of file [parwind.F90](#).

19.8.2.26 isotachspercycle integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::isotachspercycle

Definition at line 235 of file [parwind.F90](#).

19.8.2.27 ispeed integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ispeed

Definition at line 183 of file [parwind.F90](#).

19.8.2.28 istormspeed integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::istormspeed

Definition at line 225 of file [parwind.F90](#).

19.8.2.29 ivr integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ivr

Definition at line 191 of file [parwind.F90](#).

19.8.2.30 `lat` `real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::lat`

Definition at line 180 of file [parwind.F90](#).

19.8.2.31 `loaded` `logical parwind::asymmetricvortexdata_t::loaded = .FALSE.`

Definition at line 167 of file [parwind.F90](#).

19.8.2.32 `lon` `real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::lon`

Definition at line 180 of file [parwind.F90](#).

19.8.2.33 `maxseas` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::maxseas`

Definition at line 217 of file [parwind.F90](#).

19.8.2.34 `month` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::month`

Definition at line 173 of file [parwind.F90](#).

19.8.2.35 `ncycles` `integer parwind::asymmetricvortexdata_t::ncycles`

Definition at line 233 of file [parwind.F90](#).

19.8.2.36 `ns` `character(len=1), dimension(:), allocatable parwind::asymmetricvortexdata_t::ns`

Definition at line 181 of file [parwind.F90](#).

19.8.2.37 `numcycle` `integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::numcycle`

Definition at line 234 of file [parwind.F90](#).

19.8.2.38 numrec integer parwind::asymmetricvortexdata_t::numrec

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

19.8.2.39 prp real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::prp

Definition at line 197 of file [parwind.F90](#).

19.8.2.40 quadflag integer, dimension(:, :), allocatable parwind::asymmetricvortexdata_t::quadflag

Definition at line 236 of file [parwind.F90](#).

19.8.2.41 rmaxw real(sz), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::rmaxw

Definition at line 237 of file [parwind.F90](#).

19.8.2.42 rmw real(sz), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::rmw

Definition at line 203 of file [parwind.F90](#).

19.8.2.43 rrp real(sz), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::rrp

Definition at line 200 of file [parwind.F90](#).

19.8.2.44 speed real(sz), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::speed

Definition at line 184 of file [parwind.F90](#).

19.8.2.45 stormname character(len=[stormnamelen](#)), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::stormname

Definition at line 227 of file [parwind.F90](#).

19.8.2.46 stormnumber integer, dimension(:), allocatable `parwind::asymmetricvortexdata_t::stormnumber`

Definition at line 171 of file [parwind.F90](#).

19.8.2.47 stormspeed real(sz), dimension(:), allocatable `parwind::asymmetricvortexdata_t::stormspeed`

Definition at line 226 of file [parwind.F90](#).

19.8.2.48 subregion character(len=3), dimension(:), allocatable `parwind::asymmetricvortexdata_t::subregion`

Definition at line 207 of file [parwind.F90](#).

19.8.2.49 thisstorm character(len=10) `parwind::asymmetricvortexdata_t::thisstorm`

Definition at line 166 of file [parwind.F90](#).

19.8.2.50 trvx real(sz), dimension(:), allocatable `parwind::asymmetricvortexdata_t::trvx`

Definition at line 220 of file [parwind.F90](#).

19.8.2.51 trvy real(sz), dimension(:), allocatable `parwind::asymmetricvortexdata_t::trvy`

Definition at line 220 of file [parwind.F90](#).

19.8.2.52 ty character(len=2), dimension(:), allocatable `parwind::asymmetricvortexdata_t::ty`

Definition at line 189 of file [parwind.F90](#).

19.8.2.53 vmaxesbl real(sz), dimension(:, :), allocatable `parwind::asymmetricvortexdata_t::vmaxesbl`

Definition at line 240 of file [parwind.F90](#).

19.8.2.54 windcode character(len=3), dimension(:), allocatable parwind::asymmetricvortexdata_t \leftarrow
::windcode

Definition at line 192 of file [parwind.F90](#).

19.8.2.55 year integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::year

Definition at line 173 of file [parwind.F90](#).

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

- [/home/takis/CSDL/parwinds-doc/src/parwind.F90](#)

19.9 parwind::besttrackdata_t Type Reference

Collaboration diagram for parwind::besttrackdata_t:

parwind::besttrackdata_t
+ filename + thisstorm + loaded + numrec + basin + cnum + dtg + technum + tech + tau and 31 more...

Public Attributes

- character(len=fnamelen) `filename`
- character(len=10) `thisstorm`
- logical `loaded` = .FALSE.
- integer `numrec`
- character(len=2), dimension(:), allocatable `basin`
- integer, dimension(:), allocatable `cynum`
- character(len=10), dimension(:), allocatable `dtg`
- integer, dimension(:), allocatable `technum`
- character(len=4), dimension(:), allocatable `tech`
- integer, dimension(:), allocatable `tau`
- integer, dimension(:), allocatable `intlat`
- integer, dimension(:), allocatable `intlon`
- character(len=1), dimension(:), allocatable `ew`
- character(len=1), dimension(:), allocatable `ns`
- integer, dimension(:), allocatable `intvmax`
- integer, dimension(:), allocatable `intmslp`
- character(len=2), dimension(:), allocatable `ty`
- integer, dimension(:), allocatable `rad`
- character(len=3), dimension(:), allocatable `windcode`
- integer, dimension(:), allocatable `intrad1`
- integer, dimension(:), allocatable `intrad2`
- integer, dimension(:), allocatable `intrad3`
- integer, dimension(:), allocatable `intrad4`
- integer, dimension(:), allocatable `intpouter`
- integer, dimension(:), allocatable `introuter`
- integer, dimension(:), allocatable `intrmw`
- integer, dimension(:), allocatable `gusts`
- integer, dimension(:), allocatable `eye`
- character(len=3), dimension(:), allocatable `subregion`
- integer, dimension(:), allocatable `maxseas`
- character(len=3), dimension(:), allocatable `initials`
- integer, dimension(:), allocatable `dir`
- integer, dimension(:), allocatable `intspeed`
- character(len=stormnamelen), 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`

19.9.1 Detailed Description

Definition at line 34 of file `parwind.F90`.

19.9.2 Member Data Documentation

19.9.2.1 `basin` character(len=2), dimension(:), allocatable parwind::besttrackdata_t::basin

Definition at line 41 of file [parwind.F90](#).

19.9.2.2 `cyclenum` integer, dimension(:), allocatable parwind::besttrackdata_t::cyclenum

Definition at line 110 of file [parwind.F90](#).

19.9.2.3 `cynum` integer, dimension(:), allocatable parwind::besttrackdata_t::cynum

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

19.9.2.4 `day` integer, dimension(:), allocatable parwind::besttrackdata_t::day

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

19.9.2.5 `dir` integer, dimension(:), allocatable parwind::besttrackdata_t::dir

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

19.9.2.6 `dtg` character(len=10), dimension(:), allocatable parwind::besttrackdata_t::dtg

Definition at line 43 of file [parwind.F90](#).

19.9.2.7 `ew` character(len=1), dimension(:), allocatable parwind::besttrackdata_t::ew

Definition at line 51 of file [parwind.F90](#).

19.9.2.8 `eye` integer, dimension(:), allocatable `parwind::besttrackdata_t::eye`

Definition at line 91 of file [parwind.F90](#).

19.9.2.9 `filename` character(len=fnamelen) `parwind::besttrackdata_t::filename`

Definition at line 35 of file [parwind.F90](#).

19.9.2.10 `gusts` integer, dimension(:), allocatable `parwind::besttrackdata_t::gusts`

Definition at line 90 of file [parwind.F90](#).

19.9.2.11 `hour` integer, dimension(:), allocatable `parwind::besttrackdata_t::hour`

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

19.9.2.12 `initials` character(len=3), dimension(:), allocatable `parwind::besttrackdata_t::initials`

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

19.9.2.13 `intlat` integer, dimension(:), allocatable `parwind::besttrackdata_t::intlat`

Definition at line 49 of file [parwind.F90](#).

19.9.2.14 `intlon` integer, dimension(:), allocatable `parwind::besttrackdata_t::intlon`

Definition at line 50 of file [parwind.F90](#).

19.9.2.15 `intmslp` integer, dimension(:), allocatable `parwind::besttrackdata_t::intmslp`

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

19.9.2.16 intpouter integer, dimension(:), allocatable parwind::besttrackdata_t::intpouter

Definition at line 87 of file [parwind.F90](#).

19.9.2.17 intrad1 integer, dimension(:), allocatable parwind::besttrackdata_t::intrad1

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

19.9.2.18 intrad2 integer, dimension(:), allocatable parwind::besttrackdata_t::intrad2

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

19.9.2.19 intrad3 integer, dimension(:), allocatable parwind::besttrackdata_t::intrad3

Definition at line 83 of file [parwind.F90](#).

19.9.2.20 intrad4 integer, dimension(:), allocatable parwind::besttrackdata_t::intrad4

Definition at line 85 of file [parwind.F90](#).

19.9.2.21 intrmw integer, dimension(:), allocatable parwind::besttrackdata_t::intrmw

Definition at line 89 of file [parwind.F90](#).

19.9.2.22 introuter integer, dimension(:), allocatable parwind::besttrackdata_t::introuter

Definition at line 88 of file [parwind.F90](#).

19.9.2.23 intspeed integer, dimension(:), allocatable parwind::besttrackdata_t::intspeed

Definition at line 105 of file [parwind.F90](#).

19.9.2.24 `intvmax` integer, dimension(:), allocatable `parwind::besttrackdata_t::intvmax`

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

19.9.2.25 `lat` real(sz), dimension(:), allocatable `parwind::besttrackdata_t::lat`

Definition at line 114 of file [parwind.F90](#).

19.9.2.26 `loaded` logical `parwind::besttrackdata_t::loaded = .FALSE.`

Definition at line 37 of file [parwind.F90](#).

19.9.2.27 `lon` real(sz), dimension(:), allocatable `parwind::besttrackdata_t::lon`

Definition at line 114 of file [parwind.F90](#).

19.9.2.28 `maxseas` integer, dimension(:), allocatable `parwind::besttrackdata_t::maxseas`

Definition at line 102 of file [parwind.F90](#).

19.9.2.29 `month` integer, dimension(:), allocatable `parwind::besttrackdata_t::month`

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

19.9.2.30 `ns` character(len=1), dimension(:), allocatable `parwind::besttrackdata_t::ns`

Definition at line 52 of file [parwind.F90](#).

19.9.2.31 `numrec` integer `parwind::besttrackdata_t::numrec`

Definition at line 38 of file [parwind.F90](#).

19.9.2.32 rad integer, dimension(:), allocatable parwind::besttrackdata_t::rad

Definition at line 75 of file [parwind.F90](#).

19.9.2.33 stormname character(len=[stormnamelen](#)), dimension(:), allocatable parwind::besttrackdata_t::stormname

Definition at line 106 of file [parwind.F90](#).

19.9.2.34 subregion character(len=3), dimension(:), allocatable parwind::besttrackdata_t::subregion

Definition at line 92 of file [parwind.F90](#).

19.9.2.35 tau integer, dimension(:), allocatable parwind::besttrackdata_t::tau

Definition at line 47 of file [parwind.F90](#).

19.9.2.36 tech character(len=4), dimension(:), allocatable parwind::besttrackdata_t::tech

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

19.9.2.37 technum integer, dimension(:), allocatable parwind::besttrackdata_t::technum

Definition at line 44 of file [parwind.F90](#).

19.9.2.38 thisstorm character(len=10) parwind::besttrackdata_t::thisstorm

Definition at line 36 of file [parwind.F90](#).

19.9.2.39 ty character(len=2), dimension(:), allocatable parwind::besttrackdata_t::ty

Definition at line 56 of file [parwind.F90](#).

19.9.2.40 windcode character(len=3), dimension(:), allocatable parwind::besttrackdata_t::windcode

Definition at line 76 of file [parwind.F90](#).

19.9.2.41 year integer, dimension(:), allocatable parwind::besttrackdata_t::year

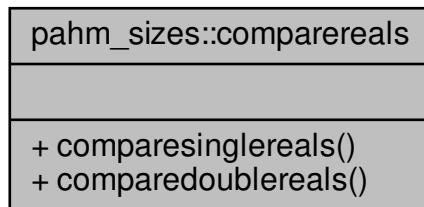
Definition at line 113 of file [parwind.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[parwind.F90](#)

19.10 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.

19.10.1 Detailed Description

Definition at line 22 of file [sizes.F90](#).

19.10.2 Member Function/Subroutine Documentation

```
19.10.2.1 comparedoublereals() integer function pahm_sizes::comparereals::comparedoublereals (
    real(sp), intent(in) rVal1,
    real(sp), intent(in) rVal2,
    real(sp), 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

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

Returns

myValOut

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

Note

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

Definition at line 101 of file [sizes.F90](#).

```
19.10.2.2 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

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

Returns

myValOut

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

Note

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

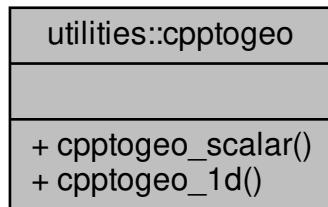
Definition at line 168 of file [sizes.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sizes.F90](#)

19.11 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.

19.11.1 Detailed Description

Definition at line 34 of file [utilities.F90](#).

19.11.2 Member Function/Subroutine Documentation

```
19.11.2.1 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

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

Definition at line 1947 of file [utilities.F90](#).

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

```
19.11.2.2 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

in	x	X coordinate: x (m) - real, scalar
----	---	------------------------------------

Parameters

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

Definition at line 1900 of file [utilities.F90](#).

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

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

- /home/takis/CSDL/parwinds-doc/src/[utilities.F90](#)

19.12 csv_module::csv_file Type Reference

Collaboration diagram for csv_module::csv_file:

csv_module::csv_file
+ quote + 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
+ 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` = ""
- 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'

19.12.1 Detailed Description

Definition at line 45 of file `csv_module.F90`.

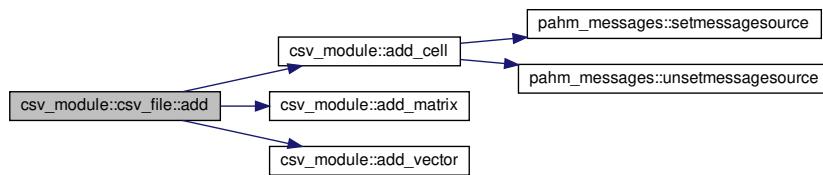
19.12.2 Member Function/Subroutine Documentation

19.12.2.1 add() generic, public csv_module::csv_file::add

Definition at line 115 of file [csv_module.F90](#).

References [csv_module::add_cell\(\)](#), [csv_module::add_matrix\(\)](#), and [csv_module::add_vector\(\)](#).

Here is the call graph for this function:

**19.12.2.2 add_cell()** procedure csv_module::csv_file::add_cell

Definition at line 118 of file [csv_module.F90](#).

19.12.2.3 add_matrix() procedure csv_module::csv_file::add_matrix

Definition at line 120 of file [csv_module.F90](#).

19.12.2.4 add_vector() procedure csv_module::csv_file::add_vector

Definition at line 119 of file [csv_module.F90](#).

19.12.2.5 close() procedure, public csv_module::csv_file::close

Definition at line 123 of file [csv_module.F90](#).

19.12.2.6 csv_get_value() procedure csv_module::csv_file::csv_get_value

Definition at line 106 of file [csv_module.F90](#).

19.12.2.7 destroy() procedure, public csv_module::csv_file::destroy

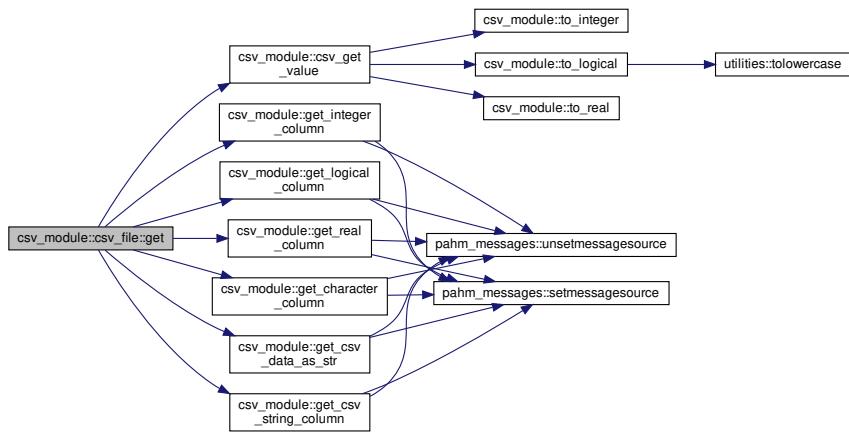
Definition at line 87 of file [csv_module.F90](#).

19.12.2.8 get() generic, public csv_module::csv_file::get

Definition at line 98 of file [csv_module.F90](#).

References [csv_module::csv_get_value\(\)](#), [csv_module::get_character_column\(\)](#), [csv_module::get_csv_data_as_str\(\)](#), [csv_module::get_csv_string_column\(\)](#), [csv_module::get_integer_column\(\)](#), [csv_module::get_logical_column\(\)](#), and [csv_module::get_real_column\(\)](#).

Here is the call graph for this function:

**19.12.2.9 get_character_column()** procedure csv_module::csv_file::get_character_column

Definition at line 110 of file [csv_module.F90](#).

19.12.2.10 get_column() procedure csv_module::csv_file::get_column

Definition at line 127 of file [csv_module.F90](#).

19.12.2.11 `get_csv_data_as_str()` procedure `csv_module::csv_file::get_csv_data_as_str`

Definition at line 105 of file [csv_module.F90](#).

19.12.2.12 `get_csv_string_column()` procedure `csv_module::csv_file::get_csv_string_column`

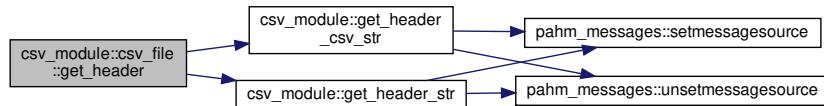
Definition at line 111 of file [csv_module.F90](#).

19.12.2.13 `get_header()` generic, public `csv_module::csv_file::get_header`

Definition at line 91 of file [csv_module.F90](#).

References [csv_module::get_header_csv_str\(\)](#), and [csv_module::get_header_str\(\)](#).

Here is the call graph for this function:

**19.12.2.14 `get_header_csv_str()`** procedure `csv_module::csv_file::get_header_csv_str`

Definition at line 94 of file [csv_module.F90](#).

19.12.2.15 `get_header_str()` procedure `csv_module::csv_file::get_header_str`

Definition at line 93 of file [csv_module.F90](#).

19.12.2.16 `get_integer_column()` procedure `csv_module::csv_file::get_integer_column`

Definition at line 108 of file [csv_module.F90](#).

19.12.2.17 get_logical_column() procedure csv_module::csv_file::get_logical_column

Definition at line 109 of file [csv_module.F90](#).

19.12.2.18 get_real_column() procedure csv_module::csv_file::get_real_column

Definition at line 107 of file [csv_module.F90](#).

19.12.2.19 initialize() procedure, public csv_module::csv_file::initialize

Definition at line 85 of file [csv_module.F90](#).

19.12.2.20 next_row() procedure, public csv_module::csv_file::next_row

Definition at line 122 of file [csv_module.F90](#).

19.12.2.21 open() procedure, public csv_module::csv_file::open

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

19.12.2.22 read() procedure, public csv_module::csv_file::read

Definition at line 86 of file [csv_module.F90](#).

19.12.2.23 read_line_from_file() procedure csv_module::csv_file::read_line_from_file

Definition at line 126 of file [csv_module.F90](#).

19.12.2.24 tokenize() procedure csv_module::csv_file::tokenize

Definition at line 125 of file [csv_module.F90](#).

19.12.2.25 variable_types() procedure, public csv_module::csv_file::variable_types

Definition at line 89 of file [csv_module.F90](#).

19.12.3 Member Data Documentation

19.12.3.1 chunk_size integer csv_module::csv_file::chunk_size = 100

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

19.12.3.2 csv_data type(csv_string), dimension(:, :), allocatable csv_module::csv_file::csv_data

Definition at line 63 of file [csv_module.F90](#).

19.12.3.3 delimiter character(len=1) csv_module::csv_file::delimiter = ','

Definition at line 56 of file [csv_module.F90](#).

19.12.3.4 enclose_all_in_quotes logical csv_module::csv_file::enclose_all_in_quotes = .false.

Definition at line 70 of file [csv_module.F90](#).

19.12.3.5 enclose_strings_in_quotes logical csv_module::csv_file::enclose_strings_in_quotes = .true.

Definition at line 68 of file [csv_module.F90](#).

19.12.3.6 header type(csv_string), dimension(:), allocatable csv_module::csv_file::header

Definition at line 62 of file [csv_module.F90](#).

19.12.3.7 icol integer csv_module::csv_file::icol = 0

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

19.12.3.8 iunit integer csv_module::csv_file::iunit = LUN_BTRK

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

19.12.3.9 logical_false_string character(len=1) csv_module::csv_file::logical_false_string = 'F'

Definition at line 76 of file [csv_module.F90](#).

19.12.3.10 logical_true_string character(len=1) csv_module::csv_file::logical_true_string = 'T'

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

19.12.3.11 n_cols integer, public csv_module::csv_file::n_cols = 0

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

19.12.3.12 n_rows integer, public csv_module::csv_file::n_rows = 0

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

19.12.3.13 quote character(len=1) csv_module::csv_file::quote = ''''

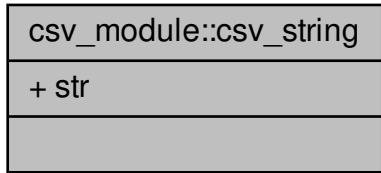
Definition at line 55 of file [csv_module.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[csv_module.F90](#)

19.13 csv_module::csv_string Type Reference

Collaboration diagram for csv_module::csv_string:



Public Attributes

- character(len=::), allocatable str

19.13.1 Detailed Description

Definition at line 37 of file [csv_module.F90](#).

19.13.2 Member Data Documentation

19.13.2.1 str character(len=::), allocatable csv_module::csv_string::str

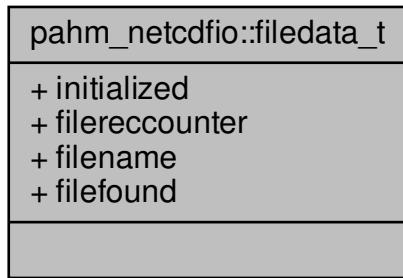
Definition at line 42 of file [csv_module.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[csv_module.F90](#)

19.14 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.

19.14.1 Detailed Description

Definition at line 39 of file [netcdfio.F90](#).

19.14.2 Member Data Documentation

19.14.2.1 filefound logical pahm_netcdfio::filedata_t::filefound = .FALSE.

Definition at line 43 of file [netcdfio.F90](#).

19.14.2.2 filename character(len=fnamelen) pahm_netcdfio::filedata_t::filename

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

19.14.2.3 filereccounter integer pahm_ncdfio::filedata_t::filereccounter = 0

Definition at line 41 of file [netcdfio.F90](#).

19.14.2.4 initialized logical pahm_ncdfio::filedata_t::initialized = .FALSE.

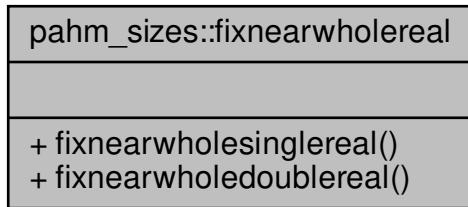
Definition at line 40 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

19.15 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.

19.15.1 Detailed Description

Definition at line 27 of file [sizes.F90](#).

19.15.2 Member Function/Subroutine Documentation

```
19.15.2.1 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

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

Returns

`myValOut` : Either **rVal** or its nearest integer **iVar** converted to double
`rVal` (if `abs(rVal - iVal) > eps`
`iVal` (if `abs(rVal - iVal) <= eps`

Definition at line 235 of file `sizes.F90`.

```
19.15.2.2 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

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

Returns

`myValOut` : Either **rVal** or its nearest integer **iVar** converted to real
`rVal` (if `abs(rVal - iVal) > eps`
`iVal` (if `abs(rVal - iVal) <= eps`

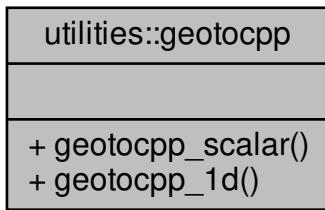
Definition at line 291 of file [sizes.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sizes.F90](#)

19.16 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.

19.16.1 Detailed Description

Definition at line 29 of file [utilities.F90](#).

19.16.2 Member Function/Subroutine Documentation

```
19.16.2.1 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

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

Definition at line 1854 of file [utilities.F90](#).

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

19.16.2.2 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

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

Definition at line 1807 of file [utilities.F90](#).

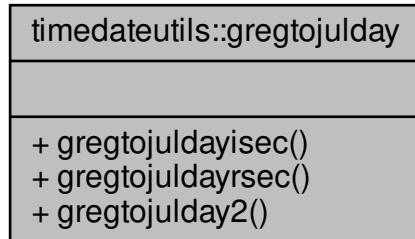
References [pahm_global::deg2rad](#), and [pahm_global::rearth](#).

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

- /home/takis/CSDL/parwinds-doc/src/[utilities.F90](#)

19.17 `timedateutils::gregtojulday` Interface Reference

Collaboration diagram for `timedateutils::gregtojulday`:



Public Member Functions

- `real(sz)` function `gregtojuldayisec` (`iYear, iMonth, iDay, iHour, iMin, iSec, mJD`)
Determines the Julian date from a Gregorian date.
- `real(sz)` function `gregtojuldayrsec` (`iYear, iMonth, iDay, iHour, iMin, rSec, mJD`)
Determines the Julian date from a Gregorian date.
- `real(sz)` function `gregtojulday2` (`iDate, iTime, mJD`)
Determines the Julian date from a Gregorian date.

19.17.1 Detailed Description

Definition at line 31 of file `timedateutils.F90`.

19.17.2 Member Function/Subroutine Documentation

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

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.

The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is useful to compute differences between dates.

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

Parameters

Returns

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

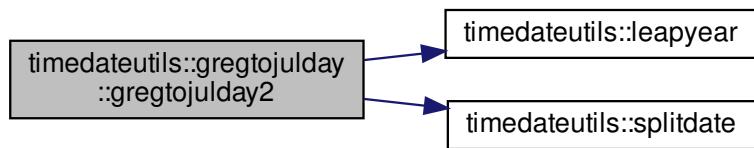
Note

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

Definition at line 775 of file `timedateutils.F90`.

References `timedateutils::firstgregdate`, `timedateutils::leapyear()`, `timedateutils::mdjoffset`, `timedateutils::splitdate()`, and `timedateutils::usemodjulday`.

Here is the call graph for this function:



```
19.17.2.2 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 useful to compute differences between dates.

Parameters

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

Returns

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

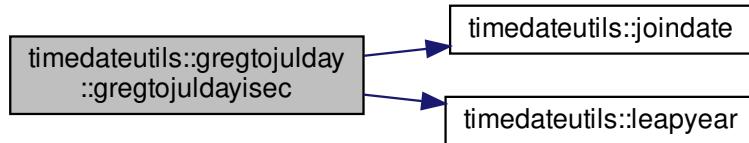
Note

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

Definition at line 536 of file [timedateutils.F90](#).

References [timedateutils::firstgregdate](#), [pahm_sizes::hp](#), [timedateutils::joindate\(\)](#), [timedateutils::leapyear\(\)](#), [timedateutils::mdjoffset](#), [pahm_sizes::rmissv](#), and [timedateutils::usemodjulday](#).

Here is the call graph for this function:



19.17.2.3 gregtojuldaysec() `real(sz) function timedateutils::gregtojulday::gregtojuldaysec (`

<code>integer, intent(in) iYear,</code>	
<code>integer, intent(in) iMonth,</code>	
<code>integer, intent(in) iDay,</code>	
<code>integer, intent(in) iHour,</code>	
<code>integer, intent(in) iMin,</code>	
<code>real(sz), intent(in) rSec,</code>	
<code>integer, intent(in), optional mJD)</code>	

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

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

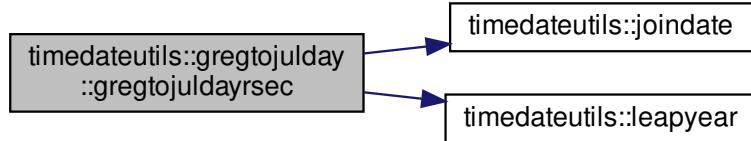
Note

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

Definition at line 654 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:

- /home/takis/CSDL/parwinds-doc/src/[timedateutils.F90](#)

19.18 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**

19.18.1 Detailed Description

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

19.18.2 Member Data Documentation

19.18.2.1 basin character(len=2), dimension(:), allocatable `parwind::hollanddata_t::basin`

Definition at line 130 of file [parwind.F90](#).

19.18.2.2 casttime real(sz), dimension(:), allocatable `parwind::hollanddata_t::casttime`

Definition at line 134 of file [parwind.F90](#).

19.18.2.3 casttype character(len=4), dimension(:), allocatable `parwind::hollanddata_t::casttype`

Definition at line 135 of file [parwind.F90](#).

19.18.2.4 cprdt real(sz), dimension(:), allocatable `parwind::hollanddata_t::cprdt`

Definition at line 153 of file [parwind.F90](#).

19.18.2.5 `cpress` `real(sz), dimension(:), allocatable parwind::hollanddata_t::cpress`

Definition at line 145 of file [parwind.F90](#).

19.18.2.6 `day` `integer, dimension(:), allocatable parwind::hollanddata_t::day`

Definition at line 133 of file [parwind.F90](#).

19.18.2.7 `dtg` `character(len=10), dimension(:), allocatable parwind::hollanddata_t::dtg`

Definition at line 132 of file [parwind.F90](#).

19.18.2.8 `fcstinc` `integer, dimension(:), allocatable parwind::hollanddata_t::fcstinc`

Definition at line 136 of file [parwind.F90](#).

19.18.2.9 `filename` `character(len=fnamelen) parwind::hollanddata_t::filename`

Definition at line 125 of file [parwind.F90](#).

19.18.2.10 `hour` `integer, dimension(:), allocatable parwind::hollanddata_t::hour`

Definition at line 133 of file [parwind.F90](#).

19.18.2.11 `icpress` `integer, dimension(:), allocatable parwind::hollanddata_t::icpress`

Definition at line 144 of file [parwind.F90](#).

19.18.2.12 `ilat` `integer, dimension(:), allocatable parwind::hollanddata_t::ilat`

Definition at line 138 of file [parwind.F90](#).

19.18.2.13 ilon integer, dimension(:), allocatable parwind::hollanddata_t::ilon

Definition at line 138 of file [parwind.F90](#).

19.18.2.14 irmw integer, dimension(:), allocatable parwind::hollanddata_t::irmw

Definition at line 150 of file [parwind.F90](#).

19.18.2.15 irrp integer, dimension(:), allocatable parwind::hollanddata_t::irrp

Definition at line 147 of file [parwind.F90](#).

19.18.2.16 ispeed integer, dimension(:), allocatable parwind::hollanddata_t::ispeed

Definition at line 141 of file [parwind.F90](#).

19.18.2.17 lat real(sz), dimension(:), allocatable parwind::hollanddata_t::lat

Definition at line 139 of file [parwind.F90](#).

19.18.2.18 loaded logical parwind::hollanddata_t::loaded = .FALSE.

Definition at line 127 of file [parwind.F90](#).

19.18.2.19 lon real(sz), dimension(:), allocatable parwind::hollanddata_t::lon

Definition at line 139 of file [parwind.F90](#).

19.18.2.20 month integer, dimension(:), allocatable parwind::hollanddata_t::month

Definition at line 133 of file [parwind.F90](#).

19.18.2.21 numrec integer `parwind::hollanddata_t::numrec`

Definition at line 128 of file [parwind.F90](#).

19.18.2.22 rmw real(sz), dimension(:), allocatable `parwind::hollanddata_t::rmw`

Definition at line 151 of file [parwind.F90](#).

19.18.2.23 rrp real(sz), dimension(:), allocatable `parwind::hollanddata_t::rrp`

Definition at line 148 of file [parwind.F90](#).

19.18.2.24 speed real(sz), dimension(:), allocatable `parwind::hollanddata_t::speed`

Definition at line 142 of file [parwind.F90](#).

19.18.2.25 stormnumber integer, dimension(:), allocatable `parwind::hollanddata_t::stormnumber`

Definition at line 131 of file [parwind.F90](#).

19.18.2.26 thisstorm character(len=10) `parwind::hollanddata_t::thisstorm`

Definition at line 126 of file [parwind.F90](#).

19.18.2.27 trvx real(sz), dimension(:), allocatable `parwind::hollanddata_t::trvx`

Definition at line 154 of file [parwind.F90](#).

19.18.2.28 trvy real(sz), dimension(:), allocatable `parwind::hollanddata_t::trvy`

Definition at line 154 of file [parwind.F90](#).

19.18.2.29 year integer, dimension(:), allocatable parwind::hollanddata_t::year

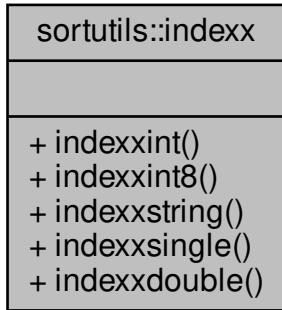
Definition at line 133 of file [parwind.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[parwind.F90](#)

19.19 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.

19.19.1 Detailed Description

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

19.19.2 Member Function/Subroutine Documentation

```
19.19.2.1 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

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

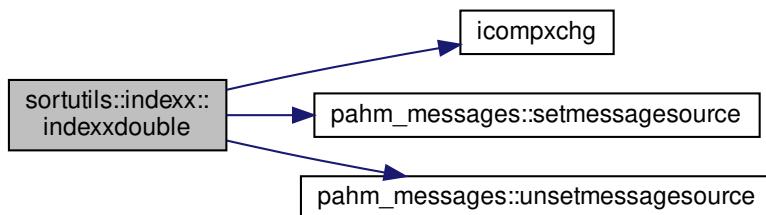
Note

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:



```
19.19.2.2 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

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

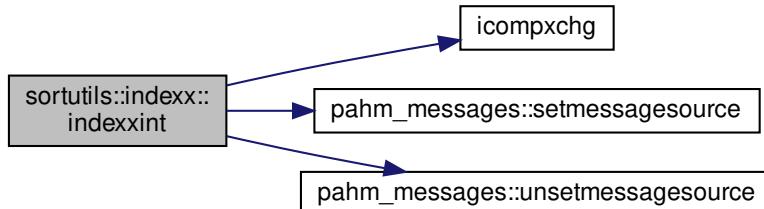
Note

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:



```
19.19.2.3 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

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

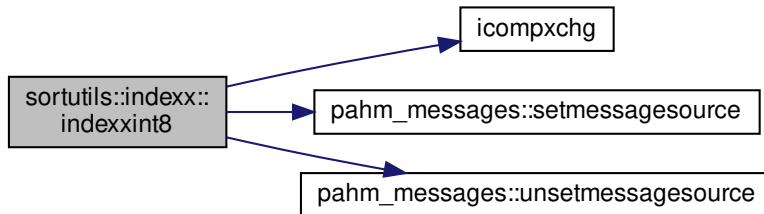
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 257 of file [sortutils.F90](#).

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

Here is the call graph for this function:



19.19.2.4 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

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

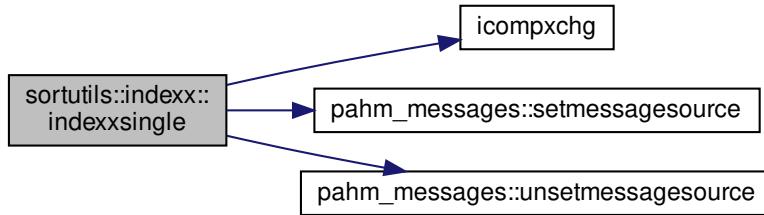
Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 607 of file [sortutils.F90](#).

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

Here is the call graph for this function:



19.19.2.5 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

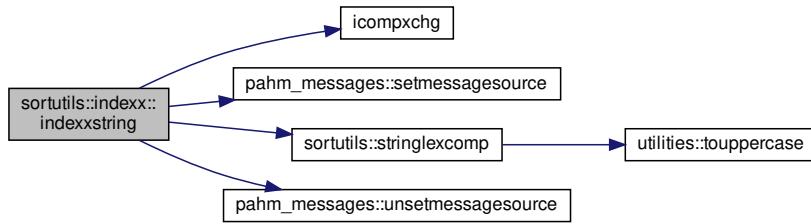
Parameters

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

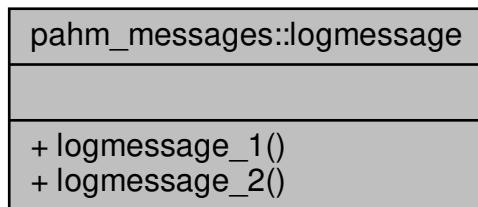


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

19.20 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)

19.20.1 Detailed Description

Definition at line 49 of file [messages.F90](#).

19.20.2 Member Function/Subroutine Documentation

19.20.2.1 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.

Parameters

in	message	The message to display
----	---------	------------------------

Definition at line 251 of file [messages.F90](#).

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

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

Definition at line 275 of file [messages.F90](#).

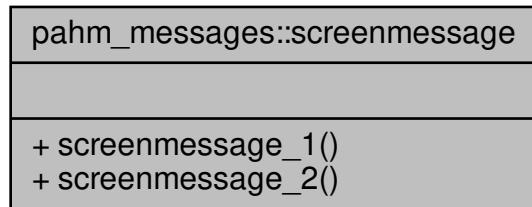
References [pahm_messages::logfileopened](#), [pahm_messages::loginitcalled](#), [pahm_messages::loglevelname](#)s, [pahm_global::lun_log](#), [pahm_messages::messagesources](#), and [pahm_messages::sourcenumber](#).

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

- /home/takis/CSDL/parwinds-doc/src/[messages.F90](#)

19.21 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)

19.21.1 Detailed Description

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

19.21.2 Member Function/Subroutine Documentation

19.21.2.1 [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.

Parameters

in	<i>message</i>	The message to display
----	----------------	------------------------

Definition at line 180 of file [messages.F90](#).

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

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

Definition at line 204 of file [messages.F90](#).

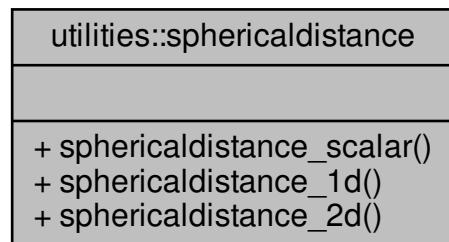
References [pahm_messages::loginitcalled](#), [pahm_messages::loglevelname](#)s, [pahm_global::lun_screen](#), [pahm_messages::messagesources](#), [pahm_messages::nscreen](#), and [pahm_messages::sourcenumber](#).

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

- /home/takis/CSDL/parwinds-doc/src/[messages.F90](#)

19.22 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.

19.22.1 Detailed Description

Definition at line 39 of file `utilities.F90`.

19.22.2 Member Function/Subroutine Documentation

```
19.22.2.1 sphericaldistance_1d() real(sz) function, dimension(:), allocatable utilities::sphericaldistance<-
::sphericaldistance_1d (
    real(sz), dimension(:), intent(in) lats,
    real(sz), dimension(:), intent(in) lons,
    real(sz), intent(in) lat0,
    real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas
https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".

Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points (Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

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

Returns

myValOut: The great-circle distance in meters, 1D array

Definition at line 2068 of file [utilities.F90](#).

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

```
19.22.2.2 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 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

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

Returns

myValOut: The great-circle distance in meters, 2D array

Definition at line 2167 of file [utilities.F90](#).

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

```
19.22.2.3 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

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

Returns

myValOut: The great-circle distance in meters

Definition at line 2000 of file [utilities.F90](#).

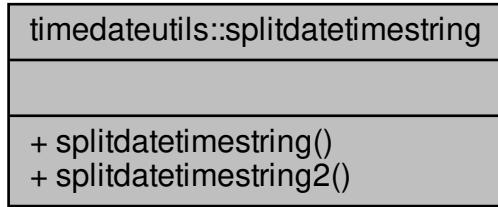
References [pahm_global::deg2rad](#), and [pahm_global::rearth](#).

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

- /home/takis/CSDL/parwinds-doc/src/[utilities.F90](#)

19.23 `timedateutils::splitdatetimestring` Interface Reference

Collaboration diagram for `timedateutils::splitdatetimestring`:



Public Member Functions

- subroutine `splitdatetimestring` (`inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec`)
Splits a date string into components.
- subroutine `splitdatetimestring2` (`inDateTime, iDate, iTime`)
Splits a date string into two components.

19.23.1 Detailed Description

Definition at line 37 of file [timedateutils.F90](#).

19.23.2 Constructor & Destructor Documentation

19.23.2.1 `splitdatetimestring()` subroutine `timedateutils::splitdatetimestring::splitdatetimestring` (

```
character(len=*) , intent(in)  inDateTime,
integer, intent(out)  iYear,
integer, intent(out)  iMonth,
integer, intent(out)  iDay,
integer, intent(out)  iHour,
integer, intent(out)  iMin,
integer, intent(out)  iSec )
```

Splits a date string into components.

This subroutine splits the string `inDate` (YYYYMMDDhhmmss) in six integers that is, "iYear (YYYY)", "iMonth (MM)", "iDay (DD)", "iHour (hh)", "iMin (mm)" and "iSec (ss)".

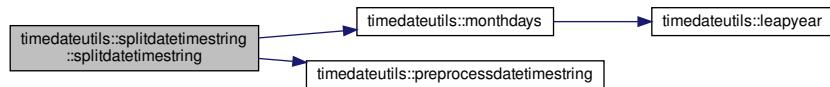
Parameters

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

Definition at line 1072 of file [timedateutils.F90](#).

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

Here is the call graph for this function:

**19.23.3 Member Function/Subroutine Documentation**

19.23.3.1 `splittimetimestring2()` subroutine `timedateutils::splittimetimestring::splittimetimestring2`
 (

```

    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)".

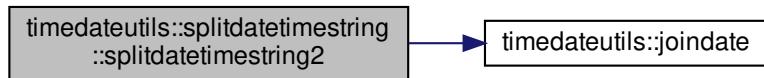
Parameters

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

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

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

Here is the call graph for this function:

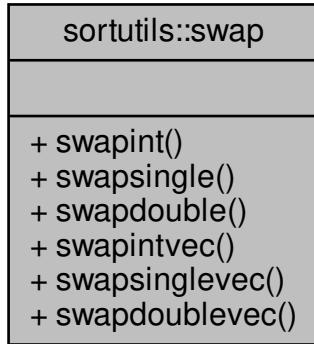


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[timedateutils.F90](#)

19.24 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).
- subroutine [swapsingle](#) (a, b, mask)
Swaps the contents of a and b (single precision).
- subroutine [swapdouble](#) (a, b, mask)

- subroutine [swapintvec](#) (a, b, mask)
Swaps the contents of a and b (integer).
- subroutine [swapsinglevec](#) (a, b, mask)
Swaps the contents of a and b (single precision).
- subroutine [swapdoublevec](#) (a, b, mask)
Swaps the contents of a and b (double precision).

19.24.1 Detailed Description

Definition at line 50 of file [sortutils.F90](#).

19.24.2 Member Function/Subroutine Documentation

19.24.2.1 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1607 of file [sortutils.F90](#).

```
19.24.2.2 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1760 of file [sortutils.F90](#).

```
19.24.2.3 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1505 of file [sortutils.F90](#).

```
19.24.2.4 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1658 of file [sortutils.F90](#).

```
19.24.2.5 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

Definition at line 1556 of file [sortutils.F90](#).

```
19.24.2.6 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

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

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

Note

Adopted from Numerical Recipes for Fortran 90

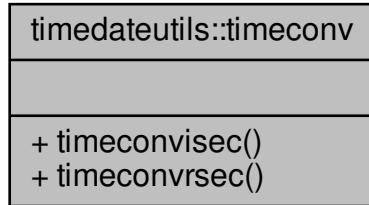
Definition at line 1709 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

19.25 `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.

19.25.1 Detailed Description

Definition at line 26 of file [timedateutils.F90](#).

19.25.2 Member Function/Subroutine Documentation

```
19.25.2.1 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.

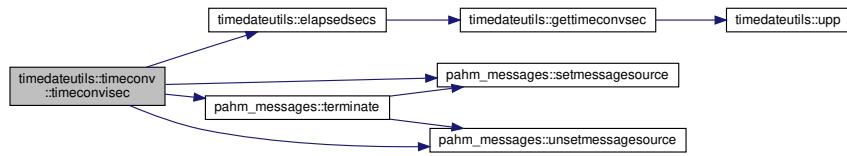
Parameters

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



19.25.2.2 **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

in	<i>iYear</i>	The year (integer)
in	<i>iMonth</i>	The month of the year (1-12, integer)

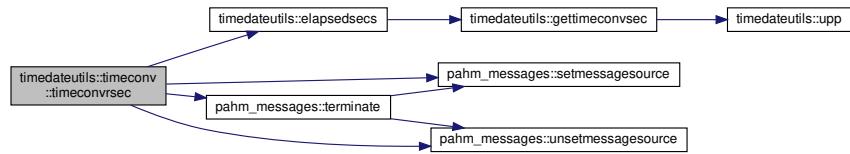
Parameters

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

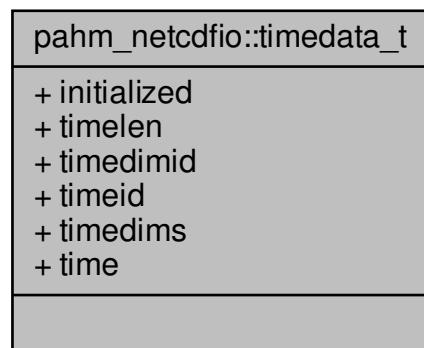


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[timedateutils.F90](#)

19.26 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**

19.26.1 Detailed Description

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

19.26.2 Member Data Documentation

19.26.2.1 initialized logical pahm_ncdfio::timedata_t::initialized = .FALSE.

Definition at line 47 of file [netcdfio.F90](#).

19.26.2.2 time real(sz), dimension(:,), allocatable pahm_ncdfio::timedata_t::time

Definition at line 52 of file [netcdfio.F90](#).

19.26.2.3 timedimid integer pahm_ncdfio::timedata_t::timedimid

Definition at line 49 of file [netcdfio.F90](#).

19.26.2.4 timedims integer, dimension(1) pahm_ncdfio::timedata_t::timedims

Definition at line 51 of file [netcdfio.F90](#).

19.26.2.5 timeid integer pahm_ncdfio::timedata_t::timeid

Definition at line 50 of file [netcdfio.F90](#).

19.26.2.6 timelen integer pahm_ncdfio::timedata_t::timelen = 1

Definition at line 48 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

20 File Documentation

20.1 user-guide/abstract.md File Reference

20.2 user-guide/application.md File Reference

20.3 user-guide/code.md File Reference

20.4 user-guide/credits.md File Reference

20.5 user-guide/deliverables.md File Reference

20.6 user-guide/evaluation.md File Reference

20.7 user-guide/features.md File Reference

20.8 user-guide/figures.md File Reference

20.9 user-guide/glossary.md File Reference

20.10 user-guide/intro.md File Reference

20.11 user-guide/models.md File Reference

20.12 user-guide/pahm_manual.md File Reference

20.13 user-guide/references-dox.md File Reference

20.14 user-guide/tables.md File Reference

20.15 /home/takis/CSDL/parwinds-doc/src/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)]].
- subroutine `csv_module::destroy_csv_file` (me)
Destroy a [[csv_file(type)]].
- subroutine `csv_module::read_csv_file` (me, filename, header_row, skip_rows, status_ok)
Reads a CSV file.
- subroutine `csv_module::open_csv_file` (me, filename, n_cols, status_ok, append)
Open a CSV file for writing.
- subroutine `csv_module::close_csv_file` (me, status_ok)
Close a CSV file after writing.
- subroutine `csv_module::add_cell` (me, val, int_fmt, real_fmt, trim_str)
Adds a cell to a CSV file.
- subroutine `csv_module::add_vector` (me, val, int_fmt, real_fmt, trim_str)
Adds a vector to a CSV file.
- subroutine `csv_module::add_matrix` (me, val, int_fmt, real_fmt, trim_str)
Adds a matrix to a CSV file.
- subroutine `csv_module::next_row` (me)
Advance to the next row in the CSV file (write any blank cells that are necessary to finish the row).
- subroutine `csv_module::get_header_csv_str` (me, header, status_ok)
Returns the header as a 'type(csv_string)' array ('read' must have already been called to read the file).
- subroutine `csv_module::get_header_str` (me, header, status_ok)
Returns the header as a 'character(len=)' array.*
- subroutine `csv_module::get_csv_data_as_str` (me, csv_data, status_ok)
Returns a 'character(len=)' array containing the csv data ('read' must have already been called to read the file).*
- pure elemental subroutine `csv_module::to_real` (str, val, status_ok)
Converts a string to a 'real(wp)'.
- pure elemental subroutine `csv_module::to_integer` (str, val, status_ok)
Converts a string to a 'integer(ip)'.
- pure elemental subroutine `csv_module::to_logical` (str, val, status_ok)
Converts a string to a 'logical'.
- subroutine `csv_module::variable_types` (me, itypes, status_ok)
Returns an array indicating the variable type of each columns.
- subroutine `csv_module::infer_variable_type` (str, itype)
Infers the variable type.
- subroutine `csv_module::csv_get_value` (me, row, col, val, status_ok)

- Get an individual value from the 'csv_data' structure in the CSV class.*
- subroutine `csv_module::get_column` (me, icol, r, status_ok)
Return a column from a CSV file vector.
 - subroutine `csv_module::get_real_column` (me, icol, r, status_ok)
Return a column from a CSV file as a 'real(wp)' vector.
 - subroutine `csv_module::get_integer_column` (me, icol, r, status_ok)
Return a column from a CSV file as a 'integer(ip)' vector.
 - subroutine `csv_module::get_logical_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'logical' vector.
 - subroutine `csv_module::get_character_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'character(len=)' vector.*
 - subroutine `csv_module::get_csv_string_column` (me, icol, r, status_ok)
Convert a column from a 'csv_string' matrix to a 'type(csv_string)' vector.
 - subroutine `csv_module::tokenize_csv_line` (me, line, cells)
Tokenize a line from a CSV file.
 - integer function `csv_module::number_of_lines_in_file` (iunit)
Returns the number of lines in a text file.
 - subroutine `csv_module::read_line_from_file` (me, iunit, line, status_ok)
Reads the next line from a file.
 - pure subroutine `csv_module::split` (str, token, chunk_size, vals)
Splits a character string using a token.

Variables

- integer, parameter, public `csv_module::csv_type_string` = 1
- integer, parameter, public `csv_module::csv_type_double` = 2
- integer, parameter, public `csv_module::csv_type_integer` = 3
- integer, parameter, public `csv_module::csv_type_logical` = 4
- real(wp), parameter `csv_module::zero` = 0.0_wp

20.15.1 Detailed Description

For reading and writing CSV files.

Author

Jacob Williams

Copyright

License BSD

Definition in file `csv_module.F90`.

20.16 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_btrkl  
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 ! a character string cell  
00031 INTEGER, PARAMETER, PUBLIC :: csv_type_double = 2 ! a 'real(wp)' cell  
00032 INTEGER, PARAMETER, PUBLIC :: csv_type_integer = 3 ! an 'integer(ip)' cell  
00033 INTEGER, PARAMETER, PUBLIC :: csv_type_logical = 4 ! a logical cell  
00034  
00035 REAL(wp), PARAMETER :: zero = 0.0_wp  
00036  
00037 type,public :: csv_string  
00038     ! a cell from a CSV file.  
00039     !  
00040     ! This is used to store the data internally  
00041     ! in the [[csv_file]] class.  
00042     character(len=:),allocatable :: str  
00043 end type csv_string  
00044  
00045 type,public :: csv_file  
00046  
00047     ! the main class for reading and writing CSV files.  
00048     !  
00049     ! @note A CSV file is assumed to contain the same number  
00050     !       of columns in each row. It may optionally contain  
00051     !       a header row.  
00052  
00053 private  
00054  
00055 character(len=1) :: quote      = '"' ! quotation character  
00056 character(len=1) :: delimiter = ',' ! delimiter character  
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
00096      ! For getting data from the class
00097      ! after the file has been read.
00098      generic,public :: get => get_csv_data_as_str,&
00099                           csv_get_value,&
00100                           get_real_column,&
00101                           get_integer_column,&
00102                           get_logical_column,&
00103                           get_character_column,&
00104                           get_csv_string_column
00105      procedure :: get_csv_data_as_str
00106      procedure :: csv_get_value
00107      procedure :: get_real_column
00108      procedure :: get_integer_column
00109      procedure :: get_logical_column
00110      procedure :: get_character_column
00111      procedure :: get_csv_string_column
00112
00113      procedure,public :: open => open_csv_file
00114
00115      generic,public :: add => add_cell,&
00116                           add_vector,&
00117                           add_matrix
00118      procedure :: add_cell
00119      procedure :: add_vector
00120      procedure :: add_matrix
00121
00122      procedure,public :: next_row
00123      procedure,public :: close => close_csv_file
00124
00125      procedure :: tokenize => tokenize_csv_line
00126      procedure :: read_line_from_file
00127      procedure :: get_column
00128
00129  end type csv_file
00130
00131  CONTAINS
00132
00133
00134
00135 !-----
00136 ! S U B R O U T I N E   I N I T I A L I Z E _ C S V _ F I L E
00137 !-----
00138 !-----
00139 !-----
```

```

00165
00166  subroutine initialize_csv_file(me,quote,delimiter,&
00167                           enclose_strings_in_quotes,&
00168                           enclose_all_in_quotes,&
00169                           logical_true_string,&
00170                           logical_false_string,&
00171                           chunk_size)
00172
00173  implicit none
00174
00175  class(csv_file),intent(out) :: me
00176  character(len=1),intent(in),optional :: quote           ! note: can only be one character
00177                                         ! (Default is '')
00178  character(len=1),intent(in),optional :: delimiter        ! note: can only be one character
00179                                         ! (Default is ',')
00180  logical,intent(in),optional :: enclose_strings_in_quotes ! if true, all string cells
00181                                         ! will be enclosed in quotes.
00182                                         ! (Default is True)
00183  logical,intent(in),optional :: enclose_all_in_quotes    ! if true, *all* cells will
00184                                         ! be enclosed in quotes.
00185                                         ! (Default is False)
00186  character(len=1),intent(in),optional :: logical_true_string ! when writing a logical 'true'
00187                                         ! value to a CSV file, this
00188                                         ! is the string to use
00189                                         ! (default is 'T')
00190  character(len=1),intent(in),optional :: logical_false_string ! when writing a logical 'false'
00191                                         ! value to a CSV file, this
00192                                         ! is the string to use
00193                                         ! (default is 'F')
00194  integer,intent(in),optional :: chunk_size ! factor for expanding vectors
```

```

00195                                     ! (default is 100)
00196
00197     if (present(quote)) me%quote = quote
00198     if (present(delimiter)) me%delimiter = delimiter
00199     if (present(enclose_strings_in_quotes)) &
00200         me%enclose_strings_in_quotes = enclose_strings_in_quotes
00201     if (present(enclose_all_in_quotes)) &
00202         me%enclose_all_in_quotes = enclose_all_in_quotes
00203     if (present(logical_true_string)) &
00204         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 ! S U B R O U T I N E   D E S T R O Y _ C S V _ F I L E
00217 !-----
00218 !-----
00219
00220     subroutine destroy_csv_file(me)
00221
00222     implicit none
00223
00224     class(csv_file),intent(out) :: me
00225
00226     end subroutine destroy_csv_file
00227 !=====
00228
00229 !-----
00230 ! S U B R O U T I N E   R E A D _ C S V _ F I L E
00231 !-----
00232 !-----
00233
00234     subroutine read_csv_file(me, filename, header_row, skip_rows, status_ok)
00235
00236     implicit none
00237
00238     !-----
00239 ! S U B R O U T I N E   R E A D _ C S V _ F I L E
00240 !-----
00241 !-----
00242
00243     character(len=*) intent(in) :: filename
00244     logical,intent(out) :: status_ok
00245     integer,intent(in),optional :: header_row
00246     integer,dimension(:),intent(in),optional :: skip_rows ! rows to skip
00247
00248     type(csv_string),dimension(:),allocatable :: row_data ! a tokenized row
00249     type(csv_string) :: empty_data
00250     integer,dimension(:),allocatable :: rows_to_skip ! the actual rows to skip
00251     character(len=:),allocatable :: line ! a line from the file
00252
00253     integer :: i           ! counter
00254     integer :: j           ! counter
00255     integer :: irow        ! row counter
00256     integer :: n_rows_in_file ! number of lines in the file
00257     integer :: n_rows       ! number of rows in the output data matrix
00258     integer :: n_cols       ! number of columns in the file (and output data matrix)
00259     integer :: istat        ! open status flag
00260     integer :: line_n_cols ! number of columns in the line (not necessarily equal to n_cols)
00261     integer :: iunit        ! open file unit
00262     logical :: arrays_allocated ! if the arrays in the
00263                               ! class have been allocated
00264     integer :: iheader      ! row number of header row
00265                               ! (0 if no header specified)
00266     character(len=1) :: tmp ! for skipping a row
00267
00268     empty_data%str = ' '
00269     iunit = lun_btrk
00270
00271     CALL setmessagesource("read_csv_file")
00272
00273     call me%destroy()
00274     arrays_allocated = .false.
00275
00276     CALL openfileforread(iunit, trim(adjustl(filename)), istat)
00277
00278     IF (istat /= 0) THEN
00279         WRITE(scratchmessage, '(a)') 'Error opening the file: ' // trim(adjustl(filename))
00280         CALL allmessage(error, scratchmessage)
00281
00282
00283
00284
00285
00286
00287
00288
00289
00290
00291
00292
00293
00294
00295
00296
00297
00298
00299
00300
00301

```

```

00302     CALL unsetmessagesource()
00303
00304     CALL terminate()
00305 ELSE
00306     WRITE(scratchmessage, '(a)') 'Processing the file: ' // trim(adjustl(filename))
00307     CALL logmessage(info, scratchmessage)
00308 END IF
00309
00310 ! if (istat==0) then
00311
00312     !get number of lines in the file
00313     n_rows_in_file = number_of_lines_in_file(iunit)
00314
00315     !get number of lines in the data array
00316     if (present(skip_rows)) then
00317         !get size of unique elements in skip_rows,
00318         !and subtract from n_rows_in_file
00319         rows_to_skip = unique(skip_rows,chunk_size=me%chunk_size)
00320         n_rows = n_rows_in_file - size(rows_to_skip)
00321     else
00322         n_rows = n_rows_in_file
00323     end if
00324     if (present(header_row)) then
00325         iheader = max(0,header_row)
00326         n_rows = n_rows - 1
00327     else
00328         iheader = 0
00329     end if
00330
00331     me%n_rows = n_rows
00332
00333     ! we don't know the number of columns
00334     ! until we parse the first row (or the header)
00335     ! Panagiotis Velissariou: some csv files do not have the same number
00336     ! of columns, so we need to determine the max number of columns
00337     ! for the allocation of the arrays
00338     !--- PV
00339     n_cols = 0
00340     do i=1,n_rows_in_file ! rows in the file
00341         call me%read_line_from_file(iunit,line,status_ok)
00342         call me%tokenize(line,row_data)
00343         n_cols = max(n_cols,size(row_data))
00344     end do
00345     rewind(iunit)
00346
00347     me%n_cols = n_cols
00348     allocate(me%csv_data(n_rows,n_cols))
00349     if (iheader/=0) allocate(me%header(n_cols))
00350     arrays_allocated = .true.
00351     !--- PV
00352
00353     !read each line in the file, parse it, and populate data
00354     irow = 0
00355     do i=1,n_rows_in_file ! rows in the file
00356
00357         ! skip row if necessary
00358         if (allocated(rows_to_skip)) then
00359             if (any(i==rows_to_skip)) then
00360                 read(iunit,fmt='(A1)',iostat=istat) tmp
00361                 if (istat/-0) then
00362                     scratchmessage = 'Error skipping row in file: '//trim(filename)
00363                     CALL allmessage(error, scratchmessage)
00364
00365                     close(unit=iunit,iostat=istat)
00366                     status_ok = .false.
00367
00368                     CALL unsetmessagesource()
00369                     return
00370                 end if
00371                 cycle
00372             end if
00373         end if
00374
00375         call me%read_line_from_file(iunit,line,status_ok)
00376         if (.not. status_ok) then
00377             CALL unsetmessagesource()
00378             return ! file read error
00379         end if
00380         call me%tokenize(line,row_data)
00381         line_n_cols = size(row_data)
00382

```

```

00383      if (i==iheader) then
00384          do j=1,me%n_cols
00385              me%header(j)%str = row_data(j)%str
00386          end do
00387      else
00388          irow = irow + 1 ! row counter in data array
00389          do j=1,n_cols
00390              if(j <= line_n_cols) then
00391                  me%csv_data(irow,j) = row_data(j) !%str
00392              else
00393                  me%csv_data(irow,j) = empty_data !%str
00394              end if
00395          end do
00396      end if
00397
00398  end do
00399
00400      ! close the file
00401      close(unit=iunit,iostat=istat)
00402
00403      status_ok = .true.
00404
00405 ! else
00406 !     scratchMessage = 'Error opening file: '//trim(filename)
00407 !     CALL AllMessage(ERROR, scratchMessage)
00408 !     status_ok = .false.
00409 ! end if
00410
00411 CALL unsetmessagesource()
00412
00413 end subroutine read_csv_file
00414 !=====
00415
00416 !-----
00417 ! S U B R O U T I N E   O P E N _ C S V _ F I L E
00418 !-----
00436 !
00437 subroutine open_csv_file(me, filename, n_cols, status_ok, append)
00438
00439 implicit none
00440
00441 class(csv_file),intent(inout) :: me
00442 character(len=*),intent(in) :: filename
00443 integer,intent(in) :: n_cols
00444 logical,intent(out) :: status_ok
00445 logical,intent(in),optional :: append
00446
00447 integer :: istat ! open 'iostat' flag
00448 logical :: append_flag ! local copy of 'append' argument
00449 logical :: file_exists ! if the file exists
00450
00451 CALL setmessagesource("open_csv_file")
00452
00453 call me%destroy()
00454
00455 me%n_cols = n_cols
00456
00457 ! optional append argument:
00458 append_flag = .false.
00459 file_exists = .false.
00460 if (present(append)) then
00461     append_flag = append
00462     if (append) inquire(file=filename, exist=file_exists)
00463 end if
00464
00465 if (append_flag .and. file_exists) then
00466     open(unit=me%iunit,file=filename,status='OLD',position='APPEND',iostat=istat)
00467 else
00468     open(unit=me%iunit,file=filename,status='REPLACE',iostat=istat)
00469 end if
00470
00471 if (istat==0) then
00472     status_ok = .true.
00473 else
00474     scratchmessage = 'Error opening file: '//trim(filename)
00475     CALL allmessage(error, scratchmessage)
00476     status_ok = .false.
00477 end if
00478
00479 CALL unsetmessagesource()
00480

```

```

00481   end subroutine open_csv_file
00482 !=====
00483
00484 !-----
00485 ! S U B R O U T I N E   C L O S E _ C S V _ F I L E
00486 !-----
00487 !-----
00498 !
00499 subroutine close_csv_file(me, status_ok)
00500
00501 implicit none
00502
00503 class(csv_file),intent(inout) :: me
00504 logical,intent(out) :: status_ok
00505
00506 integer :: istat ! close 'iostat' flag
00507
00508 close(me%iunit,iostat=istat)
00509 status_ok = istat==0
00510
00511 end subroutine close_csv_file
00512 !=====
00513
00514 !-----
00515 ! S U B R O U T I N E   A D D _ C E L L
00516 !-----
00536 !
00537 subroutine add_cell(me, val, int_fmt, real_fmt, trim_str)
00538
00539 implicit none
00540
00541 class(csv_file),intent(inout) :: me
00542 class(*),intent(in) :: val
00543 character(len=*),intent(in),optional :: int_fmt
00544 character(len=*),intent(in),optional :: real_fmt
00545 logical,intent(in),optional :: trim_str
00546
00547 integer :: istat ! write 'iostat' flag
00548 character(len=:),allocatable :: ifmt ! actual format string to use for integers
00549 character(len,:),allocatable :: rfmt ! actual format string to use for reals
00550 logical :: trimstr ! if the strings are to be trimmed
00551 character(len=max_real_str_len) :: real_val ! for writing a real value
00552 character(len=max_integer_str_len) :: int_val ! for writing an integer value
00553
00554 CALL setmessagesource("add_cell")
00555
00556 ! make sure the row isn't already finished
00557 if (me%icol<me%n_cols) then
00558
00559   me%icol = me%icol + 1
00560
00561   if (me%enclose_all_in_quotes) then
00562     write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me
00563   end if
00564
00565   select type (val)
00566   type is (integer(ip))
00567     if (present(int_fmt)) then
00568       ifmt = trim(adjustl(int_fmt))
00569     else
00570       ifmt = default_int_fmt
00571     end if
00572     write(int_val,fmt=ifmt,iostat=istat) val
00573     write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(int_val))
00574   type is (real(wp))
00575     if (present(real_fmt)) then
00576       rfmt = trim(adjustl(real_fmt))
00577     else
00578       rfmt = default_real_fmt
00579     end if
00580     write(real_val,fmt=rfmt,iostat=istat) val
00581     write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(adjustl(real_val))
00582   type is (logical)
00583     if (val) then
00584       write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_true_string
00585     else
00586       write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%logical_false_string
00587     end if
00588   type is (character(len=*))
00589     if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00590       write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me
00591     if (present(trim_str)) then

```

```

00592         trimstr = trim_str
00593     else
00594         trimstr = .false.
00595     end if
00596     if (trimstr) then
00597         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val)
00598     else
00599         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val
00600     end if
00601     if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00602         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00603 type is (csv_string)
00604     if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00605         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00606     if (present(trim_str)) then
00607         trimstr = trim_str
00608     else
00609         trimstr = .false.
00610     end if
00611     if (trimstr) then
00612         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) trim(val%str)
00613     else
00614         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) val%str
00615     end if
00616     if (me%enclose_strings_in_quotes .and. .not. me%enclose_all_in_quotes) &
00617         write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00618 class default
00619     scratchmessage = 'Error: cannot write unknown variable type to CSV file.'
00620     CALL allmessage(error, scratchmessage)
00621 end select
00622
00623 if (me%enclose_all_in_quotes) then
00624     write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%quote
00625 end if
00626 if (me%icol<me%n_cols) write(me%iunit,fmt='(A)',advance='NO',iostat=istat) me%delimiter
00627
00628 else
00629     scratchmessage = 'Error: cannot write more cells to the current row.'
00630     CALL allmessage(error, scratchmessage)
00631 end if
00632
00633 CALL unsetmessagesource()
00634
00635 end subroutine add_cell
00636 !=====
00637
00638 !-----+
00639 ! S U B R O U T I N E   A D D _ V E C T O R
00640 !-----+
00641 !
00642 !-----
00643 subroutine add_vector(me, val, int_fmt, real_fmt, trim_str)
00644
00645 implicit none
00646
00647 class(csv_file),intent(inout) :: me
00648 class(*),dimension(:),intent(in) :: val
00649 character(len=*) ,intent(in),optional :: int_fmt
00650 character(len=*) ,intent(in),optional :: real_fmt
00651
00652 logical,intent(in),optional :: trim_str
00653
00654 integer :: i ! counter
00655
00656 do i=1,size(val)
00657
00658 #if defined __GFORTRAN__
00659     ! This is a stupid workaround for gfortran bugs (tested with 7.2.0)
00660     select type (val)
00661     type is (character(len=*)) 
00662         call me%add(val(i),int_fmt,real_fmt,trim_str)
00663     class default
00664         call me%add(val(i),int_fmt,real_fmt,trim_str)
00665     end select
00666 #else
00667     call me%add(val(i),int_fmt,real_fmt,trim_str)
00668 #endif
00669
00670 end do
00671
00672 end subroutine add_vector
00673 !=====
```

```

00690
00691 !-----
00692 ! S U B R O U T I N E   A D D _ M A T R I X
00693 !-----
00713 !-----
00714 subroutine add_matrix(me, val, int_fmt, real_fmt, trim_str)
00715
00716     implicit none
00717
00718     class(csv_file),intent(inout) :: me
00719     class(*),dimension(:,:),intent(in) :: val
00720     character(len=*),intent(in),optional :: int_fmt
00721     character(len=*),intent(in),optional :: real_fmt
00722     logical,intent(in),optional :: trim_str
00723
00724     integer :: i ! counter
00725
00726     ! add each row:
00727     do i=1,size(val,1)
00728         call me%add(val(i,:),int_fmt,real_fmt,trim_str)
00729         call me%next_row()
00730     end do
00731
00732 end subroutine add_matrix
00733 !=====
00734
00735 !-----
00736 ! S U B R O U T I N E   N E X T _ R O W
00737 !-----
00748 !-----
00749 subroutine next_row(me)
00750
00751     implicit none
00752
00753     class(csv_file),intent(inout) :: me
00754
00755     integer :: i ! counter
00756     integer :: n ! number of blank cells to write
00757
00758     if (me%icol>0) then
00759         n = me%n_cols - me%icol
00760         do i=1,n
00761             if (i==n) then ! no trailing delimiter
00762                 if (me%enclose_strings_in_quotes) then
00763                     write(me%iunit,'(A)',advance='NO') me%quote//me%quote
00764                 end if
00765             else
00766                 if (me%enclose_strings_in_quotes) then
00767                     write(me%iunit,'(A)',advance='NO') me%quote//me%quote//me%delimiter
00768                 else
00769                     write(me%iunit,'(A)',advance='NO') me%delimiter
00770                 end if
00771             end if
00772         end do
00773         write(me%iunit,'(A)') " ! new line
00774     end if
00775
00776     me%icol = 0 ! this row is finished
00777
00778 end subroutine next_row
00779 !=====
00780
00781 !-----
00782 ! S U B R O U T I N E   G E T _ H E A D E R _ C S V _ S T R
00783 !-----
00798 !-----
00799 subroutine get_header_csv_str(me, header, status_ok)
00800
00801     implicit none
00802
00803     class(csv_file),intent(inout) :: me
00804     type(csv_string),dimension(:),allocatable,intent(out) :: header
00805     logical,intent(out) :: status_ok
00806
00807     integer :: i ! column counter
00808
00809     CALL setmessagesource("get_header_csv_str")
00810
00811     if (allocated(me%header)) then
00812
00813         allocate(header(me%n_cols))

```

```

00814      do i=1,me%n_cols
00815         header(i) = me%header(i)
00816     end do
00817     status_ok = .false.
00818
00819   else
00820     scratchmessage = 'Error: no header in class.'
00821     CALL allmessage(error, scratchmessage)
00822     status_ok = .false.
00823   end if
00824
00825   CALL unsetmessagesource()
00826
00827 end subroutine get_header_csv_str
00828 !=====
00829
00830 !-----+
00831 ! S U B R O U T I N E   G E T _ H E A D E R _ S T R
00832 !-----+
00847 !-----+
00848 subroutine get_header_str(me,header,status_ok)
00849
00850   implicit none
00851
00852   class(csv_file),intent(inout) :: me
00853   character(len=*),dimension(:,),allocatable,intent(out) :: header
00854   logical,intent(out) :: status_ok
00855
00856   integer :: i ! column counter
00857
00858   CALL setmessagesource("get_header_str")
00859
00860   if (allocated(me%header)) then
00861
00862     allocate(header(me%n_cols))
00863     do i=1,me%n_cols
00864       header(i) = me%header(i)%str
00865     end do
00866     status_ok = .false.
00867
00868   else
00869     scratchmessage = 'Error: no header in class.'
00870     CALL allmessage(error, scratchmessage)
00871     status_ok = .false.
00872   end if
00873
00874   CALL unsetmessagesource()
00875
00876 end subroutine get_header_str
00877 !=====
00878
00879 !-----+
00880 ! S U B R O U T I N E   G E T _ C S V _ D A T A _ A S _ S T R
00881 !-----+
00896 !-----+
00897 subroutine get_csv_data_as_str(me, csv_data, status_ok)
00898
00899   implicit none
00900
00901   class(csv_file),intent(inout) :: me
00902   character(len=*),dimension(:, :),allocatable,intent(out) :: csv_data
00903   logical,intent(out) :: status_ok
00904
00905   integer :: i ! row counter
00906   integer :: j ! column counter
00907
00908   CALL setmessagesource("get_csv_data_as_str")
00909
00910   if (allocated(me%csv_data)) then
00911     ! size the output array:
00912     allocate(csv_data(me%n_rows,me%n_cols))
00913     ! convert each element to a string:
00914     do concurrent(i=1:me%n_rows)
00915       do concurrent(j=1:me%n_cols)
00916         csv_data(i,j) = me%csv_data(i,j)%str
00917       end do
00918     end do
00919     status_ok = .true.
00920   else
00921     scratchmessage = 'Error: class has not been initialized'
00922     CALL allmessage(error, scratchmessage)

```

```

00923     status_ok = .false.
00924   end if
00925
00926   CALL unsetmessagesource()
00927
00928   end subroutine get_csv_data_as_str
00929 !=====
00930
00931 !-----
00932 ! S U B R O U T I N E   T O _ R E A L
00933 !-----
00947 !-----
00948 pure elemental subroutine to_real(str, val, status_ok)
00949
00950   implicit none
00951
00952   character(len=*),intent(in) :: str
00953   real(wp),intent(out) :: val
00954   logical,intent(out) :: status_ok
00955
00956   integer :: istat
00957
00958   read(str,fmt=*,iostat=istat) val
00959   if (istat==0) then
00960     status_ok = .true.
00961   else
00962     status_ok = .false.
00963     val = zero
00964   end if
00965
00966   end subroutine to_real
00967 !=====
00968
00969 !-----
00970 ! S U B R O U T I N E   T O _ I N T E G E R
00971 !-----
00985 !-----
00986 pure elemental subroutine to_integer(str, val, status_ok)
00987
00988   implicit none
00989
00990   character(len=*),intent(in) :: str
00991   integer(ip),intent(out) :: val
00992   logical,intent(out) :: status_ok
00993
00994   integer :: istat
00995
00996   read(str,fmt=default_int_fmt,iostat=istat) val
00997   if (istat==0) then
00998     status_ok = .true.
00999   else
01000     status_ok = .false.
01001     val = 0
01002   end if
01003
01004   end subroutine to_integer
01005 !=====
01006
01007 !-----
01008 ! S U B R O U T I N E   T O _ L O G I C A L
01009 !-----
01027 !-----
01028 pure elemental subroutine to_logical(str, val, status_ok)
01029
01030   implicit none
01031
01032   character(len=*),intent(in) :: str
01033   logical,intent(out) :: val
01034   logical,intent(out) :: status_ok
01035
01036   character(len=:),allocatable :: tmp
01037
01038   ! True and False options (all lowercase):
01039   character(len=4),dimension(4),parameter :: true_str = ['1      ',&
01040                                         't      ',&
01041                                         'true   ',&
01042                                         '.true.']
01043   character(len=4),dimension(4),parameter :: false_str = ['0      ',&
01044                                         'f      ',&
01045                                         'false  ',&
01046                                         '.false.]
```

```

01047
01048     tmp = tolowercase(str)
01049     if ( any(tmp==true_str) ) then
01050         val = .true.
01051         status_ok = .true.
01052     else if ( any(tmp==false_str) ) then
01053         val = .false.
01054         status_ok = .true.
01055     else
01056         val = .false.
01057         status_ok = .false.
01058     end if
01059
01060 end subroutine to_logical
01061 !=====
01062
01063 !-----
01064 ! SUBROUTINE VARIABLE_TYPES
01065 !-----
01066 !-----
01067
01068 subroutine variable_types(me, itypes, status_ok)
01069
01070     implicit none
01071
01072     class(csv_file),intent(inout) :: me
01073     integer,dimension(:),allocatable,intent(out) :: itypes
01074     logical,intent(out) :: status_ok
01075
01076     integer :: i ! counter
01077
01078     CALL setmessagesource("variable_types")
01079
01080     if (allocated(me%csv_data)) then
01081         allocate(itypes(me%n_cols))
01082         do i=1,me%n_cols
01083             call infer_variable_type(me%csv_data(1,i)%str,itypes(i))
01084         end do
01085     else
01086         scratchmessage = 'Error: class has not been initialized'
01087         CALL allmessage(error, scratchmessage)
01088         status_ok = .false.
01089     end if
01090
01091     CALL unsetmessagesource()
01092
01093 end subroutine variable_types
01094 !=====
01095
01096 !-----
01097 ! SUBROUTINE INFERN_VARIABLE_TYPE
01098 !-----
01099 !-----
01100
01101 subroutine infer_variable_type(str, itype)
01102
01103     implicit none
01104
01105     character(len=*),intent(in) :: str
01106     integer,intent(out) :: itype
01107
01108     real(wp) :: rval      ! a real value
01109     integer(ip) :: ival      ! an integer value
01110     logical :: lval      ! a logical value
01111     logical :: status_ok ! status flag
01112
01113     call to_integer(str,ival,status_ok)
01114     if (status_ok) then
01115         itype = csv_type_integer
01116         return
01117     end if
01118
01119     call to_real(str,rval,status_ok)
01120     if (status_ok) then
01121         itype = csv_type_double
01122         return
01123     end if
01124
01125     call to_logical(str,lval,status_ok)
01126     if (status_ok) then
01127         itype = csv_type_logical
01128         return
01129     end if
01130

```

```

01171      ! default is string:
01172      type = csv_type_string
01173
01174      end subroutine infer_variable_type
01175  =====
01176 !-----+
01177 !-----+
01178 !-----+
01179 ! S U B R O U T I N E   C S V _ G E T _ V A L U E
01180 !-----+
01181 !-----+
01182 subroutine csv_get_value(me, row, col, val, status_ok)
01183
01184     implicit none
01185
01186     class(csv_file),intent(inout) :: me
01187     integer,intent(in)    :: row
01188     integer,intent(in)    :: col
01189     class(*),intent(out)  :: val
01190     logical,intent(out)   :: status_ok
01191
01192     select type (val)
01193     type is (integer(ip))
01194         call to_integer(me%csv_data(row,col)%str,val,status_ok)
01195     type is (real(wp))
01196         call to_real(me%csv_data(row,col)%str,val,status_ok)
01197     type is (logical)
01198         call to_logical(me%csv_data(row,col)%str,val,status_ok)
01199     type is (character(len=*))
01200         status_ok = .true.
01201         if (allocated(me%csv_data(row,col)%str)) then
01202             val = me%csv_data(row,col)%str
01203         else
01204             val = ""
01205         end if
01206     type is (csv_string)
01207         status_ok = .true.
01208         val = me%csv_data(row,col)
01209     class default
01210         status_ok = .false.
01211     end select
01212
01213     end subroutine csv_get_value
01214  =====
01215 !-----+
01216 !-----+
01217 !-----+
01218 ! S U B R O U T I N E   G E T _ C O L U M N
01219 !-----+
01220 !-----+
01221 subroutine get_column(me, icol, r, status_ok)
01222
01223     implicit none
01224
01225     class(csv_file),intent(inout) :: me
01226     integer,intent(in)    :: icol
01227     class(*),dimension(:),intent(out) :: r
01228     logical,intent(out)   :: status_ok
01229
01230     integer :: i ! counter
01231 #if defined __GFORTRAN__
01232     character(len=:),allocatable :: tmp ! for gfortran workaround
01233 #endif
01234
01235     CALL setmessagesource("get_column")
01236
01237     ! we know the data is allocated, since that
01238     ! was checked by the calling routines.
01239
01240     if (me%n_cols>=icol .and. icol>0) then
01241
01242         do i=1,me%n_rows ! row loop
01243
01244 #if defined __GFORTRAN__
01245         ! the following is a workaround for gfortran bugs:
01246         select type (r)
01247         type is (character(len=*))
01248             tmp = repeat(' ',len(r)) ! size the string
01249             call me%csv_get_value(i,icol,tmp,status_ok)
01250             r(i) = tmp
01251         class default
01252             call me%csv_get_value(i,icol,r(i),status_ok)

```

```

01292      end select
01293 #else
01294   call me%csv_get_value(i,icol,r(i),status_ok)
01295 #endif
01296   if (.not. status_ok) then
01297     select type (r)
01298       ! note: character conversion can never fail, so not
01299       ! checking for that here. also we know it is real,
01300       ! integer, or logical at this point.
01301     type is (integer(ip))
01302       scratchmessage = 'Error converting string to integer: '//trim(me%csv_data(i,icol)%str)
01303       CALL allmessage(error, scratchmessage)
01304     r(i) = 0
01305     type is (real(wp))
01306       scratchmessage = 'Error converting string to real: '//trim(me%csv_data(i,icol)%str)
01307       CALL allmessage(error, scratchmessage)
01308     r(i) = zero
01309     type is (logical)
01310       scratchmessage = 'Error converting string to logical: '//trim(me%csv_data(i,icol)%str)
01311       CALL allmessage(error, scratchmessage)
01312     r(i) = .false.
01313   end select
01314 end if
01315
01316 end do
01317
01318 else
01319   WRITE(scratchmessage,'(A,1X,I5)') 'Error: invalid column number: ', icol
01320   CALL allmessage(error, scratchmessage)
01321   status_ok = .false.
01322 end if
01323
01324 CALL unsetmessagesource()
01325
01326 end subroutine get_column
01327 !=====
01328 !-----
01329 !-----S U B R O U T I N E   G E T _ R E A L _ C O L U M N
01330 !-----
01331 !-----
01347 !-----
01348 subroutine get_real_column(me, icol, r, status_ok)
01349
01350 implicit none
01351
01352 class(csv_file),intent(inout) :: me
01353 integer,intent(in) :: icol ! column number
01354 real(wp),dimension(:),allocatable,intent(out) :: r
01355 logical,intent(out) :: status_ok
01356
01357 CALL setmessagesource("get_real_column")
01358
01359 if (allocated(me%csv_data)) then
01360   allocate(r(me%n_rows)) ! size the output vector
01361   call me%get_column(icol,r,status_ok)
01362 else
01363   scratchmessage = 'Error: class has not been initialized'
01364   CALL allmessage(error, scratchmessage)
01365   status_ok = .false.
01366 end if
01367
01368 CALL unsetmessagesource()
01369
01370 end subroutine get_real_column
01371 !=====
01372 !-----
01373 !-----S U B R O U T I N E   G E T _ I N T E G E R _ C O L U M N
01374 !-----
01375 !-----
01391 !-----
01392 subroutine get_integer_column(me,icols,r,status_ok)
01393
01394 implicit none
01395
01396 class(csv_file),intent(inout) :: me
01397 integer,intent(in) :: icol ! column number
01398 integer(ip),dimension(:),allocatable,intent(out) :: r
01399 logical,intent(out) :: status_ok
01400
01401 CALL setmessagesource("get_integer_column")
01402

```

```

01403     if (allocated(me%csv_data)) then
01404         allocate(r(me%n_rows)) ! size the output vector
01405         call me%get_column(icol,r,status_ok)
01406     else
01407         scratchmessage = 'Error: class has not been initialized'
01408         CALL allmessage(error, scratchmessage)
01409         status_ok = .false.
01410     end if
01411
01412     CALL unsetmessagesource()
01413
01414 end subroutine get_integer_column
01415 !=====
01416
01417 !-----+
01418 ! S U B R O U T I N E   G E T _ L O G I C A L _ C O L U M N
01419 !-----+
01420 !-----+
01421
01422 subroutine get_logical_column(me,icol,r,status_ok)
01423
01424     implicit none
01425
01426     class(csv_file),intent(inout) :: me
01427     integer,intent(in) :: icol
01428     logical,dimension(:),allocatable,intent(out) :: r
01429     logical,intent(out) :: status_ok
01430
01431     CALL setmessagesource("get_logical_column")
01432
01433     if (allocated(me%csv_data)) then
01434         allocate(r(me%n_rows)) ! size the output vector
01435         call me%get_column(icol,r,status_ok)
01436     else
01437         scratchmessage = 'Error: class has not been initialized'
01438         CALL allmessage(error, scratchmessage)
01439         status_ok = .false.
01440     end if
01441
01442     CALL unsetmessagesource()
01443
01444 end subroutine get_logical_column
01445 !=====
01446
01447 !-----+
01448 ! S U B R O U T I N E   G E T _ C H A R A C T E R _ C O L U M N
01449 !-----+
01450 !-----+
01451
01452 subroutine get_character_column(me, icol ,r, status_ok)
01453
01454     implicit none
01455
01456     class(csv_file),intent(inout) :: me
01457     integer,intent(in) :: icol ! column number
01458     character(len=*,dimension(:),allocatable,intent(out) :: r
01459     logical,intent(out) :: status_ok
01460
01461     CALL setmessagesource("get_character_column")
01462
01463     if (allocated(me%csv_data)) then
01464         allocate(r(me%n_rows)) ! size the output vector
01465         call me%get_column(icol,r,status_ok)
01466     else
01467         scratchmessage = 'Error: class has not been initialized'
01468         CALL allmessage(error, scratchmessage)
01469         status_ok = .false.
01470     end if
01471
01472     CALL unsetmessagesource()
01473
01474 end subroutine get_character_column
01475 !=====
01476
01477 !-----+
01478 ! S U B R O U T I N E   G E T _ C S V _ S T R I N G _ C O L U M N
01479 !-----+
01480 !-----+
01481
01482 subroutine get_csv_string_column(me,icol,r,status_ok)
01483
01484     implicit none
01485
01486     class(csv_file),intent(inout) :: me

```

```

01529     integer,intent(in) :: icol
01530     type(csv_string),dimension(:),allocatable,intent(out) :: r
01531     logical,intent(out) :: status_ok
01532
01533     CALL setmessagesource("get_csv_string_column")
01534
01535     if (allocated(me%csv_data)) then
01536         allocate(r(me%n_rows)) ! size the output vector
01537         call me%get_column(icol,r,status_ok)
01538     else
01539         scratchmessage = 'Error: class has not been initialized'
01540         CALL allmessage(error, scratchmessage)
01541         status_ok = .false.
01542     end if
01543
01544     CALL unsetmessagesource()
01545
01546     end subroutine get_csv_string_column
01547 !=====
01548
01549 !-----
01550 ! S U B R O U T I N E   T O K E N I Z E _ C S V _ L I N E
01551 !-----
01552 !-----
01553 subroutine tokenize_csv_line(me, line, cells)
01554
01555     implicit none
01556
01557     class(csv_file),intent(inout) :: me
01558     character(len=*),intent(in) :: line
01559     type(csv_string),dimension(:),allocatable,intent(out) :: cells
01560
01561     integer :: i ! counter
01562     character(len=:),allocatable :: tmp ! a temp string with whitespace removed
01563     integer :: n ! length of compressed string
01564
01565     call split(line,me%delimiter,me%chunk_size,cells)
01566
01567     ! remove quotes if present:
01568     do i = 1, size(cells)
01569
01570         ! remove whitespace from the string:
01571         tmp = trim(adjustl(cells(i)%str))
01572         n = len(tmp)
01573
01574         if (n>1) then
01575             ! if the first and last non-blank character is
01576             ! a quote, then remove them and replace with what
01577             ! is inside the quotes. Otherwise, leave it as is.
01578             if (tmp(1:1)==me%quote .and. tmp(n:n)==me%quote) then
01579                 if (n>2) then
01580                     cells(i)%str = tmp(2:n-1) ! remove the quotes
01581                 else
01582                     cells(i)%str = " ! empty string
01583                 end if
01584             end if
01585         end if
01586
01587     end do
01588
01589     end subroutine tokenize_csv_line
01590 !=====
01591
01592 !-----
01593 ! F U N C T I O N   N U M B E R _ O F _ L I N E S _ I N _ F I L E
01594 !-----
01595 !-----
01596 function number_of_lines_in_file(iunit) result(n_lines)
01597
01598     implicit none
01599
01600     integer,intent(in) :: iunit ! the file unit number
01601                                         ! (assumed to be open)
01602     integer :: n_lines ! the number of lines in the file
01603
01604     character(len=1) :: tmp
01605     integer :: istat
01606
01607     rewind(iunit)
01608     n_lines = 0
01609     do

```

```

01644      read(iunit,fmt='(A1)',iostat=istat) tmp
01645      if (is_iostat_end(istat)) exit
01646      n_lines = n_lines + 1
01647   end do
01648   rewind(iunit)
01649
01650 end function number_of_lines_in_file
01651 !=====
01652 !-----
01653 !----- S U B R O U T I N E   R E A D _ L I N E _ F R O M _ F I L E
01654 !-----
01655 !-----
01671 !-----
01672 subroutine read_line_from_file(me, iunit, line, status_ok)
01673
01674   implicit none
01675
01676   class(csv_file),intent(in) :: me
01677   integer,intent(in) :: iunit
01678   character(len=:),allocatable,intent(out) :: line
01679   logical,intent(out) :: status_ok ! true if no problems
01680
01681   integer :: nread ! character count specifier for read statement
01682   integer :: istat ! file read io status flag
01683   character(len=me%chunk_size) :: buffer ! the file read buffer
01684
01685   CALL setmessagesource("read_line_from_file")
01686
01687   nread = 0
01688   buffer = ""
01689   line = ""
01690   status_ok = .true.
01691
01692   do
01693     ! read in the next block of text from the line:
01694     read(iunit,fmt='(A)',advance='NO',size=nread,iostat=istat) buffer
01695     if (is_iostat_end(istat) .or. is_iostat_eor(istat)) then
01696       ! add the last block of text before the end of record
01697       if (nread>0) line = line//buffer(1:nread)
01698       exit
01699     else if (istat==0) then ! all the characters were read
01700       line = line//buffer ! add this block of text to the string
01701     else ! some kind of error
01702       WRITE(scratchmessage,'(A,1X,I5)') 'Read error for file unit: ', iunit
01703       CALL allmessage(error, scratchmessage)
01704       status_ok = .false.
01705       exit
01706     end if
01707   end do
01708
01709   CALL unsetmessagesource()
01710
01711 end subroutine read_line_from_file
01712 !=====
01713 !-----
01714 !----- S U B R O U T I N E   S P L I T
01715 !-----
01716 !-----
01744 !-----
01745 pure subroutine split(str, token, chunk_size, vals)
01746
01747   implicit none
01748
01749   character(len=*),intent(in) :: str
01750   character(len=*),intent(in) :: token
01751   integer,intent(in) :: chunk_size
01752   type(csv_string),dimension(:,),allocatable,intent(out) :: vals
01753
01754   integer :: i ! counter
01755   integer :: len_str ! significant length of 'str'
01756   integer :: len_token ! length of the token
01757   integer :: n_tokens ! number of tokens
01758   integer :: i1 ! index
01759   integer :: i2 ! index
01760   integer :: j ! counters
01761   integer,dimension(:,),allocatable :: itokens ! start indices of the
01762                                         ! token locations in 'str'
01763
01764   len_token = len(token) ! length of the token
01765   n_tokens = 0 ! initialize the token counter
01766   j = 0 ! index to start looking for the next token

```

```

01767
01768 ! first, count the number of times the token
01769 ! appears in the string, and get the token indices.
01770 !
01771 ! Examples:
01772 !   ',',      --> 1
01773 !   '1234,67,90' --> 5,8
01774 !   '123,'     --> 4
01775
01776 ! length of the string
01777 if (token == ' ') then
01778   ! in this case, we can't ignore trailing space
01779   len_str = len(str)
01780 else
01781   ! safe to ignore trailing space when looking for tokens
01782   len_str = len_trim(str)
01783 end if
01784
01785 j = 1
01786 n_tokens = 0
01787 do
01788   if (j>len_str) exit      ! end of string, finished
01789   i = index(str(j:),token) ! index of next token in remaining string
01790   if (i<=0) exit          ! no more tokens found
01791   call expand_vector(itokens,n_tokens,chunk_size,i+j-1) ! save the token location
01792   j = j + i + (len_token - 1)
01793 end do
01794 call expand_vector(itokens,n_tokens,chunk_size,finished=.true.) ! resize the vector
01795
01796 allocate(vals(n_tokens+1))
01797
01798 if (n_tokens>0) then
01799
01800   len_str = len(str)
01801
01802   i1 = 1
01803   i2 = itokens(1)-1
01804   if (i2>=i1) then
01805     vals(1)%str = str(i1:i2)
01806   else
01807     vals(1)%str = "" !the first character is a token
01808   end if
01809
01810   !      1 2 3
01811   !      'a,b,c,d'
01812
01813   do i=2,n_tokens
01814     i1 = itokens(i-1)+len_token
01815     i2 = itokens(i)-1
01816     if (i2>=i1) then
01817       vals(i)%str = str(i1:i2)
01818     else
01819       vals(i)%str = "" !empty element (e.g., 'abc,def')
01820     end if
01821   end do
01822
01823   i1 = itokens(n_tokens) + len_token
01824   i2 = len_str
01825   if (itokens(n_tokens)+len_token<=len_str) then
01826     vals(n_tokens+1)%str = str(i1:i2)
01827   else
01828     vals(n_tokens+1)%str = "" !the last character was a token
01829   end if
01830
01831 else
01832   !no tokens present, so just return the original string:
01833   vals(1)%str = str
01834 end if
01835
01836 end subroutine split
01837 =====
01838
01839 END MODULE csv_module

```

20.17 /home/takis/CSDL/parwinds-doc/src/csv_parameters.F90 File Reference

Various parameters.

Modules

- module csv_parameters

Variables

- integer(ip), parameter, public csv_parameters::max_real_str_len = 27
- character(len= *), parameter, public csv_parameters::default_real_fmt = '(E27.17E4)'
- integer(ip), parameter, public csv_parameters::max_integer_str_len = 256
- character(len= *), parameter, public csv_parameters::default_int_fmt = '(I256)'

20.17.1 Detailed Description

Various parameters.

Author

Jacob Williams

Copyright

License BSD

Definition in file [csv_parameters.F90](#).

20.18 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   ! maximum string length of a real number  
00023   INTEGER(IP), PARAMETER, PUBLIC      :: max_real_str_len = 27  
00024  
00025   ! default real number format statement (for writing real values to strings and files)  
00026   CHARACTER(LEN=*), PARAMETER, PUBLIC :: default_real_fmt = '(E27.17E4)'  
00027  
00028   ! maximum string length of an integer  
00029   INTEGER(IP), PARAMETER, PUBLIC      :: max_integer_str_len = 256  
00030  
00031   ! default integer number format statement (for writing real values to strings and files)  
00032   CHARACTER(LEN=*), PARAMETER, PUBLIC :: default_int_fmt = '(I256)'  
00033  
00034  
00035 END MODULE csv_parameters
```

20.19 /home/takis/CSDL/parwinds-doc/src/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)
Finds the unique elements in a vector of integers.
- subroutine, public [csv_utilities::sortAscending](#) (ivec)
Sorts an integer array ivec in increasing order.
- recursive subroutine [quicksort](#) (ilow, ihigh)
- subroutine [partition](#) (ilow, ihigh, ipivot)
- pure elemental subroutine [csv_utilities::swap](#) (i1, i2)
Swap two integer values.

Variables

- integer, parameter [csv_utilities::max_size_for_insertion_sort](#) = 20

20.19.1 Detailed Description

Utility routines.

Author

Jacob Williams

Copyright

License BSD

Definition in file [csv_utilities.F90](#).

20.19.2 Function/Subroutine Documentation

```
20.19.2.1 partition() subroutine sort_descending::partition (
    integer, intent(in) ilow,
    integer, intent(in) ihigh,
    integer, intent(out) ipivot ) [private]
```

Definition at line 214 of file [csv_utilities.F90](#).

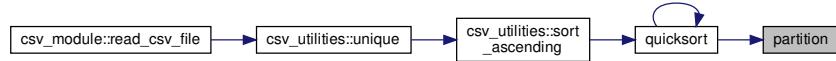
References [csv_utilities::swap\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



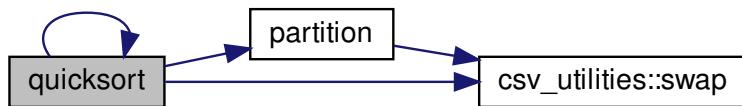
```
20.19.2.2 quicksort() recursive subroutine sort_descending::quicksort (
    integer, intent(in) ilow,
    integer, intent(in) ihigh ) [private]
```

Definition at line 177 of file [csv_utilities.F90](#).

References [csv_utilities::max_size_for_insertion_sort](#), [partition\(\)](#), [quicksort\(\)](#), and [csv_utilities::swap\(\)](#).

Referenced by [quicksort\(\)](#), and [csv_utilities::sort_descending\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



20.20 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 !-----
00004 !-----
00005
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 ! max size for using insertion sort.
00024
00025 PUBLIC :: unique
00026 PUBLIC :: expand_vector
00027 PUBLIC :: sortAscending
00028
00029
00030 CONTAINS
00031
00032 !
00033 !-----+
00034 ! S U B R O U T I N E   E X P A N D _ V E C T O R
00035 !-----+
00036 !
00037 !-----+
00038
00039 pure subroutine expand_vector(vec, n, chunk_size, val, finished)
00040
00041 implicit none
00042
00043 integer,dimension(:),allocatable,intent(inout) :: vec
00044 integer,intent(inout)          :: n           ! counter for last element added to 'vec'.
00045                                ! must be initialized to 'size(vec)'
00046                                ! (or 0 if not allocated) before first call
00047 integer,intent(in)          :: chunk_size ! allocate 'vec' in blocks of this size (>0)
00048 integer,intent(in),optional :: val         ! the value to add to 'vec'
00049 logical,intent(in),optional :: finished    ! set to true to return 'vec'
00050                                ! as its correct size ('n')
00051
00052
00053 integer,dimension(:),allocatable :: tmp ! temporary array
00054
00055
00056 if (present(val)) then
00057     if (allocated(vec)) then
00058         if (n==size(vec)) then
00059             ! have to add another chunk:
00060             allocate(tmp(size(vec)+chunk_size))
00061             tmp(1:size(vec)) = vec
00062             call move_alloc(tmp,vec)
00063         end if
00064         n = n + 1
00065     else
00066         ! the first element:
00067         allocate(vec(chunk_size))
00068         n = 1
00069     end if
00070     vec(n) = val
00071 end if
00072
00073 if (present(finished)) then
00074     if (finished) then
00075         deallocate(vec)
00076     end if
00077 end if

```

```

00088      if (finished) then
00089          ! set vec to actual size (n):
00090          if (allocated(tmp)) deallocate(tmp)
00091          allocate(tmp(n))
00092          tmp = vec(1:n)
00093          call move_alloc(tmp,vec)
00094      end if
00095  end if
00096
00097  end subroutine expand_vector
00098 !=====
00099 !-----+
00100 ! F U N C T I O N   U N I Q U E
00101 !-----+
00118 !-----
00119 function unique(vec,chunk_size) result(ivec_unique)
00120
00121 implicit none
00122
00123 integer,dimension(:),intent(in) :: vec
00124 integer,intent(in) :: chunk_size
00125 integer,dimension(:),allocatable :: ivec_unique
00126
00127 integer,dimension(size(vec)) :: ivec ! temp copy of vec
00128 integer :: i ! counter
00129 integer :: n ! number of unique elements
00130
00131 ! first we sort it:
00132 ivec = vec ! make a copy
00133 call sortAscending(ivec)
00134
00135 ! add the first element:
00136 n = 1
00137 ivec_unique = [ivec(1)]
00138
00139 ! walk through array and get the unique ones:
00140 if (size(ivec)>1) then
00141     do i = 2, size(ivec)
00142         if (ivec(i)/=ivec(i-1)) then
00143             call expand_vector(ivec_unique,n,chunk_size,val=ivec(i))
00144         end if
00145     end do
00146     call expand_vector(ivec_unique,n,chunk_size,finished=.true.)
00147 end if
00148
00149 end function unique
00150 !=====+
00151 !-----+
00152 ! S U B R O U T I N E   U N I Q U E
00153 !-----+
00166 !-----
00167 subroutine sortAscending(ivec)
00168
00169 implicit none
00170
00171 integer,dimension(:),intent(inout) :: ivec
00172
00173 call quicksort(1,size(ivec))
00174
00175 contains
00176
00177     recursive subroutine quicksort(ilow,ihigh)
00178
00179     ! Sort the array
00180
00181     implicit none
00182
00183     integer,intent(in) :: ilow
00184     integer,intent(in) :: ihigh
00185
00186     integer :: ipivot ! pivot element
00187     integer :: i ! counter
00188     integer :: j ! counter
00189
00190     if ( ihigh-ilow<=max_size_for_insertion_sort .and. ihigh>ilow ) then
00191
00192         ! do insertion sort:
00193         do i = ilow + 1,ihigh
00194             do j = i,ilow + 1,-1

```

```

00195      if ( ivec(j) < ivec(j-1) ) then
00196          call swap(ivec(j),ivec(j-1))
00197      else
00198          exit
00199      end if
00200    end do
00201  end do
00202
00203 else if ( ihigh-ilow>max_size_for_insertion_sort ) then
00204
00205     ! do the normal quicksort:
00206     call partition(ilow,ihigh,ipivot)
00207     call quicksort(ilow,ipivot - 1)
00208     call quicksort(ipivot + 1,ihigh)
00209
00210 end if
00211
00212 end subroutine quicksort
00213
00214 subroutine partition(ilow,ihigh,ipivot)
00215
00216     ! Partition the array, based on the
00217     ! lexical ivecng comparison.
00218
00219     implicit none
00220
00221     integer,intent(in) :: ilow
00222     integer,intent(in) :: ihigh
00223     integer,intent(out) :: ipivot
00224
00225     integer :: i,ip
00226
00227     call swap(ivec(ilow),ivec((ilow+ihigh)/2))
00228     ip = ilow
00229     do i = ilow + 1, ihigh
00230         if ( ivec(i) < ivec(ilow) ) then
00231             ip = ip + 1
00232             call swap(ivec(ip),ivec(i))
00233         end if
00234     end do
00235     call swap(ivec(ilow),ivec(ip))
00236     ipivot = ip
00237
00238 end subroutine partition
00239
00240 end subroutine sortAscending
00241 !=====
00242
00243 !-----
00244 ! S U B R O U T I N E   U N I Q U E
00245 !-----
00258 !
00259 pure elemental subroutine swap(il,i2)
00260
00261     implicit none
00262
00263     integer,intent(inout) :: il
00264     integer,intent(inout) :: i2
00265
00266     integer :: tmp
00267
00268     tmp = il
00269     il = i2
00270     i2 = tmp
00271
00272 end subroutine swap
00273 !=====
00274
00275 END MODULE csv_utilities

```

20.21 /home/takis/CSDL/parwinds-doc/src/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`

20.21.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `driver_mod.F90`.

20.22 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 : controlfilename  

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(argc, 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      controlfilename = trim(adjustl(argcmdline))
00075  ENDIF
00076
00077      CALL readcontrolfile(trim(controlfilename))
00078
00079  END SUBROUTINE getprogramcmdlargs
00080
00081 !=====
00082 !-----
00083 !----- S U B R O U T I N E   P A H M   M O D E L   I N I T
00084 !----- S U B R O U T I N E   P A H M   M O D E L   R U N
00085 !-----
00086 !-----
00087
00088  SUBROUTINE pahm_init()
00089
00090      USE pahm_global, ONLY : noutdt
00091      USE pahm_mesh, ONLY : readmesh
00092      USE parwind, ONLY : readbesttrackfile, readcsvbesttrackfile
00093
00094      ! Initialize the logging system, needs to be called first
00095      CALL initlogging()
00096
00097      CALL setmessagesource("PaHM_Init")
00098
00099      ! Get possible command line arguments
00100      CALL getprogramcmdlargs()
00101
00102      ! Read the mesh/grid of the domain or the generic mesh/grid input file
00103      CALL readmesh()
00104
00105      ! Read all track files and save the data into the array of the best track structures
00106      ! for subsequent access by the P-W models in the program
00107      !CALL ReadBestTrackFile()
00108      CALL readcsvbesttrackfile()
00109
00110      cnttimebegin = 1
00111      cnttimeend   = noutdt
00112
00113      CALL unsetmessagesource()
00114
00115  END SUBROUTINE pahm_init
00116
00117 !=====
00118 !----- S U B R O U T I N E   P A H M   M O D E L   R U N
00119 !----- S U B R O U T I N E   P A H M   M O D E L   I N I T
00120 !-----
00121
00122  SUBROUTINE pahm_run(nTimeSTP)
00123
00124      USE pahm_global, ONLY : modeltype
00125      USE parwind
00126      USE pahm_netcdfio
00127
00128      IMPLICIT NONE
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146

```

```

00147      INTEGER, INTENT(IN), OPTIONAL :: nTimeSTP
00148
00149      INTEGER :: iCnt
00150
00151      CALL setmessagesource("PaHM_Run")
00152
00153      IF (PRESENT(ntimestp)) THEN
00154          cnttimeend = cnttimebegin + ntimestp - 1
00155      ENDIF
00156
00157      SELECT CASE (modeltype)
00158          CASE (1)
00159              DO icnt = cnttimebegin, cnttimeend
00160                  CALL gethollandfields(icnt)
00161
00162                  IF (outfilenamespecified) THEN
00163                      ! Create the output NetCDF file and fill it with the static data only
00164                      ! Initialize all variables. This subroutine is called just once
00165                      CALL initadcircnetcdfoutfile(outfilename)
00166
00167                      CALL writenetcdfrecord(outfilename, icnt)
00168                  END IF
00169              END DO
00170
00171          CASE (10)
00172              DO icnt = cnttimebegin, cnttimeend
00173                  CALL getgahmfields(icnt)
00174
00175                  IF (outfilenamespecified) THEN
00176                      ! Create the output NetCDF file and fill it with the static data only
00177                      ! Initialize all variables. This subroutine is called just once
00178                      CALL initadcircnetcdfoutfile(outfilename)
00179
00180                      CALL writenetcdfrecord(outfilename, icnt)
00181                  END IF
00182              END DO
00183
00184          CASE DEFAULT
00185              WRITE(scratchmessage, '(a, i0)' ) &
00186                  'This model type is not supported: modelType = ', modeltype
00187              CALL logmessage(error, scratchmessage)
00188          END SELECT
00189
00190      IF (PRESENT(ntimestp)) THEN
00191          cnttimebegin = cnttimeend + 1
00192      ENDIF
00193
00194      CALL unsetmessagesource()
00195
00196  END SUBROUTINE pahm_run
00197
00198 !=====
00199
00200 !-----
00201 !  S U B R O U T I N E   P A H M   M O D E L   F I N A L I Z E
00202 !-----
00210 !-----
00211 SUBROUTINE pahm_finalize()
00212
00213     CALL setmessagesource("PaHM_Finalize")
00214
00215     CALL unsetmessagesource()
00216
00217     ! Close the logging facilities
00218     CALL closelogfile()
00219
00220  END SUBROUTINE pahm_finalize
00221
00222 !=====
00223
00224 END MODULE pahm_drivermod

```

20.23 /home/takis/CSDL/parwinds-doc/src/global.F90 File Reference

Modules

- module pahm_global

Functions/Subroutines

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

This function calculates the density of the moist air.

Variables

- integer, parameter `pahm_global::lun_screen` = 6
- integer, parameter `pahm_global::lun_ctrl` = 10
- integer, parameter `pahm_global::lun_inp` = 14
- integer, parameter `pahm_global::lun_inp1` = 15
- integer, parameter `pahm_global::lun_log` = 35
- integer, parameter `pahm_global::lun_btrk` = 22
- integer, parameter `pahm_global::lun_btrk1` = 23
- integer, parameter `pahm_global::lun_out` = 25
- integer, parameter `pahm_global::lun_out1` = 26
- real(sz), parameter `pahm_global::defv_gravity` = 9.80665_SZ
- real(sz), parameter `pahm_global::defv_atmpress` = 1013.25_SZ
- real(sz), parameter `pahm_global::defv_rhoair` = 1.1478_SZ
- real(sz), parameter `pahm_global::defv_rhowater` = 1000.0000
- real(sz), parameter `pahm_global::one2ten` = 0.8928_SZ
- real(sz), parameter `pahm_global::ten2one` = 1.0_SZ / 0.8928_SZ
- real(sz), parameter `pahm_global::pi` = 3.141592653589793_SZ
- real(sz), parameter `pahm_global::deg2rad` = PI / 180.0_SZ
- real(sz), parameter `pahm_global::rad2deg` = 180.0_SZ / PI
- real(sz), parameter `pahm_global::basee` = 2.718281828459045_SZ
- real(sz), parameter `pahm_global::rearth` = 6378206.4_SZ
- real(sz), parameter `pahm_global::nm2m` = 1852.0_SZ
- real(sz), parameter `pahm_global::m2nm` = 1.0_SZ / NM2M
- real(sz), parameter `pahm_global::kt2ms` = NM2M / 3600.0_SZ
- real(sz), parameter `pahm_global::ms2kt` = 1.0_SZ / KT2MS
- real(sz), parameter `pahm_global::omega` = 2.0_SZ * PI / 86164.2_SZ
- real(sz), parameter `pahm_global::mb2pa` = 100.0_SZ
- real(sz), parameter `pahm_global::mb2kpa` = 0.1_SZ
- character(len=fnamelen) `pahm_global::logfilename` = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) `pahm_global::controlfilename` = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_control.in'
- logical `pahm_global::meshfilenamespecified` = .FALSE.
- character(len=fnamelen) `pahm_global::meshfilename` = BLANK
- character(len=64) `pahm_global::meshfiletype` = BLANK
- character(len=64) `pahm_global::meshfileform` = BLANK
- logical `pahm_global::besttrackfilenamespecified` = .FALSE.
- integer `pahm_global::nbtrfiles` = IMISSV
- character(len=fnamelen), dimension(:), allocatable `pahm_global::besttrackfilename`
- character(len=512) `pahm_global::title` = BLANK
- real(sz) `pahm_global::gravity` = DEFV_GRAVITY
- real(sz) `pahm_global::rhowater` = DEFV_RHOWATER
- real(sz) `pahm_global::rhoair` = DEFV_RHOAIR
- real(sz) `pahm_global::backgroundatmpress` = DEFV_ATMPRESS
- real(sz), parameter `pahm_global::defv_windreduction` = 0.90_SZ
- real(sz) `pahm_global::windreduction` = DEFV_WINDREDUCTION

- 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::ncplevel = 0
- character(len=20), parameter pahm_global::def_ncnam_pres = 'P'
- character(len=20), parameter pahm_global::def_ncnam_wndx = 'uwnd'
- character(len=20), parameter pahm_global::def_ncnam_wndy = 'vwnd'
- character(len=20) pahm_global::ncvarnam_pres = DEF_NCNAM_PRES
- character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAM_WNDX
- character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAM_WNDY

- integer `pahm_global::modeltype` = IMISSV
- logical `pahm_global::writeparams` = .FALSE.
- real(sz), dimension(:), allocatable `pahm_global::wvelx`
- real(sz), dimension(:), allocatable `pahm_global::wvely`
- real(sz), dimension(:), allocatable `pahm_global::wpress`
- real(sz), dimension(:), allocatable `pahm_global::times`
- character(19), dimension(:), allocatable `pahm_global::datestimes`

20.23.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `global.F90`.

20.24 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_btrkl = 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_outl = 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                                         ! 1.1478 (1013.25 mb, Rel. Hum 90%, 30 deg C)
00047  

00048 ! 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 :: tenZone = 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 :: VARTABLES RELATED TO THE CONTROL FILE
00095 !#####
00096 CHARACTER(LEN=FNAMELEN) :: logfile = trim(adjustl(prog_name_low)) // '_model.log'
00097
00098 !----- Input files
00099 CHARACTER(FNAMELEN) :: controlfilename = 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=FNAMELEN) :: meshfilename = blank        ! there is no default value here
00103 CHARACTER(LEN=64) :: meshfiletype = blank            ! ADCIRC, SCHISM, FVCOM, ROMS, GENERIC
(no default)
00104 CHARACTER(LEN=64) :: meshfileform = blank            ! ASCII, NETCDF (no default)
00105
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_windreduction = 0.90_sz
00121 REAL(sz) :: windreduction = defv_windreduction ! BL reduction factor used in the
Holland model
00122
00123 !=====
00124 !== This block is for the : time/date and time stepping variables
00125 !=====
00126 !---
00127 ! the reference date/time for the model run YYYYMMDDhhmmss
00128 CHARACTER(LEN=64) :: refdatetime = blank
00129 INTEGER :: refdate = imissv

```

```

00130    INTEGER          :: reftime      = imissv
00131    INTEGER          :: refyear      = imissv
00132    INTEGER          :: refmonth     = 0
00133    INTEGER          :: refday       = 0
00134    INTEGER          :: refhour      = 0
00135    INTEGER          :: refmin       = 0
00136    INTEGER          :: refsec       = 0
00137    LOGICAL          :: refdatespecified = .false.
00138    !---
00139    ! the start date/time for the model run YYYYMMDDhhmmss
00140    CHARACTER(LEN=64)   :: begdatetime = blank
00141    INTEGER          :: begdate      = imissv
00142    INTEGER          :: begtime      = imissv
00143    INTEGER          :: begyear      = imissv
00144    INTEGER          :: begmonth     = 0
00145    INTEGER          :: begday       = 0
00146    INTEGER          :: beghour      = 0
00147    INTEGER          :: begin        = 0
00148    INTEGER          :: begsec       = 0
00149    LOGICAL          :: begdatespecified = .false.
00150    !---
00151    ! the stop date/time for the model run YYYYMMDDhhmmss
00152    CHARACTER(LEN=64)   :: enddatetime = blank
00153    INTEGER          :: enddate      = imissv
00154    INTEGER          :: endtime      = imissv
00155    INTEGER          :: endyear      = imissv
00156    INTEGER          :: endmonth     = 0
00157    INTEGER          :: endday       = 0
00158    INTEGER          :: endhour      = 0
00159    INTEGER          :: endmin       = 0
00160    INTEGER          :: endsec       = 0
00161    LOGICAL          :: enddatespecified = .false.
00162    !---
00163    ! alternative definitions for the stop date/time for the model run
00164    REAL(sz)          :: begsimtime   = rmissv
00165    REAL(sz)          :: endsimtime   = rmissv
00166    LOGICAL          :: begsimspecified = .false.
00167    LOGICAL          :: endsimspecified = .false.
00168
00169    CHARACTER(LEN=1)   :: unittime = 'S'
00170    !=====
00171
00172    !---
00173    ! time stepping variables for the model run
00174    REAL(sz)          :: outdt        = rmissv
00175    INTEGER          :: noutdt      = imissv
00176    REAL(sz)          :: mdoutdt     = rmissv
00177    REAL(sz)          :: mdbegsimtime = rmissv
00178    REAL(sz)          :: mdendsimtime = rmissv
00179
00180    LOGICAL          :: outfilenamespecified = .false.
00181    CHARACTER(LEN=FNAMELEN) :: outfilename = blank ! Name of the output NetCDF file
00182    INTEGER          :: ncshuffle    = 0           ! Turn on the shuffle filter (>0)
00183    INTEGER          :: ncdeflate    = 0           ! Turn on the deflate filter (>0)
00184    INTEGER          :: nclevel      = 0           ! Deflate level [0-9]
00185
00186    ! Create a list of NetCDF variable names in the form ncYyyyVarNam = value
00187    ! The user can specify his/her own values in the control file (will be hidden variables)
00188    ! Default values
00189    CHARACTER(LEN=20), PARAMETER :: def_ncnam_pres = 'P',      &
00190                                def_ncnam_wndx = 'uwnd',    &
00191                                def_ncnam_wndy = 'vwnd'
00192
00193    CHARACTER(LEN=20)      :: ncvarnam_pres = def_ncnam_pres, &
00194                                ncvarnam_wndx = def_ncnam_wndx, &
00195                                ncvarnam_wndy = def_ncnam_wndy
00196
00197    INTEGER          :: modeltype = imissv      ! The parametric model to use
00198                                ! 0: Rankin Vortex
00199                                ! 1: Holland B (1998)
00200                                ! 2: Holland B (2010)
00201                                ! 3: Willoughby model
00202                                ! 9: Assymmetric vortex model (Mattocks)
00203                                ! 10: Generalized assymmetric vortex Holland model
00204    LOGICAL          :: writeparams = .false.
00205    ##### END : VARIABLES RELATED TO THE CONTROL FILE
00206    ##### #####
00207    #####
00208
00209

```

```

00210 !#####
00211 !###     BEG :: GLOBAL DATA ARRAYS
00212 !#####
00213 ! Arrays to hold the P-W fields
00214 !REAL(SZ), DIMENSION(:, :), ALLOCATABLE :: wVelX, wVelY, wPress
00215 REAL(sz), DIMENSION(:, ), ALLOCATABLE :: wvelx, wvely, wpress
00216 REAL(sz), DIMENSION(:, ), ALLOCATABLE :: times
00217 CHARACTER(19), DIMENSION(:, ), ALLOCATABLE :: datestimes
00218 !#####
00219 !###     END :: GLOBAL DATA ARRAYS
00220 !#####
00221
00222
00223 CONTAINS
00224
00225
00226 !-----
00227 ! FUNCTION AIR DENSITY
00228 !-----
00229 ! >see http://www.emd.dk/files/windpro/WindPRO\_AirDensity.pdf
00230 !-----
00231 REAL(sz) function airdensity(atmt, atmp, relhum) result(myvalout)
00232
00233     IMPLICIT NONE
00234
00235     REAL(sz), INTENT(IN) :: atmt      ! Surface temperature in degrees C (-50.0 <= T <= 100.0)
00236     REAL(sz), INTENT(IN) :: atmp      ! Atmospheric pressure (mb)
00237     REAL(sz), INTENT(IN) :: relhum    ! Relative humidity (0 - 100)
00238
00239     ! Local variables
00240     REAL(hp)           :: es, p, pv, pd, rh
00241     REAL(hp)           :: rd, rv, temp, tempk, dens
00242
00243     rh = relhum
00244     IF (rh < 0.01) rh = 0.01_hp
00245     IF (rh > 100.0) rh = 100.0_hp
00246
00247     temp = atmt
00248     IF (temp < -50.0) temp = -50.0_hp
00249     IF (temp > 100.0) temp = 100.0_hp
00250
00251     rd = 287.058_hp ! specific gas constant for dry air (J/kg*K)
00252     rv = 461.495_hp ! specific gas constant for water vapor (J/kg*K)
00253
00254     ! Convert relative humidity to %
00255     rh = 0.01_sz * rh
00256
00257     ! Convert atmT (C) to K
00258     tempk = temp + 273.15_hp
00259
00260     ! Calculate the saturated vapor pressure (mb)
00261     ! Temperature is in degrees Celcius
00262     p = 0.99999683e+00_hp + temp * (-0.90826951e-02_hp + temp * (0.78736169e-04_hp + temp * &
00263                                         (-0.61117958e-06_hp + temp * (0.43884187e-08_hp + temp * &
00264                                         (-0.29883885e-10_hp + temp * (0.21874425e-12_hp + temp * &
00265                                         (-0.17892321e-14_hp + temp * (0.11112018e-16_hp + temp * &
00266                                         (-0.30994571e-19_hp)))))))
00267     es = 6.1078_hp / p**8    ! saturated vapour pressure (mb)
00268
00269     ! Calculate the actual vapor pressure (mb)
00270     pv = es * rh
00271
00272     ! Calculate the actual vapor pressure
00273     pd = atmp - pv
00274
00275     ! Convert the pressures from mb to Pa
00276     pd = pd * 100.0_hp
00277     pv = pv * 100.0_hp
00278
00279     ! Calculate the air density
00280     dens = pd / (rd * tempk) + pv / (rv * tempk)
00281
00282     myvalout = dens
00283
00284     RETURN
00285
00286     END FUNCTION airdensity
00287
00288 !=====
00289

```

```
00310 END MODULE pahm_global
```

20.25 /home/takis/CSDL/parwinds-doc/src/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::allocatenodalandallementalarrays ()
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::fn
- 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::xclam
- 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.

20.25.1 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](#).

20.26 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
00054 CONTAINS
00055
00056
00057 !-----+

```

```

00058 ! S U B R O U T I N E   R E A D   M E S H
00059 !-----+
00068 !-----+
00069 SUBROUTINE readmesh()
00070
00071 USE pahm_global, ONLY : meshfilenamespecified, meshfilename, meshfiletype, meshfileform
00072 USE utilities, ONLY : touppercase
00073
00074 IMPLICIT NONE
00075
00076
00077 CALL setmessagesource("ReadMesh")
00078
00079 IF (meshfilenamespecified .EQV. .false.) THEN
00080   WRITE(scratchmessage, '(a)') 'ReadMesh: First specify a valid grid filename to proceed: ' // &
00081   '!' // trim(meshfilename) // ']'
00082   CALL allmessage(error, scratchmessage)
00083   CALL terminate()
00084 END IF
00085
00086 SELECT CASE(touppercase(meshfiletype))
00087   !---- ADCIRC case
00088   CASE('ADCIRC')
00089     SELECT CASE(touppercase(meshfileform))
00090       CASE('ASCII')
00091         CALL readmeshasciifort14()
00092
00093       CASE('NETCDF')
00094         WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file
00095 type: ' // &
00096         '!' // trim(meshfiletype) // ']'
00097         CALL allmessage(error, scratchmessage)
00098         CALL terminate()
00099
00100       CASE DEFAULT
00101         WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the
00102 mesh file type: ' // &
00103         '!' // trim(meshfiletype) // ']'
00104         CALL allmessage(error, scratchmessage)
00105         CALL terminate()
00106   END SELECT
00107
00108   !---- SCHISM case
00109   CASE('SCHISM')
00110     SELECT CASE(touppercase(meshfileform))
00111       CASE('ASCII')
00112         CALL readmeshasciifort14()
00113
00114       CASE('NETCDF')
00115         WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file
00116 type: ' // &
00117         '!' // trim(meshfiletype) // ']'
00118         CALL allmessage(error, scratchmessage)
00119         CALL terminate()
00120
00121       CASE DEFAULT
00122         WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the
00123 mesh file type: ' // &
00124         '!' // trim(meshfiletype) // ']'
00125         CALL allmessage(error, scratchmessage)
00126         CALL terminate()
00127   END SELECT
00128
00129   !---- FVCOM case
00130   CASE('FVCOM')
00131     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00132     '!' // trim(meshfiletype) // ']'
00133     CALL allmessage(error, scratchmessage)
00134     CALL terminate()
00135
00136   !---- ROMS case
00137   CASE('ROMS')
00138     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00139     '!' // trim(meshfiletype) // ']'
00140     CALL allmessage(error, scratchmessage)
00141     CALL terminate()
00142
00143   !---- GENERIC case
00144   CASE('GENERIC')
00145     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00146     '!' // 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 !-----+ S U B R O U T I N E   R E A D   M E S H   A S C I I   F O R T   1 4
00160 !-----+
00161 !-----+
00162 !-----+
00163 !-----+
00164 !-----+
00165 !-----+
00166 !-----+
00167 !-----+
00168 !-----+
00169 !-----+
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("ReadMeshASCIIFort14")
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 allocateandalandelementalarrays()
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 ", io,'
00214             fmtstr = trim(fmtstr) // ' " contains invalid (lon, lat) values: ", " [", f14.4, ", ", f14.4,
00215             "]' "
00216             fmtstr = trim(fmtstr) // ' " (should be degrees east and degrees north)"'
00217             WRITE(scratchmessage, trim(fmtstr)) lineNum, slam(icnt), sfea(icnt)
00218
00219             CALL allmessage(error, scratchmessage)
00220             CLOSE(iunit)
00221             CALL terminate()
00222         END IF
00223
00224         lineNum = lineNum + 1
00225     END DO
00226
00227     ! E L E M E N T   T A B L E
00228     DO icnt = 1, ne
00229         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 ! S U B R O U T I N E A L L O C A T E N O D A L A N D E L E M E N T A L A R R A Y S
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

```

20.27 /home/takis/CSDL/parwinds-doc/src/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::sourcenumber`
- logical `pahm_messages::logfileopened` = .FALSE.
- logical `pahm_messages::loginitcalled` = .FALSE.

20.27.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Note

Adopted from the ADCIRC source code.

Definition in file `messages.F90`.

20.28 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   loglevelname(1) = "DEBUG"
00088   loglevelname(2) = "ECHO"
00089   loglevelname(3) = "INFO"
00090   loglevelname(4) = "WARNING"
00091   loglevelname(5) = "ERROR"
00092
00093   loginitcalled = .true.
00094
00095   CALL openlogfile
00096 END IF
00097
00098 END SUBROUTINE initlogging
00099
00100 !=====
00101
00102 !    S U B R O U T I N E    O P E N    L O G    F I L E
00103 !-----
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 !
00140 !-----
00141 !-----
00142 SUBROUTINE closelogfile()
00143
00144     IMPLICIT NONE
00145
00146     IF (logfileopened) CLOSE(unit=lun_log)
00147
00148 END SUBROUTINE closelogfile
00149
00150
00151
00152
00153
00154
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 !
00161 !-----
00162 !-----
00163 SUBROUTINE screenmessage_1(message)
00164
00165     IMPLICIT NONE
00166
00167
00168 ! Global variables
00169 CHARACTER(LEN=*), INTENT(IN) :: message
00170
00171
00172 IF (nscreen > 0) THEN
00173     IF (loginitcalled) THEN
00174         WRITE(lun_screen, '(a)') '    --- ' // trim(adjustl(message))
00175     ELSE
00176         WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called',
00177                                         trim(adjustl(messagesources(sourcenumber))), &
00178                                         trim(adjustl(message))
00179
00180     END IF
00181
00182 !#ifdef FLUSH_MESSAGES
00183     ! In Fortran >=2003 the call is:
00184     ! FLUSH(LUN_LOG)
00185     CALL flush(lun_screen)
00186
00187 !#endif
00188
00189 END IF
00190
00191
00192 END SUBROUTINE screenmessage_1
00193
00194
00195 SUBROUTINE screenmessage_2(level, message)
00196
00197     IMPLICIT NONE
00198
00199 ! Global variables
00200 INTEGER, INTENT(IN) :: level
00201 CHARACTER(LEN=*), INTENT(IN) :: message
00202
00203
00204 IF (nscreen > 0) THEN
00205     IF (loginitcalled) THEN
00206         WRITE(lun_screen, '(a, " :: ", a, ": ", a)') trim(adjustl(loglevelnames(level + 2))), &
00207                                         trim(adjustl(messagesources(sourcenumber))), &
00208                                         trim(adjustl(message))
00209     ELSE
00210         WRITE(lun_screen, '(a, " :: ", a, ": ", a)') 'InitLogging not called',
00211                                         trim(adjustl(messagesources(sourcenumber))), &
00212                                         trim(adjustl(message))
00213
00214     END IF
00215 !#ifdef FLUSH_MESSAGES
00216     ! In Fortran >=2003 the call is:
00217     ! FLUSH(LUN_LOG)
00218     CALL flush(lun_screen)
00219
00220 !#endif
00221
00222 END IF
00223
00224
00225 END SUBROUTINE screenmessage_2
00226
00227
00228
00229 !=====
00230
00231 !-----
00232 !      S U B R O U T I N E      L O G      M E S S A G E
00233 !
00234 !-----
00235 !-----
00236 !-----
00237 SUBROUTINE logmessage_1(message)
00238
00239
00240
00241
00242
00243
00244
00245
00246
00247
00248
00249
00250
00251
00252

```

```

00253     IMPLICIT NONE
00254
00255     ! Global variables
00256     CHARACTER(LEN=*), INTENT(IN) :: message
00257
00258     IF (logfileopened) THEN
00259         IF (loginitcalled) THEN
00260             WRITE(lun_log, '(a)') ' --- ' // trim(adjustl(message))
00261         ELSE
00262             WRITE(lun_log, '(a, :: ", a, :: ", a)') 'InitLogging not called',
00263                                         trim(adjustl(messagesources(sourcenumber))), &
00264                                         trim(adjustl(message)) &
00265         END IF
00266     !#ifdef FLUSH_MESSAGES
00267         ! In Fortran >=2003 the call is:
00268         ! FLUSH(LUN_LOG)
00269         CALL flush(lun_log)
00270     !#endiff
00271     END IF
00272
00273 END SUBROUTINE logmessage_1
00274
00275 SUBROUTINE logmessage_2(level, message)
00276
00277     IMPLICIT NONE
00278
00279     ! Global variables
00280     INTEGER, INTENT(IN) :: level
00281     CHARACTER(LEN=*), INTENT(IN) :: message
00282
00283     IF (logfileopened) THEN
00284         IF (loginitcalled) THEN
00285             WRITE(lun_log, '(a, :: ", a, :: ", a)') trim(adjustl(loglevelname(level + 2))), &
00286                                         trim(adjustl(messagesources(sourcenumber))), &
00287                                         trim(adjustl(message))
00288         ELSE
00289             WRITE(lun_log, '(a, :: ", a, :: ", a)') 'InitLogging not called',
00290                                         trim(adjustl(messagesources(sourcenumber))), &
00291                                         trim(adjustl(message)) &
00292         END IF
00293     !#ifdef FLUSH_MESSAGES
00294         ! In Fortran >=2003 the call is:
00295         ! FLUSH(LUN_LOG)
00296         CALL flush(lun_log)
00297     !#endiff
00298     END IF
00299
00300 END SUBROUTINE logmessage_2
00301
00302 !=====
00303
00304 !-----S U B R O U T I N E   A L L   M E S S A G E-----
00305 !-----S U B R O U T I N E   A L L   M E S S A G E-----
00306 !-----S U B R O U T I N E   A L L   M E S S A G E-----
00307 !-----S U B R O U T I N E   A L L   M E S S A G E-----
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
00322 SUBROUTINE allmessage_2(level, message)
00323
00324     IMPLICIT NONE
00325
00326     ! Global variables
00327     INTEGER, INTENT(IN) :: level
00328     CHARACTER(LEN=*), INTENT(IN) :: message
00329
00330     CALL screenmessage(level, message)
00331     CALL logmessage(level, message)
00332
00333 END SUBROUTINE allmessage_2
00334
00335
00336
00337
00338
00339
00340
00341
00342
00343

```

```

00344 !=====
00345 !
00346 !-----+
00347 !      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
00348 !
00360 !-----
00361 SUBROUTINE setmessagesource(source)
00362
00363     IMPLICIT NONE
00364
00365     ! Global variables
00366     CHARACTER(LEN=*) , INTENT(IN) :: source
00367
00368     sourcenumber = sourcenumber + 1
00369     messagesources(sourcenumber) = source
00370
00371 END SUBROUTINE setmessagesource
00372
00373 !=====
00374
00375 !-----+
00376 !      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
00377 !
00387 !-----
00388 SUBROUTINE unsetmessagesource()
00389
00390     IMPLICIT NONE
00391
00392     sourcenumber = sourcenumber - 1
00393
00394 END SUBROUTINE unsetmessagesource
00395
00396 !=====
00397
00398 !-----+
00399 !      S U B R O U T I N E   P R O G R A M   V E R S I O N
00400 !
00408 !-----
00409 SUBROUTINE programversion()
00410
00411     USE version
00412
00413     IMPLICIT NONE
00414
00415     WRITE(lun_screen, '(a)') trim(prog_fullname) // ' ' // trim(prog_version) // ' ' // trim(prog_date)
00416     !      WRITE(LUN_SCREEN, '(a)') 'NOAA/NOS/CSDL, Coastal Marine Modeling Branch.'
00417     !      WRITE(lun_screen, '(a)') ' Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/).'
00418     !      WRITE(lun_screen, '(a)') ' NOAA/NOS/CSDL (https://nauticalcharts.noaa.gov/).'
00419     !      WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - This is free software; see the source for copying
00420     !      conditions.'
00420     !      WRITE(lun_screen, '(a)') 'NEED FORMAL DISCLAIMER - There is NO warranty.'
00421
00422     WRITE(lun_screen, '(a)') "
00423
00424 END SUBROUTINE programversion
00425
00426 !=====
00427
00428 !-----+
00429 !      S U B R O U T I N E   P R O G R A M   H E L P
00430 !
00438 !-----
00439 SUBROUTINE programhelp()
00440
00441     IMPLICIT NONE
00442
00443     CALL programversion
00444
00445     WRITE(lun_screen, '(a)') 'Help Screen not yet implemented'
00446
00447     WRITE(lun_screen, '(a)') "
00448
00449 END SUBROUTINE programhelp
00450
00451 !=====
00452
00453 !-----+
00454 !      S U B R O U T I N E   T E R M I N A T E
00455 !
00463 !
00464 SUBROUTINE terminate()

```

```

00465      USE version
00466
00467      IMPLICIT NONE
00468
00469      CALL setmessagesource("Terminate")
00470
00471      CALL allmessage(error, trim(adjustl(prog_name)) // " Terminating.")
00472
00473      CALL unsetmessagesource()
00474
00475      stop
00476
00477      END SUBROUTINE terminate
00478
00479 !=====
00480 !=====
00481 END MODULE pahm_messages

```

20.29 /home/takis/CSDL/parwinds-doc/src/netcdfio.F90 File Reference

Data Types

- type [pahm_netcdfio::filedata_t](#)
- type [pahm_netcdfio::timedata_t](#)
- type [pahm_netcdfio::adcirccoorddata_t](#)
- type [pahm_netcdfio::adcircvardata_t](#)
- type [pahm_netcdfio::adcircvardata3d_t](#)

Modules

- module [pahm_ncdfio](#)

Macros

- #define [NetCDFCheckErr](#)(arg) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

Functions/Subroutines

- subroutine [pahm_ncdfio::initadcircnetcdfoutfile](#) (adcircOutFile)

Initializes a new NetCDF data file and puts it in define mode.
- subroutine [pahm_ncdfio::newadcircnetcdfoutfile](#) (nclD, adcircOutFile)

Creates a new NetCDF data file and puts it in define mode.
- subroutine [pahm_ncdfio::base_ncdfcheckerr](#) (ierr, file, line)

Checks the return value from netCDF calls.
- subroutine [pahm_ncdfio::netcdfterminate](#) ()

Terminates the program on NetCDF error.
- subroutine [pahm_ncdfio::writenetcdfrecord](#) (adcircOutFile, timeLoc)

Writes data to the NetCDF file.
- subroutine [pahm_ncdfio::setrecordcounterandstoretime](#) (nclD, f, t)

Sets the record counter.

Variables

- integer, private `pahm_netcdfio::ncformat`
- integer, parameter, private `pahm_netcdfio::nc4form = IOR(NF90_NETCDF4, NF90_CLASSIC_MODEL)`
- integer, parameter, private `pahm_netcdfio::nc3form = IOR(NF90_CLOBBER, 0)`
- integer, private `pahm_netcdfio::nodedimid`
- integer, private `pahm_netcdfio::vertdimid`
- integer, private `pahm_netcdfio::elemdimid`
- integer, private `pahm_netcdfio::meshdimid`
- integer, private `pahm_netcdfio::meshvarid`
- integer, private `pahm_netcdfio::projvarid`
- type(`filedata_t`), save `pahm_netcdfio::myfile`
- type(`timedata_t`), save `pahm_netcdfio::mytime`
- type(`adcirccoorddata_t`), save, private `pahm_netcdfio::crdtime`
- type(`adcirccoorddata_t`), save, private `pahm_netcdfio::crdlons`
- type(`adcirccoorddata_t`), save, private `pahm_netcdfio::crdlats`
- type(`adcirccoorddata_t`), save, private `pahm_netcdfio::crdxcs`
- type(`adcirccoorddata_t`), save, private `pahm_netcdfio::crdycs`
- type(`adcircvardata_t`), save, private `pahm_netcdfio::datelements`
- type(`adcircvardata_t`), save, private `pahm_netcdfio::datatmpres`
- type(`adcircvardata_t`), save, private `pahm_netcdfio::datwindx`
- type(`adcircvardata_t`), save, private `pahm_netcdfio::datwindy`

20.29.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `netcdfio.F90`.

20.29.2 Macro Definition Documentation

20.29.2.1 NetCDFCheckErr

```
#define NetCDFCheckErr(  
    arg ) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
```

20.30 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 :: 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 :: adcirccoorddata_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 adcirccoorddata_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 :: crdtme
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 ! S U B R O U T I N E   I N I T   A D C I R C   N E T C D F   O U T   F I L E
00109 !-----
00112 !-----
00123 SUBROUTINE initadcircnetcdfoutfile(adcircOutFile)
00124
00125 USE version
00126 USE timedateutils, ONLY : gettimeconvsec, datetime2string
00127
00128 IMPLICIT NONE
00129
00130 CHARACTER(LEN=*), INTENT(INOUT) :: adcircOutFile
00131
00132 INTEGER :: ncID
00133 CHARACTER(LEN=64) :: refDateTimeStr, modDateTimeStr, tmpVarName
00134 CHARACTER(LEN=128) :: institution, source, history, comments, host, &
00135 conventions, contact, references
00136 INTEGER :: tvals(8)
00137 INTEGER :: ierr ! success or failure of a netcdf call
00138 INTEGER :: iCnt, jCnt
00139
00140 LOGICAL, SAVE :: firstCall = .true.
00141
00142 IF (firstcall) THEN
00143   firstcall = .false.
00144
00145 CALL setmessagesource("InitAdcircNetCDFOutFile")
00146
00147
00148 refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units =
00149 unittime)
00150
00151 institution = 'NOAA/OCS/CSDL Coastal Marine Modeling Branch (https://coastaloceanmodels.noaa.gov/)'
00152 source = ''
00153 history = ''
00154 comments = ''
00155 host = ''
00156 conventions = 'UGRID-0.9.0'
00157 contact = 'Panagiotis Velissariou <panagiotis.velissariou@noaa.gov>'
00158 references = ''
00159
00160
00161 ! Create the NetCDF output file.
00162 CALL newadcircnetcdfoutfile(ncid, adcircoutfile)
00163
00164 =====
00165 !==== (1) Define all the dimensions
00166 !=====
00167 tmpvarname = 'time'
00168   ierr = nf90_def_dim(ncid, trim(tmpvarname), nf90_unlimited, crdtme%dimID)
00169   CALL netcdfcheckerr(ierr)
00170
00171 tmpvarname = 'longitude'
00172   ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlons%dimID)
00173   CALL netcdfcheckerr(ierr)
00174
00175 tmpvarname = 'latitude'
00176   ierr = nf90_def_dim(ncid, trim(tmpvarname), np, crdlats%dimID)
00177   CALL netcdfcheckerr(ierr)
00178

```

```

00179     tmpvarname = 'node'
00180     ierr = nf90_def_dim(ncid, trim(tmpvarname), np, nodedimid)
00181         CALL netcdfcheckerr(ierr)
00182
00183     tmpvarname = 'element'
00184     ierr = nf90_def_dim(ncid, trim(tmpvarname), ne, elemdimid)
00185         CALL netcdfcheckerr(ierr)
00186
00187     tmpvarname = 'noel'
00188     ierr = nf90_def_dim(ncid, trim(tmpvarname), 3, vertdimid)
00189         CALL netcdfcheckerr(ierr)
00190
00191     tmpvarname = 'mesh'
00192     ierr = nf90_def_dim(ncid, trim(tmpvarname), 1, meshdimid)
00193         CALL netcdfcheckerr(ierr)
00194
00195 !=====
00196 !===== (2) Define all the variables
00197 !=====
00198 !---- Time variable
00199     tmpvarname = 'time'
00200         crdtime%varname = trim(tmpvarname)
00201         crdtime%varDimIDs = crdtime%dimID
00202         crdtime%varDims = SIZE(times, 1)
00203         crdtime%start(1) = 1
00204         crdtime%count(1) = crdtime%varDims
00205
00206     ierr = nf90_def_var(ncid, 'time', nf90_double, crdtime%varDimIDs, crdtime%varID)
00207         CALL netcdfcheckerr(ierr)
00208     ierr = nf90_put_att(ncid, crdtime%varID, 'long_name',      'model' // trim(tmpvarname))
00209         CALL netcdfcheckerr(ierr)
00210     ierr = nf90_put_att(ncid, crdtime%varID, 'standard_name', trim(tmpvarname))
00211         CALL netcdfcheckerr(ierr)
00212     ierr = nf90_put_att(ncid, crdtime%varID, 'units',        trim(refdatetimestr))
00213         CALL netcdfcheckerr(ierr)
00214
00215 !---- Longitude variable
00216     tmpvarname = 'longitude'
00217         crdlons%varname = trim(tmpvarname)
00218         crdlons%varDimIDs = nodedimid
00219     ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlons%varDims)
00220         CALL netcdfcheckerr(ierr)
00221         crdlons%start(1) = 1
00222         crdlons%count(1) = crdlons%varDims
00223
00224     ierr = nf90_def_var(ncid, trim(crdlons%varname), nf90_double, crdlons%varDimIDs, crdlons%varID)
00225         CALL netcdfcheckerr(ierr)
00226     ierr = nf90_put_att(ncid, crdlons%varID, 'long_name',      trim(tmpvarname))
00227         CALL netcdfcheckerr(ierr)
00228     ierr = nf90_put_att(ncid, crdlons%varID, 'standard_name', trim(tmpvarname))
00229         CALL netcdfcheckerr(ierr)
00230     ierr = nf90_put_att(ncid, crdlons%varID, 'units',        'degrees_east')
00231         CALL netcdfcheckerr(ierr)
00232     ierr = nf90_put_att(ncid, crdlons%varID, '_FillValue',   rmissv)
00233         CALL netcdfcheckerr(ierr)
00234     ierr = nf90_put_att(ncid, crdlons%varID, 'positive',    'east')
00235         CALL netcdfcheckerr(ierr)
00236
00237     ALLOCATE(crdlons%var(crdlons%varDims))
00238     crdlons%var = slam
00239
00240 !---- Latitude variable
00241     tmpvarname = 'latitude'
00242         crdlats%varname = trim(tmpvarname)
00243         crdlats%varDimIDs = nodedimid
00244     ierr = nf90_inquire_dimension(ncid, nodedimid, len = crdlats%varDims)
00245         CALL netcdfcheckerr(ierr)
00246         crdlats%start(1) = 1
00247         crdlats%count(1) = crdlats%varDims
00248
00249     ierr = nf90_def_var(ncid, trim(crdlats%varname), nf90_double, crdlats%varDimIDs, crdlats%varID)
00250         CALL netcdfcheckerr(ierr)
00251     ierr = nf90_put_att(ncid, crdlats%varID, 'long_name',      trim(tmpvarname))
00252         CALL netcdfcheckerr(ierr)
00253     ierr = nf90_put_att(ncid, crdlats%varID, 'standard_name', trim(tmpvarname))
00254         CALL netcdfcheckerr(ierr)
00255     ierr = nf90_put_att(ncid, crdlats%varID, 'units',        'degrees_north')
00256         CALL netcdfcheckerr(ierr)
00257     ierr = nf90_put_att(ncid, crdlats%varID, '_FillValue',   rmissv)
00258         CALL netcdfcheckerr(ierr)
00259     ierr = nf90_put_att(ncid, crdlats%varID, 'positive',    'north')

```

```

00260      CALL netcdfcheckerr(ierr)
00261
00262      ALLOCATE(crdlats%var(crdlats%varDims))
00263      crdlats%var = sfea
00264
00265      !----- Element variable
00266      !----- We need to switch the order in array for NetCDF
00267      !----- It should be: elements(nf, icnt) and NOT elements(icnt, nf)
00268      tmpvarname = 'tri'
00269      datelements%varname = trim(tmpvarname)
00270      datelements%varDimIDs(1) = vertdimid
00271      datelements%varDimIDs(2) = elemdimid
00272      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(1), len = datelements%varDims(1))
00273      CALL netcdfcheckerr(ierr)
00274      ierr = nf90_inquire_dimension(ncid, datelements%varDimIDs(2), len = datelements%varDims(2))
00275      CALL netcdfcheckerr(ierr)
00276      datelements%start(1) = 1
00277      datelements%count(1) = datelements%varDims(1)
00278      datelements%start(2) = 1
00279      datelements%count(2) = datelements%varDims(2)
00280
00281      ierr = nf90_def_var(ncid, datelements%varname, nf90_int, datelements%varDimIDs, datelements%varID)
00282      CALL netcdfcheckerr(ierr)
00283      ierr = nf90_put_att(ncid, datelements%varID,'long_name', trim(tmpvarname))
00284      CALL netcdfcheckerr(ierr)
00285      ierr = nf90_put_att(ncid, datelements%varID,'standard_name', trim(tmpvarname))
00286      CALL netcdfcheckerr(ierr)
00287      ierr = nf90_put_att(ncid, datelements%varID, 'cf_role', 'face_node_connectivity')
00288      CALL netcdfcheckerr(ierr)
00289      ierr = nf90_put_att(ncid, datelements%varID, 'start_index', 1)
00290      CALL netcdfcheckerr(ierr)
00291      ierr = nf90_put_att(ncid, datelements%varID, 'units', 'nondimensional')
00292      CALL netcdfcheckerr(ierr)
00293      ierr = nf90_put_att(ncid, datelements%varID, '_FillValue', imissv)
00294      CALL netcdfcheckerr(ierr)
00295
00296      ALLOCATE(datelements%var(datelements%varDims(1), datelements%varDims(2)))
00297      DO icnt = 1, datelements%varDims(2)
00298        DO jcnt = 1, datelements%varDims(1)
00299          datelements%var(jcnt, icnt) = nm(icnt, jcnd)
00300        END DO
00301      END DO
00302
00303      !----- Mesh variable
00304      tmpvarname = 'adcirc_mesh'
00305      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, meshvarid)
00306      CALL netcdfcheckerr(ierr)
00307
00308      ierr = nf90_put_att(ncid, meshvarid,'long_name', 'mesh_topology')
00309      CALL netcdfcheckerr(ierr)
00310      ierr = nf90_put_att(ncid, meshvarid,'standard_name', 'mesh_topology')
00311      CALL netcdfcheckerr(ierr)
00312      ierr = nf90_put_att(ncid, meshvarid, 'cf_role', 'mesh_topology')
00313      CALL netcdfcheckerr(ierr)
00314      ierr = nf90_put_att(ncid, meshvarid, 'node_coordinates', 'lon lat')
00315      CALL netcdfcheckerr(ierr)
00316      ierr = nf90_put_att(ncid, meshvarid, 'face_node_connectivity', 'element')
00317      CALL netcdfcheckerr(ierr)
00318
00319      !----- CPP (equirectangular projection or equidistant cylindrical projection) variable
00320      tmpvarname = 'projection'
00321      ierr = nf90_def_var(ncid, trim(tmpvarname), nf90_int, meshdimid, projvarid)
00322      CALL netcdfcheckerr(ierr)
00323
00324      ierr = nf90_put_att(ncid, projvarid,'long_name', 'equidistant cylindrical projection')
00325      CALL netcdfcheckerr(ierr)
00326      ierr = nf90_put_att(ncid, projvarid,'standard_name', 'CPP')
00327      CALL netcdfcheckerr(ierr)
00328      ierr = nf90_put_att(ncid, projvarid, 'node_coordinates', 'x y')
00329      CALL netcdfcheckerr(ierr)
00330      ierr = nf90_put_att(ncid, projvarid, 'lon0', slam0)
00331      CALL netcdfcheckerr(ierr)
00332      ierr = nf90_put_att(ncid, projvarid, 'lat0', sfea0)
00333      CALL netcdfcheckerr(ierr)
00334      ierr = nf90_put_att(ncid, projvarid, 'earth_radius', rearth)
00335      CALL netcdfcheckerr(ierr)
00336
00337      !----- CPP CPP x-coordinates
00338      tmpvarname = 'x'
00339      crdxcs%varname = trim(tmpvarname)
00340      crdxcs%dimID = nodedimid

```

```

00341      crdxcs%varDimIDs = nodedimid
00342      ierr = nf90_inquire_dimension(ncid, crdxcs%dimID, len = crdxcs%varDims)
00343          CALL netcdfcheckerr(ierr)
00344      crdxcs%start(1) = 1
00345      crdxcs%count(1) = crdxcs%varDims
00346
00347      ierr = nf90_def_var(ncid, trim(crdxcs%varname), nf90_double, crdxcs%varDimIDs, crdxcs%varID)
00348          CALL netcdfcheckerr(ierr)
00349      ierr = nf90_put_att(ncid, crdxcs%varID, 'long_name',      'CPP x coordinate')
00350          CALL netcdfcheckerr(ierr)
00351      ierr = nf90_put_att(ncid, crdxcs%varID, 'standard_name', 'cpp_x')
00352          CALL netcdfcheckerr(ierr)
00353      ierr = nf90_put_att(ncid, crdxcs%varID, 'units',           'm')
00354          CALL netcdfcheckerr(ierr)
00355      ierr = nf90_put_att(ncid, crdxcs%varID, '_FillValue',    rmissv)
00356          CALL netcdfcheckerr(ierr)
00357
00358      ALLOCATE(crdxcs%var(crdxcs%varDims))
00359      crdxcs%var = xclslam
00360
00361 !----- CPP y-coordinates
00362      tmpvarname = 'y'
00363      crdycs%varname = trim(tmpvarname)
00364      crdycs%dimID = nodedimid
00365      crdycs%varDimIDs = nodedimid
00366      ierr = nf90_inquire_dimension(ncid, crdycs%dimID, len = crdycs%varDims)
00367          CALL netcdfcheckerr(ierr)
00368      crdycs%start(1) = 1
00369      crdycs%count(1) = crdycs%varDims
00370
00371      ierr = nf90_def_var(ncid, trim(crdycs%varname), nf90_double, crdycs%varDimIDs, crdycs%varID)
00372          CALL netcdfcheckerr(ierr)
00373      ierr = nf90_put_att(ncid, crdycs%varID, 'long_name',      'CPP y coordinate')
00374          CALL netcdfcheckerr(ierr)
00375      ierr = nf90_put_att(ncid, crdycs%varID, 'standard_name', 'cpp_y')
00376          CALL netcdfcheckerr(ierr)
00377      ierr = nf90_put_att(ncid, crdycs%varID, 'units',           'm')
00378          CALL netcdfcheckerr(ierr)
00379      ierr = nf90_put_att(ncid, crdycs%varID, '_FillValue',    rmissv)
00380          CALL netcdfcheckerr(ierr)
00381
00382      ALLOCATE(crdycs%var(crdycs%varDims))
00383      crdycs%var = ycsfea
00384
00385 !----- Atmospheric Pressure variable
00386      tmpvarname = trim(ncvarnam_pres)
00387      datatmpres%varname      = trim(tmpvarname)
00388      datatmpres%varDimIDs(1) = nodedimid
00389      datatmpres%varDimIDs(2) = crdtimetime%dimID
00390      datatmpres%varDims(1)   = SIZE(wpress, 1)
00391      datatmpres%varDims(2)   = crdtimetime%varDims
00392      datatmpres%start(1)     = 1
00393      datatmpres%count(1)     = datatmpres%varDims(1)
00394      datatmpres%start(2)     = 1
00395      datatmpres%count(2)     = datatmpres%varDims(2)
00396
00397      ierr = nf90_def_var(ncid, trim(datatmpres%varname), nf90_double, &
00398          datatmpres%varDimIDs, datatmpres%varID)
00399          CALL netcdfcheckerr(ierr)
00400      ierr = nf90_put_att(ncid, datatmpres%varID, 'long_name',      'air pressure at sea level')
00401          CALL netcdfcheckerr(ierr)
00402      ierr = nf90_put_att(ncid, datatmpres%varID, 'standard_name', 'air_pressure_at_sea_level')
00403          CALL netcdfcheckerr(ierr)
00404      ierr = nf90_put_att(ncid, datatmpres%varID, 'units',           'Pa')
00405          CALL netcdfcheckerr(ierr)
00406      ierr = nf90_put_att(ncid, datatmpres%varID, '_FillValue',    rmissv)
00407          CALL netcdfcheckerr(ierr)
00408      ierr = nf90_put_att(ncid, datatmpres%varID, 'coordinates',   'time lat lon')
00409          CALL netcdfcheckerr(ierr)
00410      ierr = nf90_put_att(ncid, datatmpres%varID, 'location',     'node')
00411          CALL netcdfcheckerr(ierr)
00412      ierr = nf90_put_att(ncid, datatmpres%varID, 'mesh',         'adcirc_mesh')
00413          CALL netcdfcheckerr(ierr)
00414
00415 !----- Wind velocity variables
00416 ! Eastward
00417      tmpvarname = trim(ncvarnam_wndx)
00418      datwindx%varname      = trim(tmpvarname)
00419      datwindx%varDimIDs(1) = nodedimid
00420      datwindx%varDimIDs(2) = crdtimetime%dimID
00421      datwindx%varDims(1)   = SIZE(wvelx, 1)

```

```

00422      datwindx%varDims(2) = crdtime%varDims
00423      datwindx%start(1) = 1
00424      datwindx%count(1) = datwindx%varDims(1)
00425      datwindx%start(2) = 1
00426      datwindx%count(2) = datwindx%varDims(2)
00427
00428      ierr = nf90_def_var(ncid, trim(datwindx%varname), nf90_double, &
00429                           datwindx%varDimIDs, datwindx%varID)
00430      CALL netcdfcheckerr(ierr)
00431      ierr = nf90_put_att(ncid, datwindx%varID, 'long_name',      '10-m eastward wind component')
00432      CALL netcdfcheckerr(ierr)
00433      ierr = nf90_put_att(ncid, datwindx%varID, 'standard_name', 'eastward_wind')
00434      CALL netcdfcheckerr(ierr)
00435      ierr = nf90_put_att(ncid, datwindx%varID, 'units',          'm s-1')
00436      CALL netcdfcheckerr(ierr)
00437      ierr = nf90_put_att(ncid, datwindx%varID, '_FillValue',   rmissv)
00438      CALL netcdfcheckerr(ierr)
00439      ierr = nf90_put_att(ncid, datwindx%varID, 'coordinates', 'time lat lon')
00440      CALL netcdfcheckerr(ierr)
00441      ierr = nf90_put_att(ncid, datwindx%varID, 'location',    'node')
00442      CALL netcdfcheckerr(ierr)
00443      ierr = nf90_put_att(ncid, datwindx%varID, 'mesh',        'adcirc_mesh')
00444      CALL netcdfcheckerr(ierr)
00445
00446 ! Northward
00447 tmpvarname = trim(ncvarnam_wndy)
00448      datwindy%varname = trim(tmpvarname)
00449      datwindy%varDimIDs(1) = nodedimid
00450      datwindy%varDimIDs(2) = crdtime%dimID
00451      datwindy%varDims(1) = SIZE(wvely, 1)
00452      datwindy%varDims(2) = crdtime%varDims
00453      datwindy%start(1) = 1
00454      datwindy%count(1) = datwindy%varDims(1)
00455      datwindy%start(2) = 1
00456      datwindy%count(2) = datwindy%varDims(2)
00457
00458      ierr = nf90_def_var(ncid, trim(datwindy%varname), nf90_double, &
00459                           datwindy%varDimIDs, datwindy%varID)
00460      CALL netcdfcheckerr(ierr)
00461      ierr = nf90_put_att(ncid, datwindy%varID, 'long_name',      '10-m northward wind component')
00462      CALL netcdfcheckerr(ierr)
00463      ierr = nf90_put_att(ncid, datwindy%varID, 'standard_name', 'northward_wind')
00464      CALL netcdfcheckerr(ierr)
00465      ierr = nf90_put_att(ncid, datwindy%varID, 'units',          'm s-1')
00466      CALL netcdfcheckerr(ierr)
00467      ierr = nf90_put_att(ncid, datwindy%varID, '_FillValue',   rmissv)
00468      CALL netcdfcheckerr(ierr)
00469      ierr = nf90_put_att(ncid, datwindy%varID, 'coordinates', 'time lat lon')
00470      CALL netcdfcheckerr(ierr)
00471      ierr = nf90_put_att(ncid, datwindy%varID, 'location',    'node')
00472      CALL netcdfcheckerr(ierr)
00473      ierr = nf90_put_att(ncid, datwindy%varID, 'mesh',        'adcirc_mesh')
00474      CALL netcdfcheckerr(ierr)
00475
00476 =====
00477 !=====(3) Set Deflate parameters if requested by the user
00478 =====
00479 #ifdef NETCDF_CAN_DEFLATE
00480     IF (ncformat == nc4form) THEN
00481         ierr = nf90_def_var_deflate(ncid, crdlons%varID, ncshuffle, ncdeflate, nclevel)
00482         CALL netcdfcheckerr(ierr)
00483         ierr = nf90_def_var_deflate(ncid, crdlats%varID, ncshuffle, ncdeflate, nclevel)
00484         CALL netcdfcheckerr(ierr)
00485         ierr = nf90_def_var_deflate(ncid, crdxcs%varID, ncshuffle, ncdeflate, nclevel)
00486         CALL netcdfcheckerr(ierr)
00487         ierr = nf90_def_var_deflate(ncid, crdycs%varID, ncshuffle, ncdeflate, nclevel)
00488         CALL netcdfcheckerr(ierr)
00489         ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, nclevel)
00490         CALL netcdfcheckerr(ierr)
00491         ierr = nf90_def_var_deflate(ncid, datatmpres%varID, ncshuffle, ncdeflate, nclevel)
00492         CALL netcdfcheckerr(ierr)
00493         ierr = nf90_def_var_deflate(ncid, datwindx%varID, ncshuffle, ncdeflate, nclevel)
00494         CALL netcdfcheckerr(ierr)
00495         ierr = nf90_def_var_deflate(ncid, datwindy%varID, ncshuffle, ncdeflate, nclevel)
00496         CALL netcdfcheckerr(ierr)
00497     END IF
00498 #endif
00499
00500 =====
00501 !=====(4) Global metadata definitions and variables
00502 =====

```

```

00503     ierr = nf90_put_att(ncid, nf90_global, 'model', trim(prog_fullname))
00504     CALL netcdfcheckerr(ierr)
00505     ierr = nf90_put_att(ncid, nf90_global, 'version', trim(prog_version) // ' (' // trim(prog_date) //
00506     ')')
00507     CALL netcdfcheckerr(ierr)
00508     ierr = nf90_put_att(ncid, nf90_global, 'title', trim(adjustl(title)))
00509     CALL netcdfcheckerr(ierr)
00510     ierr = nf90_put_att(ncid, nf90_global, 'grid_type', 'Triangular')
00511     CALL netcdfcheckerr(ierr)
00512     ierr = nf90_put_att(ncid, nf90_global, 'agrid', trim(adjustl(agrid)))
00513     CALL netcdfcheckerr(ierr)
00514     ierr = nf90_put_att(ncid, nf90_global, 'institution', trim(adjustl(institution)))
00515     CALL netcdfcheckerr(ierr)
00516     ierr = nf90_put_att(ncid, nf90_global, 'source', trim(adjustl(source)))
00517     CALL netcdfcheckerr(ierr)
00518     ierr = nf90_put_att(ncid, nf90_global, 'history', trim(adjustl(history)))
00519     CALL netcdfcheckerr(ierr)
00520     ierr = nf90_put_att(ncid, nf90_global, 'references', trim(adjustl(references)))
00521     CALL netcdfcheckerr(ierr)
00522     ierr = nf90_put_att(ncid, nf90_global, 'comments', trim(adjustl(comments)))
00523     CALL netcdfcheckerr(ierr)
00524     ierr = nf90_put_att(ncid, nf90_global, 'host', trim(adjustl(host)))
00525     CALL netcdfcheckerr(ierr)
00526     ierr = nf90_put_att(ncid, nf90_global, 'conventions', trim(adjustl(conventions)))
00527     CALL netcdfcheckerr(ierr)
00528     ierr = nf90_put_att(ncid, nf90_global, 'contact', trim(adjustl(contact)))
00529     CALL netcdfcheckerr(ierr)

00530     CALL date_and_time(values = tvals)
00531     WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
00532     moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
moddatetimestr)
00533
00534     ierr = nf90_put_att(ncid, nf90_global, 'creation_date', trim(moddatetimestr))
00535     CALL netcdfcheckerr(ierr)
00536     ierr = nf90_put_att(ncid, nf90_global, 'modification_date', trim(moddatetimestr))
00537     CALL netcdfcheckerr(ierr)

00538 !----- Finalize the definitions in the NetCDF file
00539     ierr = nf90_enddef(ncid)
00540     CALL netcdfcheckerr(ierr)

00541 !=====
00542 !===== (5) Put the static data into the NetCDF file and then close it
00543 !=====
00544     ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
00545     CALL netcdfcheckerr(ierr)

00546     ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
00547     CALL netcdfcheckerr(ierr)

00548     ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
00549     CALL netcdfcheckerr(ierr)

00550     ierr = nf90_put_var(ncid, crdycs%varID, crdycs%var, crdycs%start, crdycs%count)
00551     CALL netcdfcheckerr(ierr)

00552     ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00553     CALL netcdfcheckerr(ierr)

00554     ierr = nf90_put_var(ncid, datatmpres%varID, datatmpres%var, datatmpres%start, datatmpres%count)
00555     CALL netcdfcheckerr(ierr)

00556     ierr = nf90_put_var(ncid, datawindx%varID, datawindx%var, datawindx%start, datawindx%count)
00557     CALL netcdfcheckerr(ierr)

00558     ierr = nf90_put_var(ncid, datawindy%varID, datawindy%var, datawindy%start, datawindy%count)
00559     CALL netcdfcheckerr(ierr)

00560 !=====
00561 !===== (6) Set all the "initialized" flags to .TRUE.
00562 !=====
00563     crdlons%initialized = .true.
00564     crdlats%initialized = .true.
00565     crdxcs%initialized = .true.
00566     crdycs%initialized = .true.
00567     datelements%initialized = .true.
00568     datatmpres%initialized = .true.
00569     datatmpres%initialized = .true.
00570     datawindx%initialized = .true.
00571     datawindy%initialized = .true.

00572
00573     myfile%fileName = adcircoutfile
00574     myfile%initialized = .true.

00575 !----- Close the NetCDF file
00576     ierr = nf90_close(ncid)
00577     CALL netcdfcheckerr(ierr)

00578     CALL unsetmessagesource()

00579
00580
00581

```

```

00582      END IF !firstCall
00583
00584  END SUBROUTINE initadcircnetcdfoutfile
00585
00586 !=====
00587
00588 !-----  

00589 ! S U B R O U T I N E   N E W   A D C I R C   N E T C D F   O U T   F I L E
00590 !
00591 !-----
00605 !
00606 SUBROUTINE newadcircnetcdfoutfile(ncID, adcircOutFile)
00607
00608     IMPLICIT NONE
00609
00610     INTEGER, INTENT(OUT)          :: ncID
00611     CHARACTER(LEN=*) , INTENT(INOUT) :: adcircOutFile
00612
00613     LOGICAL                         :: fileFound = .false.
00614     CHARACTER(LEN=FNAMELEN)          :: outFile, sys_cmd
00615     CHARACTER(LEN=14)                :: fext, date_time
00616     INTEGER                           :: pos, ierr, tvals(8)
00617
00618
00619     CALL setmessagesource("NewAdcircNetCDFOutFile")
00620
00621 !-----
00622 ! Set some variables that depend upon the type of NetCDF supported.
00623 #if defined(HAVE_NETCDF4)
00624     fext = ".nc4"
00625     ncformat = nc4form
00626 #else
00627     fext = ".nc"
00628     ncformat = nc3form
00629 #endif
00630
00631 !-----
00632 ! Remove the extension of the adcircOutFile and add a ".nc" or ".nc4"
00633 ! extension in the filename; re-define the adcircOutFile variable.
00634 pos = scan(trim(adcircoutfile), ".", back=.true.)
00635 IF (pos > 0) THEN
00636     adcircoutfile = adcircoutfile(1:pos - 1) // trim(fext)
00637 ELSE
00638     adcircoutfile = trim(adcircoutfile) // trim(fext)
00639 END IF
00640
00641 !-----
00642 ! If the adcircOutFile exists then rename it to:
00643 !   adcircOutFile-YYYYMMDDhhmmss.
00644 ! The user can remove these files afterwards.
00645 INQUIRE(file=adcircoutfile, exist=filefound)
00646 IF (filefound) THEN
00647     CALL date_and_time(values = tvals)
00648     WRITE(date_time, '(i4.4, 5i2.2)') tvals(1:3), tvals(5:7)
00649     outfile = trim(adcircoutfile) // "-" // trim(date_time)
00650     sys_cmd = "mv " // trim(adcircoutfile) // " " // trim(outfile)
00651     ierr = system(trim(sys_cmd))
00652     IF (ierr == 0) THEN
00653         WRITE(scratchmessage, '(a)') 'Renamed: ' // trim(adcircoutfile) // ' to ' // trim(outfile)
00654         CALL logmessage(info, scratchmessage)
00655         filefound = .false.
00656     ELSE
00657         WRITE(scratchmessage, '(a)') 'Could not rename the file ' // trim(adcircoutfile) // ' to ' //
00658             trim(outfile)
00659         CALL logmessage(error, scratchmessage)
00660     END IF
00661 END IF
00662 IF (filefound) THEN
00663     WRITE(scratchmessage, '(a)') 'The NetCDF output file ' // trim(adcircoutfile) // ' exists. Remove the
00664     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
00673     mode.'
00674     CALL logmessage(info, scratchmessage)
00675

```

```

00674 ! Create the NetCDF file
00675 ierr = nf90_create(adcircoutfile, ncformat, ncid)
00676 CALL netcdfcheckerr(ierr)
00677
00678 CALL unsetmessagesource()
00679
00680 END SUBROUTINE newadcircnetcdfoutfile
00681
00682 !=====
00683
00684 !-----  

00685 ! S U B R O U T I N E   N E T C D F   C H E C K   E R R
00686 !-----  

00703 !-----  

00704 SUBROUTINE base_ncdfcheckerr(ierr, file, line)
00705
00706 IMPLICIT NONE
00707
00708 INTEGER, INTENT(IN) :: ierr
00709 CHARACTER(LEN=*) , INTENT(IN) :: file
00710 INTEGER, INTENT(IN) :: line
00711
00712 CHARACTER(LEN=1024) :: tmpSTR
00713
00714 CALL setmessagesource("NetCDFCheckErr")
00715
00716 IF (ierr /= nf90_noerr) THEN
00717   CALL allmessage(error, nf90_strerror(ierr))
00718   WRITE(tmpstr, '(a, a, i5)') trim(file), ':', line
00719   CALL allmessage(info, tmpstr)
00720   CALL netcdfterminate()
00721 END IF
00722
00723 CALL unsetmessagesource()
00724
00725 END SUBROUTINE base_ncdfcheckerr
00726
00727 !=====
00728
00729 !-----  

00730 ! S U B R O U T I N E   N E T C D F   T E R M I N A T E
00731 !-----  

00739 !-----  

00740 SUBROUTINE netcdfterminate()
00741
00742 USE version
00743
00744 IMPLICIT NONE
00745
00746 CALL setmessagesource("NetCDFTerminate")
00747
00748 CALL allmessage(info, trim(adjustl(prog_name)) // " Terminating.")
00749
00750 CALL exit(1)
00751
00752 CALL unsetmessagesource()
00753
00754 END SUBROUTINE netcdfterminate
00755
00756 !=====
00757
00758
00759 !-----  

00760 ! S U B R O U T I N E   W R I T E   N E T C D F   R E C O R D
00761 !-----  

00774 !
00775 SUBROUTINE writenetcdfrecord(adcircOutFile, timeLoc)
00776
00777 USE timedateutils, ONLY : gettimeconvsec
00778
00779 IMPLICIT NONE
00780
00781 CHARACTER(LEN=*) , INTENT(IN) :: adcircOutFile
00782 INTEGER, INTENT(IN) :: timeLoc
00783
00784 INTEGER :: ncID, ierr, nodes
00785 INTEGER :: start(2), kount(2)
00786 REAL(SZ) :: currTime
00787
00788
00789 CALL setmessagesource("WriteNetCDFRecord")

```

```

00790
00791     ierr = nf90_open(trim(adcircoutfile), nf90_write, ncid)
00792     CALL netcdfcheckerr(ierr)
00793
00794     ! Set up the 2D netcdf data extents
00795     ierr = nf90_inquire_dimension(ncid, nodedimid, len = nodes)
00796     start(1) = 1
00797     start(2) = timeloc
00798     kount(1) = nodes
00799     kount(2) = 1
00800
00801     !----- This is the for the time record
00802     currtime = times(timeloc) * gettimeconvsec(unittime, 1)
00803     ierr = nf90_put_var(ncid, crdtime%varID, currtime, (/timeloc/))
00804     CALL netcdfcheckerr(ierr)
00805
00806     !----- This is the for the atmospheric pressure record
00807     ierr = nf90_put_var(ncid, datatmpres%varID, wpress, start, kount)
00808     CALL netcdfcheckerr(ierr)
00809
00810     !----- This is the for the 10-m longitudinal wind speed component
00811     ierr = nf90_put_var(ncid, datwindx%varID, wvelx, start, kount)
00812     CALL netcdfcheckerr(ierr)
00813
00814     !----- This is the for the 10-m meridional wind speed component
00815     ierr = nf90_put_var(ncid, datwindy%varID, wvely, start, kount)
00816     CALL netcdfcheckerr(ierr)
00817
00818     ! Close netCDF file
00819     ierr = nf90_close(ncid)
00820     CALL netcdfcheckerr(ierr)
00821
00822     CALL unsetmessagesource()
00823
00824 END SUBROUTINE writenetcdfrecord
00825
00826 !=====
00827
00828 !-----!
00829 ! S U B R O U T I N E   S E T   R E C O R D   C O U N T E R   A N D   S T O R E   T I M E
00830 !-----!
00831 !-----!
00832
00833 SUBROUTINE setrecordcounterandstoretime(ncID, f, t)
00834
00835     IMPLICIT NONE
00836
00837     INTEGER, INTENT(IN) :: ncID
00838     TYPE(filedata_t), INTENT(INOUT) :: f
00839     TYPE(timedata_t), INTENT(INOUT) :: t
00840
00841     REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
00842     LOGICAL :: timeFound ! true if current time is in array of stored times
00843
00844     INTEGER :: ndim ! number of dimensions in the netcdf file
00845     INTEGER :: nvar ! number of variables in the netcdf file
00846     INTEGER :: natt ! number of attributes in the netcdf file
00847
00848     INTEGER :: counti(1), starti(1)
00849     INTEGER :: ierr ! success or failure of netcdf call
00850     INTEGER :: i ! loop counter
00851
00852
00853     CALL setmessagesource("SetRecordCounterAndStoreTime")
00854
00855     ! Inquire the time variable
00856     ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
00857     CALL netcdfcheckerr(ierr)
00858
00859     ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
00860     CALL netcdfcheckerr(ierr)
00861
00862     ierr = nf90_inq_varid(ncid, 'time', t%timeID)
00863     CALL netcdfcheckerr(ierr)
00864
00865     ! Determine the relationship between the current simulation time
00866     ! and the time array stored in the netcdf file. Set the record
00867     ! counter based on this relationship.
00868     IF (f%fileRecCounter /= 0) THEN
00869         ALLOCATE(storedtimes(f%fileRecCounter))
00870         ierr = nf90_get_var(ncid, t%timeID, storedtimes)
00871         CALL netcdfcheckerr(ierr)
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888

```

```

00890      timefound = .false.
00891
00892      DO i = 1, f%fileRecCounter
00893          IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
00894              timefound = .true.
00895          EXIT
00896      ENDIF
00897  END DO
00898
00899      IF (timefound .EQV. .false.) THEN
00900          ! Increment the record counter so that we can store data at the
00901          ! next location in the netcdf file (i.e., all of the times
00902          ! in the netcdf file were found to be earlier than the current
00903          ! adcirc simulation time).
00904          f%fileRecCounter = f%fileRecCounter + 1
00905
00906      ELSE
00907          ! set the counter at the index that reflects the
00908          ! current time within the netcdf file (or is between two times
00909          ! found in the netcdf file).
00910          ! WARNING: all subsequent data will remain in the file, we
00911          ! are just overwriting it ... if we don't overwrite all of it,
00912          ! the pre-existing data will still be there, which is probably
00913          ! not what the user intended ... but apparently there is no
00914          ! way to delete data from netcdf files:
00915          ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
00916          scratchformat = ('Overwriting pre-existing data in netcdf file ",a,' // &
00917                          '" for time=",f17.8,". ' // 'Subsequent data in netcdf file remain unchanged.')
00918          WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
00919          CALL allmessage(info, scratchmessage)
00920          f%fileRecCounter = i
00921
00922      ENDIF
00923
00924      DEALLOCATE(storedtimes)
00925
00926      ELSE
00927          ! set the counter at 1 so we can record our first time value
00928          f%fileRecCounter = 1
00929
00930      ENDIF
00931
00932      ! Store simulation time in netcdf file
00933      starti(1) = f%fileRecCounter
00934      counti(1) = t%timelen
00935      ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
00936      CALL netcdfcheckerr(ierr)
00937
00938      CALL unsetmessagesource()
00939
00940  END SUBROUTINE setrecordcounterandstoretime
00941
00942 !=====
00943 END MODULE pahm_netcdfio

```

20.31 /home/takis/CSDL/parwinds-doc/src/pahm.F90 File Reference

Main PaHM program, calls Init, Run and Finalize procedures.

Functions/Subroutines

- program [pahm](#)

20.31.1 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](#).

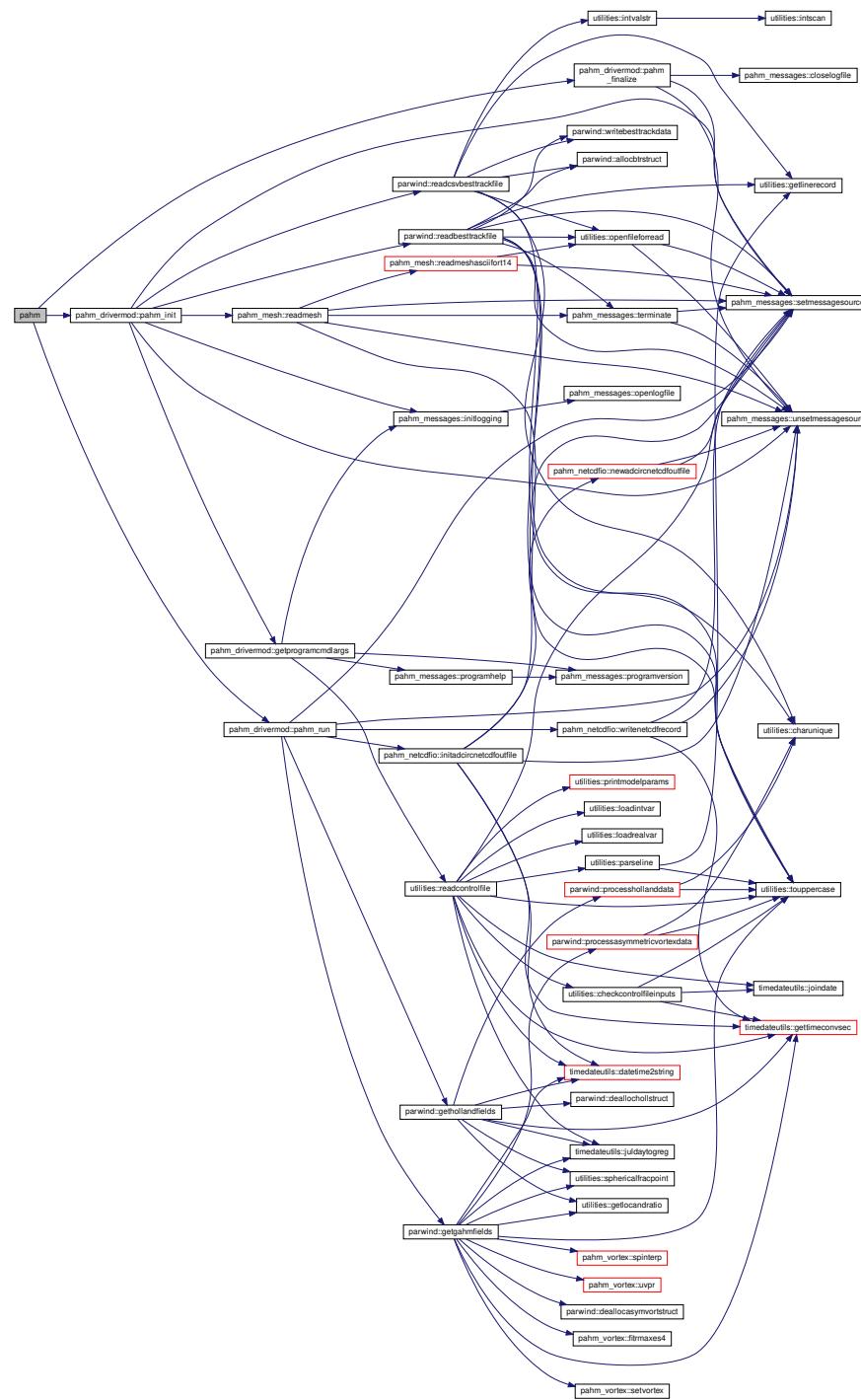
20.31.2 Function/Subroutine Documentation

20.31.2.1 `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:



20.32 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
```

20.33 /home/takis/CSDL/parwinds-doc/src/parwind.F90 File Reference

Data Types

- type `parwind::besttrackdata_t`
- type `parwind::hollanddata_t`
- type `parwind::asymmetricvortexdata_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(s) (GetHollandFields).
- subroutine `parwind::processasymmetricvortexdata (idTrFile, strOut, status)`
Subroutine to support the asymmetric vortex model(s) (Holland or otherwise).
- subroutine `parwind::gethollandfields (timeIDX)`
Calculates wind velocities and MSL pressures at the mesh nodes from the P-W Holland Wind model.
- subroutine `parwind::getgahmfields (timeIDX)`
Calculates wind velocities and MSL pressures at the mesh nodes from the GAHM Wind model.
- subroutine `parwind::writebesttrackdata (inpFile, trackStruc, suffix)`
Outputs the post-processed best track data to file.
- subroutine `parwind::writeasymmetricvortexdata (inpFile, trackStruc, suffix)`
Outputs the post-processed best track data to file.
- subroutine `parwind::allocbtrstruct (str, nRec)`
Subroutine to allocate memory for a best track structure.
- subroutine `parwind::deallocbtrstruct (str)`

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

- subroutine **parwind::allochollstruct** (str, nRec)

Subroutine to allocate memory for a holland structure.

- subroutine **parwind::deallochollstruct** (str)

Subroutine to deallocate memory of an allocated holland structure.

- subroutine **parwind::allocasymvortstruct** (str, nRec)

Subroutine to allocate memory for an asymmetric vortex structure.

- subroutine **parwind::deallocasymvortstruct** (str)

Subroutine to deallocate memory of an allocated asymmetric vortex structure.

Variables

- logical **parwind::geostrophicswitch** = .TRUE.
- integer **parwind::geofactor** = 1
- integer **parwind::method** = 4
- integer **parwind::approach** = 2
- integer, parameter, private **parwind::stormnamelen** = 10
- type(besttrackdata_t), dimension(:), allocatable **parwind::besttrackdata**
- type(hollanddata_t), dimension(:), allocatable **parwind::holstru**
- type(asymmetricvortexdata_t), dimension(:), allocatable **parwind::asyvortstru**

20.33.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file **parwind.F90**.

20.34 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. !PV shouldn't be a user input?  

00025 INTEGER :: geofactor = 1 !turn on or off geostrophic balance !PV shouldn't be a user  

input?  

00026 INTEGER :: method = 4, approach = 2 !PV shouldn't be a user input?  

00027  

00028 INTEGER, PARAMETER, PRIVATE :: stormnamelen = 10  

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=FNAMELEN)          :: 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            :: cnum()          ! 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
00044      best track: 00 - 99
00045      CHARACTER(LEN=4), ALLOCATABLE   :: tech()           ! acronym for each objective technique or CARQ or
00045      WRNG,
00046
00047      INTEGER, ALLOCATABLE            :: tau()            ! BEST for best track, up to 4 chars.
00047      best-track,                   ! forecast period: -24 through 240 hours, 0 for
00048
00049      INTEGER, ALLOCATABLE            :: intlat()         ! negative taus used for CARQ and WRNG records.
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
00054      kts
00055      INTEGER, ALLOCATABLE            :: intmslp()        ! minimum sea level pressure, 850 - 1050 mb
00056      CHARACTER(LEN=2), ALLOCATABLE   :: ty()              ! Highest level of tc development:
00056      DB - disturbance,
00056      TD - tropical depression,
00056      TS - tropical storm,
00056      TY - typhoon,
00056      ST - super typhoon,
00056      TC - tropical cyclone,
00056      HU - hurricane,
00056      SD - subtropical depression,
00056      SS - subtropical storm,
00056      EX - extratropical systems,
00056      PT - post tropical,
00056      IN - inland,
00056      DS - dissipating,
00056      LO - low,
00056      WV - tropical wave,
00056      ET - extrapolated,
00056      MD - monsoon depression,
00056      XX - unknown.
00056      ! wind intensity for the radii defined in this
00075      INTEGER, ALLOCATABLE            :: rad()            ! radius code:
00075      record: 34, 50 or 64 kt
00075      CHARACTER(LEN=3), ALLOCATABLE   :: windcode()       ! AAA - full circle
00075
00079      INTEGER, ALLOCATABLE            :: intrad1()        ! NEQ, SEQ, SWQ, NWQ - quadrant
00079      intensity, or radius of       ! if full circle, radius of specified wind
00080
00080      WINDCODE. 0 - 999 n mi
00081      INTEGER, ALLOCATABLE            :: intrad2()        ! first quadrant wind intensity as specified by
00081      of 2nd quadrant wind
00082
00082      mi
00083      INTEGER, ALLOCATABLE            :: intrad3()        ! if full circle this field not used, or radius
00083      of 3rd quadrant wind
00084
00084      mi
00085      INTEGER, ALLOCATABLE            :: intrad4()        ! if full circle this field not used, or radius
00085      of 4th quadrant wind
00086
00086      mi
00087      INTEGER, ALLOCATABLE            :: intpouter()      ! intensity as specified by WINDCODE. 0 - 999 n
00087      isobar, 900 - 1050 mb
00088      INTEGER, ALLOCATABLE            :: introuter()      ! pressure in millibars of the last closed
00088
00089      INTEGER, ALLOCATABLE            :: intrmw()         ! radius of the last closed isobar, 0 - 999 n mi
00090      INTEGER, ALLOCATABLE            :: gusts()          ! radius of max winds, 0 - 999 n mi
00091      INTEGER, ALLOCATABLE            :: eye()            ! gusts, 0 - 999 kt
00091      CHARACTER(LEN=3), ALLOCATABLE   :: subregion()      ! eye diameter, 0 - 120 n mi
00091
00092      ! subregion code: W,A,B,S,P,C,E,L,Q
00092
00093      ! A - Arabian Sea
00093      ! B - Bay of Bengal
00093      ! C - Central Pacific
00093      ! E - Eastern Pacific
00093      ! L - Atlantic
00093      ! P - South Pacific (135E - 120W)
00093      ! Q - South Atlantic
00093      ! S - South IO (20E - 135E)

```

```

00101      INTEGER, ALLOCATABLE :: maxseas(:)          ! W - Western Pacific
00102      CHARACTER(LEN=3), ALLOCATABLE   :: initials(:) ! max seas: 0 - 999 ft
00103      ! forecaster's initials used for tau 0 WRNG or
00104      ! OFCL, up to 3 chars
00105      INTEGER, ALLOCATABLE :: dir(:)           ! storm direction, 0 - 359 degrees
00106      INTEGER, ALLOCATABLE :: intspeed(:)        ! storm speed, 0 - 999 kts
00107      CHARACTER(LEN=STORMNAMELEN), ALLOCATABLE & ! literal storm name, number, NONAME or INVEST,
00108      ! or TCcyx where:
00109      !          ! cy = Annual cyclone number 01 - 99
00110      !          ! x = Subregion code: W,A,B,S,P,C,E,L,Q.
00111      INTEGER, ALLOCATABLE :: cyclenum(:)         ! the cycle number
00112      !----- converted data from the above values (if needed)
00113      INTEGER, DIMENSION(:), ALLOCATABLE :: year, month, day, hour
00114      REAL(sz), DIMENSION(:), ALLOCATABLE :: lat, lon
00115  END TYPE besttrackdata_t
00116
00117  ! Array of info about the best track data (extension to use multiple storms)
00118  TYPE(besttrackdata_t), ALLOCATABLE :: besttrackdata(:)
00119
00120 !-----
00121 ! The HollandData_T structure holds all required data for the Holland model
00122 ! The data are filtered to only include unique DTGs
00123 !-----
00124  TYPE hollanddata_t
00125      CHARACTER(LEN=FNAMELEN)      :: filename      ! full path to the best track file
00126      CHARACTER(LEN=10)           :: thisstorm     ! the name of the "named" storm
00127      LOGICAL                   :: loaded = .false. ! .TRUE. if we have loaded the data from file
00128      INTEGER                    :: numrec       ! number of records in the structure
00129
00130      CHARACTER(LEN=2),          ALLOCATABLE :: basin(:)      ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00131      INTEGER, ALLOCATABLE       :: stormnumber(:) ! annual cyclone number: 1 - 99
00132      CHARACTER(LEN=10),          ALLOCATABLE :: dtg(:)       ! warning Date-Time-Group (DTG), YYYYMMDDHH
00133      INTEGER, DIMENSION(:),    ALLOCATABLE :: year, month, day, hour
00134      REAL(sz), ALLOCATABLE     :: casttime(:)    ! converted to decimal E/N (lon, lat)
00135      CHARACTER(LEN=4),          ALLOCATABLE :: casttype(:) ! BEST, OFCL, CALM, ...
00136      INTEGER, ALLOCATABLE     :: fcstinc(:)    ! forecast period: -24 through 240 hours, 0
00137      ! for best-track
00138      INTEGER, DIMENSION(:),    ALLOCATABLE :: ilat, ilon   ! latitude, longitude for the GTD
00139      REAL(sz), DIMENSION(:),    ALLOCATABLE :: lat, lon     ! converted to decimal E/N (lon, lat)
00140
00141      INTEGER,                  ALLOCATABLE :: ispeed(:)   ! maximum sustained wind speed in knots: 0 -
00142      ! 300 kts
00143      REAL(sz),                  ALLOCATABLE :: speed(:)    ! converted from kts to m/s
00144      INTEGER,                  ALLOCATABLE :: icpress(:)   ! minimum sea level pressure, 850 - 1050 mb
00145      REAL(sz),                  ALLOCATABLE :: cpres(:)     ! converted to Pa
00146
00147      INTEGER,                  ALLOCATABLE :: irrp(:)     ! radius of the last closed isobar, 0 - 999 n
00148      ! mi
00149      REAL(sz),                  ALLOCATABLE :: rrp(:)      ! converted from nm to m
00150      INTEGER,                  ALLOCATABLE :: irmw(:)     ! radius of max winds, 0 - 999 n mi
00151      REAL(sz),                  ALLOCATABLE :: rmw(:)      ! converted from nm to m
00152
00153      REAL(sz), DIMENSION(:),   ALLOCATABLE :: cprdt      ! central pressure intensity change (Pa / h)
00154      REAL(sz), DIMENSION(:),   ALLOCATABLE :: trvx, trvy   ! translational velocity components (x, y) of
00155      ! the
00156  END TYPE hollanddata_t                                ! moving hurricane (m/s)
00157
00158  TYPE(hollanddata_t), ALLOCATABLE :: holstru(:)      ! array of Holland data structures
00159
00160 !-----
00161 ! The AsymmetricVortexData_T structure holds all required data for
00162 ! the asymmetric vortex models. The data are filtered to only include unique DTGs
00163 !-----
00164  TYPE asymmetricvortexdata_t
00165      CHARACTER(LEN=FNAMELEN)      :: filename      ! full path to the best track file
00166      CHARACTER(LEN=10)           :: thisstorm     ! the name of the "named" storm
00167      LOGICAL                   :: loaded = .false. ! .TRUE. if we have loaded the data from file
00168      INTEGER                    :: numrec       ! number of records in the structure
00169
00170      CHARACTER(LEN=2),          ALLOCATABLE :: basin(:)      ! basin, e.g. WP, IO, SH, CP, EP, AL, LS
00171      INTEGER, ALLOCATABLE       :: stormnumber(:) ! annual cyclone number: 1 - 99
00172      CHARACTER(LEN=10),          ALLOCATABLE :: dtg(:)       ! warning Date-Time-Group (DTG), YYYYMMDDHH
00173      INTEGER, DIMENSION(:),    ALLOCATABLE :: year, month, day, hour
00174      REAL(sz), ALLOCATABLE     :: casttime(:)    ! time in seconds from the reference date of
00175      ! the simulation

```

```

00175      INTEGER, ALLOCATABLE          :: casttypenum(:)      ! objective technique sorting number, minutes
00176      CHARACTER(LEN=4),           ALLOCATABLE :: casttype(:)
00177      INTEGER,                  ALLOCATABLE :: fcstinc(:)
00178      for best-track
00179      INTEGER, DIMENSION(:), ALLOCATABLE :: ilat, ilon      ! latitude, longitude for the GTD
00180      REAL(sz),   DIMENSION(:), ALLOCATABLE :: lat, lon      ! converted to decimal E/N (lon, lat)
00181      CHARACTER(LEN=1), ALLOCATABLE      :: ns(:), ew(:)      ! N/S and E/W character
00182
00183      INTEGER,                  ALLOCATABLE :: ispeed(:)    ! maximum sustained wind speed in knots: 0 -
00184      300 kts
00185      REAL(sz),                ALLOCATABLE :: speed(:)     ! converted from kts to m/s
00186      INTEGER,                  ALLOCATABLE :: icpress(:)    ! minimum sea level pressure, 850 - 1050 mb
00187      REAL(sz),                ALLOCATABLE :: cpres(:)     ! converted to Pa
00188
00189      CHARACTER(LEN=2), ALLOCATABLE      :: ty(:)          ! Highest level of tc development (see best
track structure)
00190
00191      INTEGER, ALLOCATABLE          :: ivr(:)          ! wind intensity for the radii defined in this
record: 34, 50 or 64 kt
00192      CHARACTER(LEN=3), ALLOCATABLE      :: windcode(:)    ! radius code: AAA - full circle, NEQ, SEQ,
SWQ, NWQ - quadrant
00193      INTEGER, ALLOCATABLE          :: ir(:, :)        ! if full circle, radius of specified wind
intensity, or radius of
00194      quadrant, 4: 4th quadrant      ! 1: first quadrant, 2: 2nd quadrant, 3: 3rd
00195
00196      INTEGER,                  ALLOCATABLE :: iprp(:)    ! pressure in millibars of the last closed
isobar, 900 - 1050 mb
00197      REAL(sz),                ALLOCATABLE :: prp(:)     ! converted to Pa
00198
00199      INTEGER,                  ALLOCATABLE :: irrp(:)    ! radius of the last closed isobar, 0 - 999 n
mi
00200      REAL(sz),                ALLOCATABLE :: rrp(:)     ! converted from nm to m
00201
00202      INTEGER,                  ALLOCATABLE :: irmw(:)    ! radius of max winds, 0 - 999 n mi
00203      REAL(sz),                ALLOCATABLE :: rmw(:)     ! converted from nm to m
00204
00205      INTEGER, ALLOCATABLE          :: gusts(:)        ! gusts, 0 - 999 kt
00206      INTEGER, ALLOCATABLE          :: eye(:)         ! eye diameter, 0 - 120 n mi
00207      CHARACTER(LEN=3), ALLOCATABLE      :: subregion(:)  ! subregion code: W,A,B,S,P,C,E,L,Q
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217      INTEGER, ALLOCATABLE          :: maxseas(:)    ! max seas: 0 - 999 ft
00218      CHARACTER(LEN=3), ALLOCATABLE      :: initials(:)  ! forecaster's initials used for tau 0 WRNG or
OFCL, up to 3 chars
00219
00220      REAL(sz), DIMENSION(:), ALLOCATABLE :: trvx, trvy  ! translational velocity components (x, y) of
the
00221
00222
00223      INTEGER, ALLOCATABLE          :: idir(:)        ! storm direction, 0 - 359 degrees
00224      REAL(sz), ALLOCATABLE          :: dir(:)         ! storm direction, 0 - 359 degrees
00225      INTEGER, ALLOCATABLE          :: istormspeed(:)  ! storm speed, 0 - 999 kts
00226      REAL(sz), ALLOCATABLE          :: stormspeed(:)  ! converted from kts to m/s
00227      CHARACTER(LEN=STORMNAMELEN), ALLOCATABLE &
00228                      :: stormname(:)    ! literal storm name, number, NONAME or
INVEST, or TCcyx where:
00229
00230
00231
00232      ! extended/asymmetric vortex data
00233      INTEGER,                  :: ncycles
00234      INTEGER, DIMENSION(:), ALLOCATABLE :: numcycle
00235      INTEGER, DIMENSION(:), ALLOCATABLE :: isotachspercycle
00236      INTEGER, DIMENSION(:, :), ALLOCATABLE :: quadflag
00237      REAL(sz), DIMENSION(:, :), ALLOCATABLE :: rmaxw
00238      REAL(sz), DIMENSION(:, :), ALLOCATABLE :: hollb
00239      REAL(sz), DIMENSION(:, :), ALLOCATABLE :: holls
00240      REAL(sz), DIMENSION(:, :), ALLOCATABLE :: vmaxesbl
00241 END TYPE asymmetricvortexdata_t
00242

```

```

00243  TYPE(asymmetricvortexdata_t), ALLOCATABLE :: asyvortstru(:) ! array of asymmetric vortex data structures
00244
00245
00246  CONTAINS
00247
00248
00249 !-----+
00250 !  S U B R O U T I N E   R E A D   B E S T   T R A C K   F I L E
00251 !-----+
00252 !-----+
00253
00254 SUBROUTINE readbesttrackfile()
00255
00256
00257     USE pahm_global, ONLY : lun_btrk, lun_btrk1, nbtrfiles, besttrackfilename
00258     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
00259     USE timedateutils, ONLY : timeconv
00260     USE sortutils, ONLY : arth, indexx, arrayequal
00261
00262     IMPLICIT NONE
00263
00264
00265     CHARACTER(LEN=FNAMELEN)      :: inpFile
00266     CHARACTER(LEN=512)          :: inpLine, line
00267     CHARACTER(LEN=512)          :: fmtStr
00268
00269     INTEGER                      :: lenLine
00270     INTEGER                      :: nLines           ! Number of lines counter
00271     INTEGER                      :: iFile, iCnt        ! loop counters
00272     INTEGER                      :: iUnit, errIO, ios, status
00273
00274     !CHARACTER(LEN=4)            :: castType
00275
00276     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00277     INTEGER, ALLOCATABLE         :: idxArrStr(:)
00278     INTEGER                      :: nUnique, maxCnt, kCnt, kMax
00279
00280     INTEGER, ALLOCATABLE         :: idx0(:), idx1(:)
00281     REAL(SZ)                    :: tmpFcstTime, refFcstTime
00282
00283 !-----+ Initialize variables
00284     iunit = lun_btrk
00285     errio = 0
00286
00287 !fmtStr = '(a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x,
00288 a2, '
00289     fmtstr = '(a2, 1x, i3, 2x, a10, 1x, i3, 2x, a4, 2x, i3, 2x, i3, a1, 1x, i5, a1, 2x, i3, 2x, i4, 2x,
00290 a2, '
00291     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i4, 2x, i3, 2x, i3, 2x, i3, '
00292     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, 2(i3, 2x), a11'
00293 !-----+
00294
00295     CALL setmessagesource("ReadBestTrackFile")
00296
00297 ! Allocate the best track structure array. This structure holds all the
00298 ! input values for the storm track as read in from the track input file
00299 ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00300 ! (as appropriate).
00301     ALLOCATE(besttrackdata(nbtrfiles))
00302
00303 ! This is the main loop. We loop through all the best track files
00304 ! (user input)
00305     DO ifile = 1, nbtrfiles
00306         inpfile = besttrackfilename(ifile)
00307
00308         CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)
00309
00310         IF (errio /= 0) THEN
00311             WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
00312             CALL allmessage(error, scratchmessage)
00313
00314             CALL unsetmessagesource()
00315
00316             CALL terminate()
00317
00318         ELSE
00319             WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
00320             CALL logmessage(info, scratchmessage)
00321
00322             END IF
00323
00324             besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00325             besttrackdata(ifile)%thisStorm = ""
00326             besttrackdata(ifile)%loaded = .false.
00327             besttrackdata(ifile)%numRec = -1
00328
00329
00330
00331
00332
00333
00334

```

```

00335 ! Count the number of non-empty or commented out lines in the file.
00336 ! Comments are considered those lines with the first non-blank character of "!" or "#"
00337 nlines = 0
00338 DO
00339   READ(unit=iunit, fmt='(a)', err=10, END=5, IOSTAT=errIO) inpline
00340
00341   lenline = getlinerecord(inpline, line)
00342   IF (lenline /= 0) nlines = nlines + 1
00343 END DO
00344 5 rewind(unit=iunit)
00345
00346 ! Array allocation in the structure bestTrackData
00347 CALL allocbtrstruct(besttrackdata(ifile), nlines)
00348
00349 icnt = 0
00350 kcnt = 0
00351 DO WHILE (.true.)
00352   READ(unit=iunit, fmt='(a)', err=10, END=20, IOSTAT=errIO) inpline
00353
00354   lenline = getlinerecord(inpline, line)
00355
00356   IF (lenline /= 0) THEN
00357     icnt = icnt + 1
00358     READ(line, fmt=fmtstr, err=11, iostat=ios)
00359       besttrackdata(ifile)%basin(icnt),      besttrackdata(ifile)%cyNum(icnt),
00360       besttrackdata(ifile)%dtg(icnt),        besttrackdata(ifile)%techNum(icnt),
00361       besttrackdata(ifile)%tech(icnt),       besttrackdata(ifile)%tau(icnt),
00362       besttrackdata(ifile)%intLat(icnt),    besttrackdata(ifile)%ns(icnt),
00363       besttrackdata(ifile)%intLon(icnt),    besttrackdata(ifile)%ew(icnt),
00364       besttrackdata(ifile)%intVMax(icnt),   besttrackdata(ifile)%intMslp(icnt),
00365       besttrackdata(ifile)%ty(icnt),        besttrackdata(ifile)%rad(icnt),
00366       besttrackdata(ifile)%windCode(icnt),  besttrackdata(ifile)%intRad1(icnt),
00367       besttrackdata(ifile)%intRad2(icnt),  besttrackdata(ifile)%intRad3(icnt),
00368       besttrackdata(ifile)%intRad4(icnt),  besttrackdata(ifile)%intPOuter(icnt),
00369       besttrackdata(ifile)%intROuter(icnt), besttrackdata(ifile)%intRmw(icnt),
00370       besttrackdata(ifile)%gusts(icnt),    besttrackdata(ifile)%eye(icnt),
00371       besttrackdata(ifile)%subregion(icnt), besttrackdata(ifile)%maxseas(icnt),
00372       besttrackdata(ifile)%initials(icnt), besttrackdata(ifile)%dir(icnt),
00373       besttrackdata(ifile)%intSpeed(icnt),  besttrackdata(ifile)%stormName(icnt)
00374
00375 !----- This is for the cycleNum, the last column we consider
00376 IF (icnt == 1) THEN
00377   kcnt = icnt
00378   besttrackdata(ifile)%cycleNum(icnt) = kcnt
00379 ELSE
00380   kcnt = kcnt + 1
00381   IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00382     besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00383   kcnt = kcnt - 1
00384 ELSE
00385   besttrackdata(ifile)%cycleNum(icnt) = kcnt
00386 END IF
00387 END IF
00388 !-----
00389
00390 !----- Convert lat/lon values to S/N and W/E notations
00391 IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
00392   besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00393 ELSE
00394   besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00395 END IF
00396
00397 IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00398   besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00399 ELSE
00400   besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00401 END IF
00402 !-----
00403
00404 !----- Get the year, month, day, hour from the DGT string
00405 READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00406 besttrackdata(ifile)%year(icnt)
00407   IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00408   READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00409 besttrackdata(ifile)%month(icnt)
00410   IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00411   READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00412 besttrackdata(ifile)%day(icnt)
00413   IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00414   READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00415 besttrackdata(ifile)%hour(icnt)

```

```

00412      IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00413      !-----
00414      END IF
00415  END DO
00416
00417 10 IF (erro /= 0) THEN
00418      WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00419      ' , while processing line: ' // trim(adjustl(inpline))
00420      CALL allmessage(error, scratchmessage)
00421
00422      CLOSE(iunit)
00423
00424      CALL unsetmessagesource()
00425
00426      CALL terminate()
00427  END IF
00428
00429 11 IF (ios /= 0) THEN
00430      WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00431      ' , while processing line: ' // trim(adjustl(line))
00432      CALL allmessage(error, scratchmessage)
00433
00434      CLOSE(iunit)
00435
00436      CALL unsetmessagesource()
00437
00438      CALL terminate()
00439  END IF
00440
00441 20 CLOSE(iunit)
00442
00443 besttrackdata(ifile)%thisStorm = "
00444 besttrackdata(ifile)%loaded    = .true.
00445 besttrackdata(ifile)%numRec   = nlines
00446
00447 !-----
00448 ! Get the unique storm name and store it in the thisStorm string
00449 ALLOCATE(chkarrstr(nlines))
00450 ALLOCATE(idxarrstr(nlines))
00451
00452 nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00453
00454 maxcnt = -1
00455 DO kcnt = 1, nunique
00456     kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00457     IF (kmax > maxcnt) THEN
00458         maxcnt = kmax
00459         besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00460     END IF
00461 END DO
00462
00463 DEALLOCATE(chkarrstr)
00464 DEALLOCATE(idxarrstr)
00465 !-----
00466 !-----
00467 ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00468 ! stored in ascending order
00469 ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00470 ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00471
00472 CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00473
00474 IF (status /= 0) THEN
00475     CALL unsetmessagesource()
00476
00477     CALL terminate()
00478 END IF
00479
00480 ! Create the index array to be used in the comparison below
00481 idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00482
00483 IF (.NOT. arrayequal(idx0, idx1)) THEN
00484     besttrackdata(ifile)%basin      = besttrackdata(ifile)%basin(idx1)
00485     besttrackdata(ifile)%cyNum     = besttrackdata(ifile)%cyNum(idx1)
00486     besttrackdata(ifile)%dtg       = besttrackdata(ifile)%dtg(idx1)
00487     besttrackdata(ifile)%techNum  = besttrackdata(ifile)%techNum(idx1)
00488     besttrackdata(ifile)%tech     = besttrackdata(ifile)%tech(idx1)
00489     besttrackdata(ifile)%tau      = besttrackdata(ifile)%tau(idx1)
00490     besttrackdata(ifile)%intLat  = besttrackdata(ifile)%intLat(idx1)
00491
00492

```

```

00493     besttrackdata(ifile)%ns          = besttrackdata(ifile)%ns(idx1)
00494     besttrackdata(ifile)%intLon    = besttrackdata(ifile)%intLon(idx1)
00495     besttrackdata(ifile)%ew        = besttrackdata(ifile)%ew(idx1)
00496     besttrackdata(ifile)%intVMax   = besttrackdata(ifile)%intVMax(idx1)
00497     besttrackdata(ifile)%intMslp   = besttrackdata(ifile)%intMslp(idx1)
00498     besttrackdata(ifile)%ty        = besttrackdata(ifile)%ty(idx1)
00499     besttrackdata(ifile)%rad       = besttrackdata(ifile)%rad(idx1)
00500     besttrackdata(ifile)%windCode  = besttrackdata(ifile)%windCode(idx1)
00501     besttrackdata(ifile)%intRad1   = besttrackdata(ifile)%intRad1(idx1)
00502     besttrackdata(ifile)%intRad2   = besttrackdata(ifile)%intRad2(idx1)
00503     besttrackdata(ifile)%intRad3   = besttrackdata(ifile)%intRad3(idx1)
00504     besttrackdata(ifile)%intRad4   = besttrackdata(ifile)%intRad4(idx1)
00505     besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idx1)
00506     besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idx1)
00507     besttrackdata(ifile)%intRmw    = besttrackdata(ifile)%intRmw(idx1)
00508     besttrackdata(ifile)%gusts     = besttrackdata(ifile)%gusts(idx1)
00509     besttrackdata(ifile)%eye       = besttrackdata(ifile)%eye(idx1)
00510     besttrackdata(ifile)%subregion = besttrackdata(ifile)%subregion(idx1)
00511     besttrackdata(ifile)%maxseas   = besttrackdata(ifile)%maxseas(idx1)
00512     besttrackdata(ifile)%initials  = besttrackdata(ifile)%initials(idx1)
00513     besttrackdata(ifile)%dir        = besttrackdata(ifile)%dir(idx1)
00514     besttrackdata(ifile)%intSpeed  = besttrackdata(ifile)%intSpeed(idx1)
00515     besttrackdata(ifile)%stormName = besttrackdata(ifile)%stormName(idx1)
00516     besttrackdata(ifile)%cycleNum  = besttrackdata(ifile)%cycleNum(idx1)
00517 END IF
00518
00519 DEALLOCATE(idx0)
00520 DEALLOCATE(idx1)
00521 !-----
00522 !----- This should be last after the fields are indexed in ascending order.
00523 !      It set the cycle number array in the data structure
00524 DO icnt = 1, besttrackdata(ifile)%numRec
00525 ! This is for the cycleNum, the last column we consider
00526 IF (icnt == 1) THEN
00527   kcmt = icnt
00528   besttrackdata(ifile)%cycleNum(icnt) = kcmt
00529 ELSE
00530   kcmt = kcmt + 1
00531   IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00532     besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00533     kcmt = kcmt - 1
00534 ELSE
00535   besttrackdata(ifile)%cycleNum(icnt) = kcmt
00536 END IF
00537 END IF
00538 END DO
00539 !----- This should be last after the fields are indexed in ascending order.
00540 !      We generate arbitrarily the forecast increments for internal use only.
00541 !      In the best track file, for the BEST track fields the forecast period
00542 !      is always 0.
00543 ! This is our reference time for the subsequent calculations
00544 CALL timeconv(besttrackdata(ifile)%year(1), besttrackdata(ifile)%month(1), &
00545           besttrackdata(ifile)%day(1), 0, 0, 0.0_sz, reffcsttime)
00546
00547 DO icnt = 1, besttrackdata(ifile)%numRec
00548   CALL timeconv(besttrackdata(ifile)%year(icnt), besttrackdata(ifile)%month(icnt), &
00549                 besttrackdata(ifile)%day(icnt), besttrackdata(ifile)%hour(icnt), &
00550                 0, 0.0_sz, tmpfcsttime)
00551   besttrackdata(ifile)%tau(icnt) = nint((tmpfcsttime - reffcsttime) / 3600.0_sz)
00552 END DO
00553
00554 CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile))
00555 END DO ! End of "ifFile" loop
00556
00557 CALL unsetmessagesource()
00558
00559 END SUBROUTINE readbesttrackfile
00560
00561 !=====
00562 !----- S U B R O U T I N E   R E A D   C S V   B E S T   T R A C K F I L E
00563 !-----
00564 !-----
00565 !-----
00566 !----- S U B R O U T I N E   R E A D C S V B E S T T R A C K F I L E
00567 !-----
00568 !-----
00569 !----- S U B R O U T I N E   R E A D C S V B E S T T R A C K F I L E
00570 !-----
00571 !-----
00572 SUBROUTINE readcsvbesttrackfile()
00573
00574 USE pahm_global, ONLY : nbtrfiles, besttrackfilename
00575 USE utilities, ONLY   : getlinerecord, openfileforread, touppercase, charunique, &
00576                         intvalstr

```

```

00587 USE timedateutils, ONLY : timeconv
00588 USE sortutils, ONLY      : arth, indexx, arrayequal
00589 USE csv_module
00590
00591 IMPLICIT NONE
00592
00593 TYPE(csv_file)          :: f
00594 CHARACTER(LEN=64), ALLOCATABLE :: sval2D(:, :)
00595 LOGICAL                  :: statusOK
00596
00597 CHARACTER(LEN=FNAMELEN)   :: inpFile
00598 CHARACTER(LEN=512)        :: line
00599 CHARACTER(LEN=64)         :: tmpStr
00600
00601 !CHARACTER(LEN=4)         :: castType
00602
00603 INTEGER                  :: iFile, nLines, lenLine
00604 INTEGER                  :: iCnt, jCnt, kCnt, kMax      ! loop counters
00605 INTEGER                  :: ios, status
00606
00607 CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00608 INTEGER, ALLOCATABLE        :: idxArrStr(:)
00609 INTEGER                  :: nUnique, maxCnt
00610
00611 INTEGER, ALLOCATABLE        :: idx0(:, ), idx1(:, )
00612 REAL(SZ)                  :: tmpFcstTime, refFcstTime
00613
00614 CALL setmessagesource("ReadCsvBestTrackFile")
00615
00616 ! Allocate the best track structure array. This structure holds all the
00617 ! input values for the storm track as read in from the track input file
00618 ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00619 ! (as appropriate).
00620 ALLOCATE(besttrackdata(nbtrfiles))
00621
00622 ! This is the main loop. We loop through all the best track files
00623 ! (user input)
00624 DO ifile = 1, nbtrfiles
00625   inpfile = besttrackfilename(ifile)
00626
00627   besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00628   besttrackdata(ifile)%thisStorm = ""
00629   besttrackdata(ifile)%loaded    = .false.
00630   besttrackdata(ifile)%numRec   = -1
00631
00632   CALL f%Read(trim(adjustl(inpfile)), status_ok=statusok)
00633   CALL f%Get(sval2d, status_ok=statusok)
00634
00635 ! Array allocation in the structure bestTrackData
00636 nlines = f%n_rows
00637 CALL allocbtrstruct(besttrackdata(ifile), nlines)
00638
00639 kcnt = 0
00640 DO icnt = 1, nlines
00641   DO jcmt = 1, f%n_cols
00642     line = line // trim(adjustl(sval2d(icnt, jcmt)))
00643   END DO
00644   jcmt = 0
00645
00646   lenline = len_trim(adjustl(line))
00647
00648 IF (lenline /= 0) THEN
00649   !--- col: 1
00650   tmpstr = trim(adjustl(sval2d(icnt, 1)))
00651   READ(tmpstr, '(a2)') &
00652     besttrackdata(ifile)%basin(icnt)
00653   !PV bestTrackData(iFile)%basin(iCnt)      = TRIM(ADJUSTL(sval2D(iCnt, 1)))
00654   !--- col: 2
00655   besttrackdata(ifile)%cyNum(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 2))))
00656   !--- col: 3
00657   tmpstr = trim(adjustl(sval2d(icnt, 3)))
00658   READ(tmpstr, '(a10)') &
00659     besttrackdata(ifile)%dtg(icnt)
00660   !PV bestTrackData(iFile)%dtg(iCnt)      = TRIM(ADJUSTL(sval2D(iCnt, 3)))
00661   !--- col: 4
00662   besttrackdata(ifile)%techNum(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 4))))
00663   !--- col: 5
00664   tmpstr = trim(adjustl(sval2d(icnt, 5)))
00665   READ(tmpstr, '(a4)') &
00666     besttrackdata(ifile)%tech(icnt)
00667   !PV bestTrackData(iFile)%tech(iCnt)      = TRIM(ADJUSTL(sval2D(iCnt, 5)))

```

```

00668      !--- col:  6
00669      besttrackdata(ifile)%tau(icnt)          = intvalstr(trim(adjust1(sval2d(icnt, 6))))
00670      !--- col:  7
00671      tmpstr = trim(adjust1(sval2d(icnt, 7)))
00672      READ(tmpstr, '(i3, a1)') &
00673          besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
00674      !--- col:  8
00675      tmpstr = trim(adjust1(sval2d(icnt, 8)))
00676      READ(tmpstr, '(i3, a1)') &
00677          besttrackdata(ifile)%intLon(icnt), besttrackdata(ifile)%ew(icnt)
00678      !--- col:  9
00679      besttrackdata(ifile)%intVMax(icnt)     = intvalstr(trim(adjust1(sval2d(icnt, 9))))
00680      !--- col: 10
00681      besttrackdata(ifile)%intMslp(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 10))))
00682      !--- col: 11
00683      WRITE(besttrackdata(ifile)%ty(icnt), '(a2)') trim(adjust1(sval2d(icnt, 11)))
00684      !--- col: 12
00685      besttrackdata(ifile)%rad(icnt)        = intvalstr(trim(adjust1(sval2d(icnt, 12))))
00686      !--- col: 13
00687      WRITE(besttrackdata(ifile)%windCode(icnt), '(a3)') trim(adjust1(sval2d(icnt, 13)))
00688      !--- col: 14
00689      besttrackdata(ifile)%intRad1(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 14))))
00690      !--- col: 15
00691      besttrackdata(ifile)%intRad2(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 15))))
00692      !--- col: 16
00693      besttrackdata(ifile)%intRad3(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 16))))
00694      !--- col: 17
00695      besttrackdata(ifile)%intRad4(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 17))))
00696      !--- col: 18
00697      besttrackdata(ifile)%intPOuter(icnt)  = intvalstr(trim(adjust1(sval2d(icnt, 18))))
00698      !--- col: 19
00699      besttrackdata(ifile)%intROuter(icnt)  = intvalstr(trim(adjust1(sval2d(icnt, 19))))
00700      !--- col: 20
00701      besttrackdata(ifile)%intRmw(icnt)     = intvalstr(trim(adjust1(sval2d(icnt, 20))))
00702      !--- col: 21
00703      besttrackdata(ifile)%gusts(icnt)      = intvalstr(trim(adjust1(sval2d(icnt, 21))))
00704      !--- col: 22
00705      besttrackdata(ifile)%eye(icnt)        = intvalstr(trim(adjust1(sval2d(icnt, 22))))
00706      !--- col: 23
00707      WRITE(besttrackdata(ifile)%subregion(icnt), '(a3)') trim(adjust1(sval2d(icnt, 23)))
00708      !--- col: 24
00709      besttrackdata(ifile)%maxseas(icnt)    = intvalstr(trim(adjust1(sval2d(icnt, 24))))
00710      !--- col: 25
00711      besttrackdata(ifile)%initials(icnt)   = trim(adjust1(sval2d(icnt, 25)))
00712      !--- col: 26
00713      besttrackdata(ifile)%dir(icnt)        = intvalstr(trim(adjust1(sval2d(icnt, 26))))
00714      !--- col: 27
00715      besttrackdata(ifile)%intSpeed(icnt)   = intvalstr(trim(adjust1(sval2d(icnt, 27))))
00716      !--- col: 28
00717      WRITE(besttrackdata(ifile)%stormName(icnt), '(a10)') trim(adjust1(sval2d(icnt, 28)))
00718
00719      !----- Convert lat/lon values to S/N and W/E notations
00720      IF (touppercase(besttrackdata(ifile)%ns(icnt)) == 'S') THEN
00721          besttrackdata(ifile)%lat(icnt) = -0.1_sz * besttrackdata(ifile)%intLat(icnt)
00722      ELSE
00723          besttrackdata(ifile)%lat(icnt) = 0.1_sz * besttrackdata(ifile)%intLat(icnt)
00724      END IF
00725
00726      IF (touppercase(besttrackdata(ifile)%ew(icnt)) == 'W') THEN
00727          besttrackdata(ifile)%lon(icnt) = -0.1_sz * besttrackdata(ifile)%intLon(icnt)
00728      ELSE
00729          besttrackdata(ifile)%lon(icnt) = 0.1_sz * besttrackdata(ifile)%intLon(icnt)
00730      END IF
00731
00732      !----- Get the year, month, day, hour from the DGT string
00733      READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
00734      besttrackdata(ifile)%year(icnt)
00735          IF (ios /= 0) besttrackdata(ifile)%year(icnt) = -1
00736          READ(besttrackdata(ifile)%dtg(icnt)(5:6), fmt='(i2.2)', iostat=ios)
00737          besttrackdata(ifile)%month(icnt)
00738              IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00739              READ(besttrackdata(ifile)%dtg(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00740              besttrackdata(ifile)%day(icnt)
00741                  IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00742                  READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00743                  besttrackdata(ifile)%hour(icnt)
00744                      IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00745
00746      END IF

```

```

00745      END DO
00746
00747      besttrackdata(ifile)%thisStorm = "
00748      besttrackdata(ifile)%loaded     = .true.
00749      besttrackdata(ifile)%numRec    = nlines
00750
00751 !-----
00752 ! Get the unique storm name and store it in the thisStorm string
00753 ALLOCATE(chkarrstr(nlines))
00754 ALLOCATE(idxarrstr(nlines))
00755
00756 nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00757
00758 maxcnt = -1
00759 DO kcnt = 1, nunique
00760     kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00761     IF (kmax > maxcnt) THEN
00762         maxcnt = kmax
00763         besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00764     END IF
00765 END DO
00766
00767 DEALLOCATE(chkarrstr)
00768 DEALLOCATE(idxarrstr)
00769 !-----
00770
00771 !-----
00772 ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00773 ! stored in ascending order
00774 ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00775 ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00776
00777 CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00778
00779 IF (status /= 0) THEN
00780     CALL unsetmessagesource()
00781
00782     CALL terminate()
00783 END IF
00784
00785 ! Create the index array to be used in the comparison below
00786 idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00787
00788 IF (.NOT. arrayequal(idx0, idx1)) THEN
00789     besttrackdata(ifile)%basin      = besttrackdata(ifile)%basin(idx1)
00790     besttrackdata(ifile)%cyNum     = besttrackdata(ifile)%cyNum(idx1)
00791     besttrackdata(ifile)%dtg       = besttrackdata(ifile)%dtg(idx1)
00792     besttrackdata(ifile)%techNum   = besttrackdata(ifile)%techNum(idx1)
00793     besttrackdata(ifile)%tech     = besttrackdata(ifile)%tech(idx1)
00794     besttrackdata(ifile)%tau      = besttrackdata(ifile)%tau(idx1)
00795     besttrackdata(ifile)%intLat   = besttrackdata(ifile)%intLat(idx1)
00796     besttrackdata(ifile)%ns       = besttrackdata(ifile)%ns(idx1)
00797     besttrackdata(ifile)%intLon   = besttrackdata(ifile)%intLon(idx1)
00798     besttrackdata(ifile)%ew      = besttrackdata(ifile)%ew(idx1)
00799     besttrackdata(ifile)%intVMax  = besttrackdata(ifile)%intVMax(idx1)
00800     besttrackdata(ifile)%intMslp  = besttrackdata(ifile)%intMslp(idx1)
00801     besttrackdata(ifile)%ty      = besttrackdata(ifile)%ty(idx1)
00802     besttrackdata(ifile)%rad     = besttrackdata(ifile)%rad(idx1)
00803     besttrackdata(ifile)%windCode = besttrackdata(ifile)%windCode(idx1)
00804     besttrackdata(ifile)%intRad1  = besttrackdata(ifile)%intRad1(idx1)
00805     besttrackdata(ifile)%intRad2  = besttrackdata(ifile)%intRad2(idx1)
00806     besttrackdata(ifile)%intRad3  = besttrackdata(ifile)%intRad3(idx1)
00807     besttrackdata(ifile)%intRad4  = besttrackdata(ifile)%intRad4(idx1)
00808     besttrackdata(ifile)%intPOuter = besttrackdata(ifile)%intPOuter(idx1)
00809     besttrackdata(ifile)%intROuter = besttrackdata(ifile)%intROuter(idx1)
00810     besttrackdata(ifile)%intRmw   = besttrackdata(ifile)%intRmw(idx1)
00811     besttrackdata(ifile)%gusts   = besttrackdata(ifile)%gusts(idx1)
00812     besttrackdata(ifile)%eye     = besttrackdata(ifile)%eye(idx1)
00813     besttrackdata(ifile)%subregion = besttrackdata(ifile)%subregion(idx1)
00814     besttrackdata(ifile)%maxseas  = besttrackdata(ifile)%maxseas(idx1)
00815     besttrackdata(ifile)%initials = besttrackdata(ifile)%initials(idx1)
00816     besttrackdata(ifile)%dir     = besttrackdata(ifile)%dir(idx1)
00817     besttrackdata(ifile)%intSpeed = besttrackdata(ifile)%intSpeed(idx1)
00818     besttrackdata(ifile)%stormName = besttrackdata(ifile)%stormName(idx1)
00819     besttrackdata(ifile)%cycleNum = besttrackdata(ifile)%cycleNum(idx1)
00820
00821 END IF
00822
00823 DEALLOCATE(idx0)
00824 DEALLOCATE(idx1)
00825 !-----

```

```

00826     CALL f%Destroy()
00827
00828 !----- This should be last after the fields are indexed in ascending order.
00829 !      It set the cycle number array in the data structure
00830 DO icnt = 1, besttrackdata(ifile)%numRec
00831 ! This is for the cycleNum, the last column we consider
00832 IF (icnt == 1) THEN
00833   kcnt = icnt
00834   besttrackdata(ifile)%cycleNum(icnt) = kcnt
00835 ELSE
00836   kcnt = kcnt + 1
00837   IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00838     besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00839     kcnt = kcnt - 1
00840   ELSE
00841     besttrackdata(ifile)%cycleNum(icnt) = kcnt
00842   END IF
00843 END IF
00844 END DO
00845
00846 !----- This should be last after the fields are indexed in ascending order. !PV NEED TO CHECK
00847 ON THIS
00848 ! We generate arbitrarily the forecast increments for internal use only.
00849 ! In the best track file, for the BEST track fields the forecast period
00850 ! is always 0.
00851 ! This is our reference time for the subsequent calculations
00852 CALL timeconv(besttrackdata(ifile)%year(1), besttrackdata(ifile)%month(1), &
00853           besttrackdata(ifile)%day(1), 0, 0, 0.0_sz, reffcstime)
00854 DO icnt = 1, besttrackdata(ifile)%numRec
00855   CALL timeconv(besttrackdata(ifile)%year(icnt), besttrackdata(ifile)%month(icnt), &
00856                 besttrackdata(ifile)%day(icnt), besttrackdata(ifile)%hour(icnt), &
00857                 0, 0.0_sz, tmpfcstime)
00858   besttrackdata(ifile)%tau(icnt) = nint((tmpfcstime - reffcstime) / 3600.0_sz)
00859 END DO
00860
00861 CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile))
00862
00863 END DO ! End of "iFile" loop
00864
00865 CALL unsetmessagesource()
00866
00867 END SUBROUTINE readcsvbesttrackfile
00868
00869 !=====
00870
00871 !----- S U B R O U T I N E   P R O C E S S   H O L L A N D   D A T A
00872 !-----
00873 !
00874 SUBROUTINE processhollanddata(idTrFile, strOut, status)
00875
00876 USE pahm_global, ONLY : nm2m, kt2ms, nbtrfiles
00877 USE utilities, ONLY : touppercase, charunique
00878 USE timedateutils, ONLY : timeconv
00879 USE pahm_vortex, ONLY : calcintensitychange, uvtrans
00880
00881 IMPLICIT NONE
00882
00883 INTEGER, INTENT(IN) :: idTrFile
00884 TYPE(hollanddata_t), INTENT(OUT) :: strOut
00885 INTEGER, INTENT(OUT) :: status ! error status
00886
00887 ! numUniqRec, outDTG, idxDTG are used to identify the unique DTG elements in the input structure
00888 INTEGER :: numUniqRec
00889 CHARACTER(LEN=10), ALLOCATABLE :: outDTG(:)
00890 INTEGER, ALLOCATABLE :: idxDTG(:)
00891
00892 INTEGER :: plIdx ! populated index for Holland Data array
00893 INTEGER :: iCnt ! loop counters
00894
00895 CHARACTER(LEN=4) :: castType !hindcast,forecast
00896 REAL(SZ), ALLOCATABLE :: castTime(:) ! seconds since start of year
00897
00898 REAL(SZ) :: spdVal, pressVal, rrpVal, rmwVal
00899
00900 status = 0 ! no error
00901
00902 CALL setmessagesource("ProcessHollandData")
00903
```

```

00925      IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
00926          IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
00927              status = 2
00928
00929          WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ',
00930          idtrfile
00931          CALL allmessage(error, scratchmessage)
00932
00933          CALL unsetmessagesource()
00934
00935          RETURN
00936      END IF
00937
00938      ELSE
00939          status = 1
00940
00941          WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
00942          ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ',
00943          nbtrfiles
00944          CALL allmessage(error, scratchmessage)
00945
00946          CALL unsetmessagesource()
00947
00948          RETURN
00949      END IF
00950
00951      WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
00952      CALL logmessage(info, scratchmessage)
00953
00954      ! Most likely the array size will be larger if repeated times are found
00955      ! in the best track structure.
00956      ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
00957      ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))
00958
00959      ! Get unique lines that represent new points in time.
00960      ! Repeated time points occur in hindcasts for the purpose of
00961      ! describing winds in the quadrants of the storm. We don't use the
00962      ! quadrant-by-quadrant wind data. Repeated time data occur in the
00963      ! forecast because the time data is just the time that the forecast
00964      ! was made. The important parameter in the forecast file is the
00965      ! forecast increment.
00966      numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)
00967
00968      !-----
00969      ! Populate the Holland structure
00970      !-----
00971      CALL allochollstruct(strout, numuniqrec)
00972
00973      ALLOCATE(casttime(numuniqrec))
00974
00975      strout%fileName = besttrackdata(idtrfile)%fileName
00976      strout>thisStorm = besttrackdata(idtrfile)%thisStorm
00977      strout%loaded = .true.
00978      strout%numRec = numuniqrec
00979
00980      WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
00981      CALL logmessage(info, scratchmessage)
00982
00983      DO icnt = 1, numuniqrec
00984          plidx = idxdtg(icnt)
00985
00986          casttype = toupper(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))
00987
00988          ! Convert speeds from knots to m/s
00989          spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)
00990
00991          ! Convert pressure(s) from mbar to Pa
00992          pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)
00993
00994          ! Convert all distances from nm to km/m
00995          rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
00996          rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx) ! in m
00997
00998          strout%basin(icnt) = besttrackdata(idtrfile)%basin(plidx)
00999          strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)
01000          strout%dtg(icnt) = besttrackdata(idtrfile)%dtg(plidx)
01001          strout%year(icnt) = besttrackdata(idtrfile)%year(plidx)
01002          strout%month(icnt) = besttrackdata(idtrfile)%month(plidx)
01003          strout%day(icnt) = besttrackdata(idtrfile)%day(plidx)
01004          strout%hour(icnt) = besttrackdata(idtrfile)%hour(plidx)
01005          strout%castType(icnt) = besttrackdata(idtrfile)%tech(plidx)
01006          strout%fcstInc(icnt) = besttrackdata(idtrfile)%tau(plidx)

```

```

01004     strout%iLat(icnt)          = besttrackdata(idtrfile)%intLat(plidx)
01005     strout%lat(icnt)          = besttrackdata(idtrfile)%lat(plidx)
01006     strout%iLon(icnt)         = besttrackdata(idtrfile)%intLon(plidx)
01007     strout%lon(icnt)          = besttrackdata(idtrfile)%lon(plidx)
01008
01009     strout%iSpeed(icnt)       = besttrackdata(idtrfile)%intVMax(plidx)
01010     strout%speed(icnt)        = spdval
01011     strout%iCPress(icnt)      = besttrackdata(idtrfile)%intMslp(plidx)
01012     strout%cPress(icnt)       = pressval
01013     strout%iRrp(icnt)         = besttrackdata(idtrfile)%intROuter(plidx)
01014     strout%rrp(icnt)          = rrpval
01015     strout%iRmw(icnt)         = besttrackdata(idtrfile)%intRmw(plidx)
01016     strout%rmw(icnt)          = rmwval
01017
01018 ! PV check if this SELECT code is actually needed. Need to check the different format
01019 ! of input files.
01020 SELECT CASE(casttype)
01021   CASE("BEST")           ! nowcast/hindcast
01022     ! PV check if this is needed
01023     CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0,
01024     0.0_sz, casttime(icnt))
01025   CASE("OFCL")            ! forecast
01026     ! PV check if this is needed
01027     IF (icnt > 1) THEN
01028       IF ((strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
01029     cycle
01030       END IF
01031
01032       IF (strout%fcstInc(icnt) == 0) THEN
01033         CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
01034           strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
01035       ELSE
01036         casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1)) *
01037           3600.0_sz
01038       END IF
01039
01040       IF ((strout%iCPress(icnt) == 0) .OR. (strout%iRmw(icnt) == 0)) THEN
01041         CALL allmessage(error,
01042           'The storm hindcast/forecast input file ' // trim(strout%fileName) // &
01043             ' contains invalid data for central pressure or rMax.')
01044         CALL terminate()
01045       END IF
01046
01047       ! Adding a new type to allow the analyst to add lines
01048       ! that do nothing but produce zero winds and background barometric
01049       ! pressure. These lines can have a date/time like a BEST line or
01050       ! a date/time and forecast period like an OFCL line.
01051       CASE("CALM")
01052         ! PV check if this is needed
01053         WRITE(scratchmessage, '(a)') 'The file: ' // trim(strout%fileName) // ' contains at least one
01054         "CALM" line.'
01055         CALL logmessage(echo, scratchmessage)
01056
01057         IF (icnt > 1) THEN
01058           IF ((strout%fcstInc(icnt) /= 0) .AND. (strout%fcstInc(icnt) == strout%fcstInc(icnt - 1)))
01059         cycle
01060           END IF
01061
01062           IF (strout%fcstInc(icnt) == 0) THEN
01063             CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), &
01064               strout%hour(icnt), 0, 0.0_sz, casttime(icnt))
01065           ELSE
01066             casttime(icnt) = casttime(icnt - 1) + (strout%fcstInc(icnt) - strout%fcstInc(icnt - 1)) *
01067               3600.0_sz
01068           END IF
01069
01070           CASE DEFAULT          ! unrecognized
01071             WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of '
01072             // &
01073             trim(adjustl(strout%fileName))
01074             CALL allmessage(error, scratchmessage)
01075
01076             CALL terminate()
01077           END SELECT
01078
01079           strout%castTime(icnt) = casttime(icnt)
01080         END DO    ! numUniqRec
01081
01082           ! Calculate the cPress intensity change (dP/dt)
01083           CALL calcintensitychange(strout%cPress, casttime, strout%cPrDt, status, 2)

```

```

01078
01079      ! Calculate storm translation velocities based on change in position,
01080      ! approximate u and v translation velocities
01081      CALL uvtrans(strout%lat, strout%lon, casttime, strout%trVx, strout%trVy, status, 2)
01082
01083      DEALLOCATE(casttime)
01084      !-----
01085
01086      DEALLOCATE(outdtg)
01087      DEALLOCATE(idxdtg)
01088
01089      CALL unsetmessagesource()
01090
01091      END SUBROUTINE processhollanddata
01092
01093 !=====
01094
01095 !-----
01096 !  S U B R O U T I N E   P R O C E S S   A S Y M M E T R I C   V O R T E X   D A T A
01097 !-----
01117 !-----
01118 SUBROUTINE processasymmetricvortexdata(idTrFile, strOut, status)
01119
01120     USE pahm_global, ONLY : rad2deg, deg2rad, nm2m, kt2ms, ms2kt, &
01121                      & backgroundatmpress, windreduction, besttrackfilename, nbtrfiles
01122     USE utilities, ONLY : touppercase, charunique, sphericaldistance
01123     USE timedateutils, ONLY : timeconv
01124     USE pahm_vortex
01125
01126
01127     IMPLICIT NONE
01128
01129     INTEGER, INTENT(IN)          :: idTrFile
01130     TYPE(asymmetricvortexdata_t), INTENT(OUT) :: strOut
01131     INTEGER, INTENT(OUT)          :: status ! error status
01132
01133 ! ----- Local variables
01134     INTEGER                      :: numRec
01135
01136     INTEGER                      :: iCnt, iCyc, i, k ! loop counters
01137
01138     CHARACTER(LEN=4)              :: castType      !hindcast,forecast
01139     REAL(SZ), ALLOCATABLE         :: castTime(:)    ! seconds since start of year
01140
01141     INTEGER                      :: nCycles
01142     REAL(SZ), DIMENSION(:), ALLOCATABLE :: cycleTime
01143     INTEGER, DIMENSION(:), ALLOCATABLE :: totRecPerCycle
01144
01145     INTEGER                      :: radiiSum ! record radius values for filling in missing vals
01146     INTEGER                      :: numNonZero ! number of nonzero isotach radii
01147     INTEGER                      :: firstEntry   ! first entry in the cycle
01148     INTEGER                      :: lastEntry    ! last entry in the cycle
01149     REAL(sz)                     :: stormMotion  ! portion of Vmax attributable to storm motion
01150     REAL(sz)                     :: stormMotionU ! U portion of Vmax attributable to storm motion
01151     REAL(sz)                     :: stormMotionV ! V portion of Vmax attributable to storm motion
01152     REAL(sz)                     :: U_Vr, V_Vr
01153     REAL(SZ), DIMENSION(4)       :: vmwBL
01154     INTEGER, DIMENSION(4)        :: vmwBLflag
01155     REAL(SZ)                     :: vMaxPseudo
01156     REAL(SZ), DIMENSION(:,:,), ALLOCATABLE :: phiFactors
01157     REAL(SZ), DIMENSION(4)       :: vMaxesBLTemp
01158     INTEGER, DIMENSION(:, :, ), ALLOCATABLE :: irad ! working isotach radii
01159     REAL(SZ), DIMENSION(4)       :: rMaxWTemp
01160     INTEGER                      :: iQuadRot, nQuadRot
01161
01162     INTEGER, DIMENSION(0:5)     :: lookupRadii ! periodic interpolation
01163     REAL(SZ), DIMENSION(4)       :: r
01164     REAL(SZ)                     :: pn ! Ambient surface pressure (mb)
01165     REAL(SZ)                     :: pc ! Surface pressure at center of storm (mb)
01166     REAL(SZ)                     :: cLat, cLon ! Current eye location (degrees north, degrees east)
01167     REAL(SZ)                     :: vMax ! Current Max sustained wind velocity in storm (knots)
01168     REAL(SZ), DIMENSION(4)       :: gamma ! factor applied to the StormMotion
01169     REAL(SZ), DIMENSION(4)       :: quadRotateAngle, quadRotateAngle_new
01170     REAL(SZ), DIMENSION(4)       :: rMaxWHighIso
01171     REAL(SZ), DIMENSION(4)       :: epsilonAngle
01172     INTEGER, DIMENSION(4)        :: irr
01173     LOGICAL, DIMENSION(4)        :: vioFlag
01174     REAL(SZ)                     :: vMaxBL ! max sustained wind at top of atm. b.l.
01175     REAL(SZ)                     :: azimuth ! angle of node w.r.t. vortex (radians)
01176     REAL(SZ), DIMENSION(4)       :: quadrantVr, quadrantAngles, quadrantVecAngles
01177     REAL(SZ)                     :: vr ! Current velocity @ wind radii (knots)

```

```

01178
01179     status = 0 ! no error
01180
01181     CALL setmessagesource("ProcessAsymmetricVortexData")
01182
01183     IF ((idtrfile >= 1) .AND. (idtrfile <= nbtrfiles)) THEN
01184         IF (.NOT. besttrackdata(idtrfile)%loaded) THEN
01185             status = 2
01186
01187         WRITE(scratchmessage, '(a, i0)') 'Error while loading best track data structure with id: ',
01188         idtrfile
01189         CALL allmessage(error, scratchmessage)
01190
01191         CALL unsetmessagesource()
01192
01193         RETURN
01194     END IF
01195     ELSE
01196         status = 1
01197
01198         WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
01199                         ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ',
01200         nbtrfiles
01201         CALL allmessage(error, scratchmessage)
01202
01203         CALL unsetmessagesource()
01204
01205         RETURN
01206     END IF
01207
01208     WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
01209     CALL logmessage(info, scratchmessage)
01210
01211     ! This is the number of all records in the processed best track data structure
01212     numrec = besttrackdata(idtrfile)%numRec
01213
01214     !-----!
01215     ! Populate the asymmetric vortex structure
01216     !-----!
01217     CALL allocasymvortstruct(strout, numrec)
01218
01219     ALLOCATE(casttime(numrec))
01220     ALLOCATE(cycletteime(numrec))
01221     ALLOCATE(totrecpercycle(numrec))
01222     ALLOCATE(phifactors(numrec, 4))
01223     ALLOCATE(irad(numrec, 4))
01224
01225     strout%fileName = besttrackdata(idtrfile)%fileName
01226     strout>thisStorm = besttrackdata(idtrfile)%thisStorm
01227     strout%loaded = .true.
01228     strout%numRec = numrec
01229
01230     totrecpercycle = 0
01231
01232     WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
01233     CALL logmessage(info, scratchmessage)
01234
01235     ncycles = 1
01236     totrecpercycle(ncycles) = 1
01237
01238     DO icnt = 1, numrec
01239
01240         casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))
01241
01242         strout%basin(icnt) = besttrackdata(idtrfile)%basin(icnt)
01243         strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(icnt)
01244         strout%dtg(icnt) = besttrackdata(idtrfile)%dtg(icnt)
01245         strout%year(icnt) = besttrackdata(idtrfile)%year(icnt)
01246         strout%month(icnt) = besttrackdata(idtrfile)%month(icnt)
01247         strout%day(icnt) = besttrackdata(idtrfile)%day(icnt)
01248         strout%hour(icnt) = besttrackdata(idtrfile)%hour(icnt)
01249         strout%castTypeNum(icnt) = besttrackdata(idtrfile)%techNum(icnt)
01250         strout%castType(icnt) = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))
01251         strout%fcstInc(icnt) = besttrackdata(idtrfile)%tau(icnt)
01252         strout%iLat(icnt) = besttrackdata(idtrfile)%intLat(icnt)
01253         strout%lat(icnt) = besttrackdata(idtrfile)%lat(icnt)
01254         strout%iLon(icnt) = besttrackdata(idtrfile)%intLon(icnt)
01255         strout%lon(icnt) = besttrackdata(idtrfile)%lon(icnt)
01256         strout%ew(icnt) = besttrackdata(idtrfile)%ew(icnt)
01257         strout%ns(icnt) = besttrackdata(idtrfile)%ns(icnt)

```

```

01257     strout%ispeed(icnt)      = besttrackdata(idtrfile)%intVMax(icnt)
01258     strout%speed(icnt)       = kt2ms * strout%ispeed(icnt)           ! Convert speeds from knots to m/s
01259     strout%icPress(icnt)     = besttrackdata(idtrfile)%intMslp(icnt)
01260     strout%icPress(icnt)     = 100.0_sz * strout%icPress(icnt)
01261     strout%ty(icnt)         = besttrackdata(idtrfile)%ty(icnt)
01262     strout%ivr(icnt)        = besttrackdata(idtrfile)%rad(icnt)
01263     strout%windCode(icnt)    = besttrackdata(idtrfile)%windCode(icnt)
01264     strout%ir(icnt, 1)        = besttrackdata(idtrfile)%intRad1(icnt)
01265     strout%ir(icnt, 2)        = besttrackdata(idtrfile)%intRad2(icnt)
01266     strout%ir(icnt, 3)        = besttrackdata(idtrfile)%intRad3(icnt)
01267     strout%ir(icnt, 4)        = besttrackdata(idtrfile)%intRad4(icnt)
01268     strout%ipr(icnt)         = besttrackdata(idtrfile)%intPOuter(icnt)
01269     strout%prp(icnt)         = 100.0_sz * strout%ipr(icnt)           ! Convert pressure(s) from mbar to
01270 Pa
01271     strout%irRp(icnt)        = besttrackdata(idtrfile)%intROuter(icnt)
01272     strout%rrp(icnt)          = nm2m * strout%irRp(icnt)             ! Convert all distances from nm to
01273 m
01274     strout%irMw(icnt)        = besttrackdata(idtrfile)%intRmw(icnt)
01275     strout%rmw(icnt)          = nm2m * strout%irMw(icnt)
01276     strout%gusts(icnt)        = besttrackdata(idtrfile)%gusts(icnt)
01277     strout%eye(icnt)          = besttrackdata(idtrfile)%eye(icnt)
01278     strout%subregion(icnt)    = besttrackdata(idtrfile)%subregion(icnt)
01279     strout%maxseas(icnt)      = besttrackdata(idtrfile)%maxseas(icnt)
01280     strout%initials(icnt)     = besttrackdata(idtrfile)%initials(icnt)
01281     strout%idir(icnt)         = besttrackdata(idtrfile)%dir(icnt)
01282     strout%dir(icnt)          = real(strout%dir(icnt))
01283     strout%istormSpeed(icnt)  = besttrackdata(idtrfile)%intSpeed(icnt)
01284     strout%stormSpeed(icnt)   = kt2ms * strout%istormSpeed(icnt)        ! Convert speeds from knots to m/s
01285     strout%stormName(icnt)    = besttrackdata(idtrfile)%stormName(icnt)
01286
01286 !PV DO WE NEED TO INCLUDE THE SAME CODE FOR CASTTIME AS IN HOLLAND?
01287     CALL timeconv(strout%year(icnt), strout%month(icnt), strout%day(icnt), strout%hour(icnt), 0, 0.0_sz,
01288     casttime(icnt))
01289     strout%castTime(icnt) = casttime(icnt)
01290
01291 !----- Check for a new cycle
01292 IF (icnt /= 1) THEN
01293   IF (strout%fcstInc(icnt) /= strout%fcstInc(icnt-1)) THEN
01294     ncycles = ncycles + 1
01295     totrecpercycle(ncycles) = 1
01296   ELSE
01297     ! Increment the number of isotachs for this cycle if this
01298     ! entry belongs to the same cycle as the last
01299     totrecpercycle(ncycles) = totrecpercycle(ncycles) + 1
01300 END IF
01301 END IF
01302 strout%nCycles = ncycles
01303 !strOut%numCycle(iCnt) = nCycles
01304 strout%numCycle(icnt) = besttrackdata(idtrfile)%cycleNum(icnt)
01305 cycletime(icnt) = strout%fcstInc(icnt) * 3600.0_sz
01306
01307 ! Calculate the translation velocity in m/s and knots,
01308 ! Set background pressure
01309 DO icnt = 1, numrec
01310   ! Set iprp to background pressure
01311   strout%ipr(icnt) = nint(backgroundatmpress)      ! in mbar
01312   strout%prp(icnt) = 100.0_sz * strout%ipr(icnt) ! in Pa
01313
01314 ! Check/set central pressure
01315 IF (strout%icPress(icnt) == 0) THEN
01316   IF (icnt == 1) THEN
01317     WRITE(scratchmessage, '(a)') &
01318       'Central pressure set to zero on first line/record when processing the best track file: ' // &
01319       trim(adjustl(besttrackfilename(idtrfile)))
01320     CALL alimessage(error, scratchmessage)
01321     CALL terminate()
01322   ELSE
01323     CALL alimessage(warning, "Central pressure persisted from previous value.")
01324     strout%icPress(icnt) = strout%icPress(icnt - 1)
01325     strout%cPress(icnt) = strout%cPress(icnt - 1)
01326   END IF
01327 END IF
01328
01329 ! @jasonfleming: in rare cases where the central pressure is
01330 ! higher than 1012mb (e.g., charley 2004), set the background
01331 ! pressure so that it is 1mb higher than the central pressure
01332 ! to avoid producing negative Holland B values.
01333 IF (strout%icPress(icnt) > 1012) THEN
01334   WRITE(scratchmessage,'(a,i0,a)') 'The central pressure ' // &

```

```

01335           ' is higher than the PaHM default background barometric' // &
01336           'pressure on line', icnt, &
01337           '. For this line, the background barometric' // &
01338           'pressure will be set 1mb higher than central pressure.'
01339           CALL allmessage(info, scratchmessage)
01340           strout%iPrp(icnt) = strout%icPress(icnt) + 1
01341           strout%prp(icnt) = 100.0_sz * strout%iPrp(icnt)
01342       END IF
01343
01344       IF (comparereals(cycletime(icnt), cycletime(1), 0.01_sz) == 0) THEN
01345           cycle
01346       END IF
01347
01348       IF (comparereals(cycletime(icnt), cycletime(icnt-1), 0.01_sz) == 0) THEN
01349           strout%trVx(icnt) = strout%trVx(icnt - 1)
01350           strout%trVy(icnt) = strout%trVy(icnt - 1)
01351           strout%stormSpeed(icnt)= strout%stormSpeed(icnt - 1)
01352       ELSE
01353           ! approximate u and v translation velocities
01354           CALL uvtranspoint(strout%lat(icnt - 1), strout%lon(icnt - 1), strout%lat(icnt), strout%lon(icnt),
&
01355               & cycletime(icnt - 1), cycletime(icnt), strout%trVx(icnt), strout%trVy(icnt))
01356
01357           ! Get translation speed.
01358           ! We convert this speed to Knots for the subsequent calculations, but at the
01359           ! end we will set strOut%iStormSpeed = NINT(strOut%stormSpeed) and convert
01360           ! back strOut%stormSpeed to m/s to store its value in the data structure
01361           strout%stormSpeed(icnt) = ms2kt * sqrt(strout%trVx(icnt)**2 + strout%trVy(icnt)**2) ! in Knots
01362       END IF
01363   END DO      ! numRec
01364
01365   ! now set the translation velocity in the first cycle equal
01366   ! to the translation velocity in the 2nd cycle, for lack of any
01367   ! better information
01368   DO icnt = 2, numrec
01369       IF (comparereals(cycletime(icnt), cycletime(1), 0.01_sz) /= 0) THEN
01370           strout%trVx(1:(icnt - 1)) = strout%trVx(icnt)
01371           strout%trVy(1:(icnt - 1)) = strout%trVy(icnt)
01372           strout%stormSpeed(1:(icnt - 1))= strout%stormSpeed(icnt)
01373           EXIT
01374       END IF
01375   END DO
01376
01377   ! convert trVx and trVy to speed and direction
01378   ! direction is in compass coordinates 0 == North
01379   ! increasing clockwise
01380   DO icnt = 1, numrec
01381       IF (strout%stormSpeed(icnt) < 1.0_sz ) THEN
01382           ! The vortex module can't handle speed and direction
01383           ! being zero; it will return NaNs as a result. Persist the
01384           ! direction from the previous cycle, and make the storm translation
01385           ! speed small but nonzero.
01386           strout%stormSpeed(icnt) = 1.0_sz
01387           IF (icnt > 1) THEN
01388               strout%dir(icnt) = strout%dir(icnt - 1)
01389           ELSE
01390               strout%dir(icnt) = 0.0
01391           END IF
01392       ELSE
01393           ! calculate angle in compass coordinates
01394           strout%dir(icnt) = rad2deg * atan2(strout%trVx(icnt), strout%trVy(icnt))
01395           IF (strout%dir(icnt) < 0.0_sz) THEN
01396               strout%dir(icnt) = strout%dir(icnt) + 360.0_sz
01397           END IF
01398       END IF
01399   END DO
01400
01401 !-----
01402 ! Now using the calculated translational velocities
01403 ! call the vortex module and compute the Rmax's
01404 ! to be used in the new input file
01405 !-----
01406 ! Initialize azimuth values in quadrants
01407 azimuth = 45.0_sz
01408 DO i = 1, 4
01409     quadrantangles(i) = deg2rad * azimuth
01410     azimuth = azimuth - 90.0_sz
01411 END DO
01412
01413 irad(:, :) = strout%ir(:, :)
01414

```

```

01415    DO icyc = 1, ncycles
01416      lastentry = sum(totrecpercycle(1:icyc))
01417
01418      DO k = 1, totrecpercycle(icyc)
01419        icnt = lastentry + 1 - k
01420
01421        WRITE(scratchmessage,'(a,i0)') 'Start      processing iCnt = ', icnt
01422        CALL logmessage(info, scratchmessage)
01423
01424        ! Transform variables from integers
01425        ! to real numbers for hurricane vortex calcualtions.
01426        vmax = real(strout%Speed(icnt), sz)
01427        pn   = real(strout%Prp(icnt), sz)
01428        pc   = real(strout%CPress(icnt), sz)
01429        clat = shout%lat(icnt)
01430        clon = shout%lon(icnt)
01431
01432        ! need to get some logic incase Vr is zero
01433        ! if so we will also be setting ir(:) to Rmax
01434        ! ... this happens when storms are at the "invest" stage
01435        IF (strout%ivr(icnt) == 0 ) THEN
01436          vr = vmax
01437        ELSE
01438          vr = real(strout%ivr(icnt))
01439        END IF
01440
01441        lookupradii(0) = shout%ir(icnt, 4)
01442        lookupradii(5) = shout%ir(icnt, 1)
01443        radiisum = 0
01444        num nonzero = 0
01445
01446        DO i=1, 4
01447          lookupradii(i) = shout%ir(icnt,i)
01448          radiisum = radiisum + shout%ir(icnt,i)
01449          IF (shout%ir(icnt, i) > 0) THEN
01450            num nonzero = num nonzero + 1
01451            shout%quadFlag(icnt, i) = 1 ! use the Rmax resulting from this
01452          ELSE
01453            shout%quadFlag(icnt, i) = 0 ! don't use Rmax resulting from this
01454          END IF
01455        END DO
01456
01457        ! Fill missing values based on how many are missing
01458        SELECT CASE(num nonzero)
01459        CASE(0) ! no isotachs reported, use overall Rmax; set isotach to vMax
01460          shout%quadFlag(icnt, :) = 1
01461          IF (shout%ir(icnt) /= 0) THEN
01462            irad(icnt, :) = shout%ir(icnt)
01463          ELSE
01464            irad(icnt, :) = 40 ! need a nonzero value for Rmax calcs,
01465                           ! this val will be thrown away later
01466          END IF
01467          vr = vmax
01468        CASE(1) ! set missing radii equal to half the nonzero radius
01469          WHERE(shout%ir(icnt, :) == 0) irad(icnt, :) = nint(0.5 * radiisum)
01470        CASE(2) ! set missing to half the avg of the 2 radii that are given
01471          WHERE(shout%ir(icnt, :) == 0) irad(icnt, :) = nint(0.5 * radiisum * 0.5)
01472        CASE(3) ! set missing radius to half the average of the radii on either side
01473          DO i = 1, 4
01474            IF (shout%ir(icnt,i) == 0) THEN
01475              irad(icnt,i) = nint(0.5 * (lookupradii(i + 1) + lookupradii(i - 1)))
01476            END IF
01477          END DO
01478        CASE(4)
01479          ! use all these radii as-is
01480        CASE default
01481          ! the following error message should be unreachable
01482          WRITE(scratchmessage,'(a,i0,a)') 'Number of nonzero radii on line ', icnt, &
01483                                         ' not in range 0 to 4.'
01484          CALL allmessage(info, scratchmessage)
01485        END SELECT
01486
01487        DO i = 1, 4
01488          r(i) = real(irad(icnt, i), sz)
01489        END DO
01490
01491        shout%hollB(icnt)      = 1.0_sz
01492        shout%hollBs(icnt, 1:4) = 1.0_sz
01493        phifactors(icnt, 1:4)   = 1.0_sz
01494        irr                     = shout%ir(icnt, :)
01495

```

```

01496 !-----
01497 ! Create a new asymmetric hurricane vortex.
01498 !
01499 ! Note: Subtract translational speed from vMax, then
01500 ! scale (vMax - Vt) and vr up to the top of the surface,
01501 ! where the cyclostrophic wind balance is valid.
01502 !-----
01503 CALL setusevmaxesbl(.true.)
01504
01505 stormmotion = 1.5_sz * strout%stormSpeed(icnt)**0.63_sz
01506 stormmotionu = sin(strout%dir(icnt) * deg2rad) * stormmotion
01507 stormmotionv = cos(strout%dir(icnt) * deg2rad) * stormmotion
01508 vmaxbl = (vmax - stormmotion) / windreduction
01509
01510 SELECT CASE(approach)
01511 CASE(1) !Normal approach: assume vr and quadrantVr vectors
01512 !are both tangential to azimuth
01513 DO i = 1, 4
01514 ! quadrant angles are in the radial direction, we need
01515 ! the tangential direction, b/c that is the direction of vr
01516 u_vr = vr * cos(quadrantangles(i) + (deg2rad * 90.0_sz))
01517 v_vr = vr * sin(quadrantangles(i) + (deg2rad * 90.0_sz))
01518
01519 ! eliminate the translational speed based on vortex wind speed
01520 ! gamma = |quadrantVr| / |vMaxBL|
01521 gamma(i) = ((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01522 & sqrt((2.0_sz * u_vr * stormmotionu+2.0_sz * v_vr * stormmotionv)**2.0_sz - &
01523 & 4.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * &
01524 & windreduction**2.0_sz) * vr**2.0_sz)) / (2.0_sz *
01525 & (stormmotion**2.0_sz - vmaxbl**2.0_sz * windreduction**2.0_sz))
01526 gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)
01527
01528 quadrantvr(i) = sqrt((u_vr - gamma(i) * stormmotionu)**2.0_sz +
01529 & (v_vr - gamma(i) * stormmotionv)**2.0_sz) / windreduction
01530 END DO
01531
01532 ! If violation occurs at any quadrant (quadrantVr(i)>vMaxBL),
01533 ! re-calculate quadrantVr at those violated quadrants
01534 DO i = 1, 4
01535 IF (quadrantvr(i) > vmaxbl) THEN
01536 ! Replace vMax with Vr when violation occurs (including
01537 ! situations when isotach is not reported at that quadrant:
01538 ! especially at the investment stage or for the highest isotachs
01539 ! that is not always available.
01540 u_vr = vr * cos(quadrantangles(i) + (deg2rad * 90.0_sz))
01541 v_vr = vr * sin(quadrantangles(i) + (deg2rad * 90.0_sz))
01542
01543 IF (strout%ir(icnt,i) > 0) THEN
01544 vmpseudo = vr
01545 vmbwl(i) = sqrt((vmpseudo * cos(quadrantangles(i) + (deg2rad * 90.0_sz)) - &
01546 & stormmotionu)**2.0_sz +
01547 & (vmpseudo * sin(quadrantangles(i) + (deg2rad * 90.0_sz)) - &
01548 & stormmotionv)**2.0_sz) / windreduction
01549
01550 gamma(i) = ((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01551 & sqrt((2.0_sz*u_vr*stormmotionu+2.0_sz*v_vr*stormmotionv)**2.0_sz - &
01552 & 4.0_sz*(stormmotion**2.0_sz-vmbwl(i)**2.0_sz * windreduction**2.0_sz) * &
01553 & vr**2.0_sz)) / (2.0_sz * (stormmotion**2.0_sz -
01554 & vmbwl(i)**2.0_sz * windreduction**2.0_sz))
01555 !gamma(i) = MAX(MIN(gamma(i), 1.0_SZ), 0.0_SZ)
01556
01557 quadrantvr(i) = sqrt((u_vr - gamma(i) * stormmotionu)**2.0_sz +
01558 & (v_vr - gamma(i) * stormmotionv)**2.0_sz) / windreduction
01559 ELSE
01560 vmbwl(i) = vmaxbl
01561 ! gamma = |quadrantVr| / |vMaxBL|
01562 gamma(i) = ((2.0_sz * u_vr * stormmotionu+2.0_sz * v_vr * stormmotionv) - &
01563 & sqrt((2.0_sz * u_vr * stormmotionu+2.0_sz * v_vr * &
01564 & stormmotionv)**2.0_sz -
01565 & 4.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * &
01566 & windreduction**2.0_sz) * vr**2.0_sz)) / &
01567 & (2.0_sz * (stormmotion**2.0_sz-vmaxbl**2.0_sz * windreduction**2.0_sz))
01568 gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)
01569
01570 quadrantvr(i) = (vr - gamma(i) * stormmotion) / windreduction !scalar cal.
01571 END IF
01572 ELSE
01573 vmbwl(i) = vmaxbl
01574 END IF
01575 END DO
01576

```

```

01577      CALL setusequadrantvr(.true.)
01578      CALL newvortexfull(pn, pc, clat, clon, vmaxbl)
01579      strout%hollB(icnt) = getshapeparameter()
01580      CALL setisotachwindspeeds(quadrantvr)
01581      CALL setisotachraddi(r)
01582      CALL setvmaxesbl(vmwbl)
01583      IF (geostrophicswitch .EQV. .true.) THEN
01584          CALL calcrmaxesfull()
01585      ELSE
01586          CALL calcrmaxes()
01587      END IF
01588      CALL getrmaxes(rmaxwtemp)
01589      strout%rMaxW(icnt, :) = rmaxwtemp(:)
01590
01591      CASE(2) !An updated approach: assume quadrantVr has an
01592          ! additional inward angnel quadRotateAngle, and Vr angle is not known
01593          ! calculate quadRotateAngle for the highest isotach, and then
01594          ! use it for other lower isotachs of the same numCycle
01595      vmwb1flag = 0
01596
01597      IF (k == 1) THEN
01598          nquadrot = 300
01599          quadrotateangle(:) = 25.0_sz ! initial guess of inward rotation angle (degree)
01600          rmaxwhighiso(:) = strout%rMaxW(icnt, :)
01601      ELSE
01602          DO i = 1, 4
01603              quadrotateangle(i) = fang(r(i), rmaxwhighiso(i))
01604          END DO
01605          nquadrot = 1
01606      END IF
01607
01608      ! Add loop to converge inward rotation angle
01609      DO iquadrot = 1, nquadrot
01610          vioflag = .false.
01611
01612          DO i = 1, 4
01613              quadrantvecangles(i) = quadrantangles(i) + &
01614                  (90.0_sz + quadrotateangle(i)) * deg2rad
01615          END DO
01616
01617      ! radial direction -> tangential direction ->
01618      ! add inward direction -> the direction of quadrantVr
01619      DO i = 1, 4
01620          IF ((iquadrot == 1) .OR. (vmwb1flag(i) == 0)) THEN
01621              epsilonangle(i)= 360.0_sz + rad2deg * &
01622                  atan2(vmaxbl * sin(quadrantvecangles(i)) + stormmotionv, &
01623                      vmaxbl * cos(quadrantvecangles(i)) + stormmotionu)
01624
01625          IF (epsilonangle(i) > 360.0_sz) THEN
01626              epsilonangle(i) = epsilonangle(i) - 360.0_sz
01627          END IF
01628
01629          u_vr = vr * cos(epsilonangle(i) / rad2deg)
01630          v_vr = vr * sin(epsilonangle(i) / rad2deg)
01631
01632          ! Eliminate the translational speed based on vortex wind speed
01633          ! gamma = |quadrantVr| / |vMaxBL|
01634          gamma(i) = ((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01635          &
01636          - &
01637          &
01638          (2.0_sz * (stormmotion**2.0_sz-vmaxbl**2.0_sz * windreduction**2.0_sz) *
01639          gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)
01640
01641          quadrantvr(i) = sqrt((u_vr - gamma(i) * stormmotionu)**2.0_sz + &
01642                          (v_vr - gamma(i) * stormmotionv)**2.0_sz) / windreduction
01643          END IF
01644      END DO
01645
01646      ! If violation occurs at any quadrant (quadrantVr(i)>vMaxBL),
01647      ! re-calculate quadrantVr at those violated quadrants
01648      DO i = 1, 4
01649          IF ((quadrantvr(i) > vmaxbl) .OR. (vmwb1flag(i) == 1)) THEN
01650              ! Replace vMax with Vr when violation occurs (including
01651              ! situations when isotach is not reported at that quadrant)
01652              IF (iquadrot == 1) vmwb1flag(i) = 1 ! assign violation flags
01653

```

```

01654     IF (strout%ir(icnt,i) > 0) THEN
01655         quadrantvr(i) = (-2.0_sz * (stormmotionu * cos(quadrantvecangles(i)) +
01656                                     stormmotionv * sin(quadrantvecangles(i))) +
01657                                     sqrt(4.0_sz * (stormmotionu *
01658                                         cos(quadrantvecangles(i)) +
01659                                         stormmotionv *
01660                                         sin(quadrantvecangles(i)))**2.0_sz - &
01661                                         4.0_sz * (stormmotion**2.0_sz-vr**2.0_sz))) / 2.0_sz
01662
01663         epsilonangle(i)= 360.0_sz + rad2deg * &
01664             atan2(quadrantvr(i) * sin(quadrantvecangles(i)) + stormmotionv, &
01665             quadrantvr(i) * cos(quadrantvecangles(i)) + stormmotionu)
01666
01667         IF (epsilonangle(i) > 360.0_sz) THEN
01668             epsilonangle(i) = epsilonangle(i) - 360.0_sz
01669         END IF
01670
01671         quadrantvr(i) = quadrantvr(i) / windreduction
01672         vmwbl(i) = quadrantvr(i)
01673     ELSE
01674         vmwbl(i) = vmaxbl
01675         u_vr = vr * sin(strout%dir(icnt) * deg2rad)
01676         v_vr = vr * sin(strout%dir(icnt) * deg2rad)
01677
01678         ! gamma = |quadrantVr| / |vMaxBL|
01679         gamma(i) = ((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01680             & sqrt((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01681             & stormmotionv)**2.0_sz - &
01682             & 4.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * &
01683             & windreduction**2.0_sz) * vr**2.0_sz) / &
01684             & (2.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * windreduction**2.0_sz))
01685         gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)
01686
01687         quadrantvr(i) = (vr - gamma(i) * stormmotion) / windreduction !scalar cal.
01688     END IF
01689     ELSE
01690         vmwbl(i) = vmaxbl
01691     END IF
01692 END DO
01693
01694 CALL setusequadrantvr(.true.)
01695 CALL newvortexfull(pn, pc, clat, clon, vmaxbl)
01696 strout%hollB(icnt) = getshapeparameter()
01697 CALL setisotachwindspeeds(quadrantvr)
01698 CALL setisotachradii(r)
01699 CALL setvmaxesbl(vmwbl)
01700 IF (geostrophicswitch .EQV. .true.) THEN
01701     CALL calcrmaxesfull()
01702 ELSE
01703     CALL calcrmaxes()
01704 END IF
01705 CALL getrmaxes(rmaxwtemp)
01706 strout%rMaxW(icnt, :) = rmaxwtemp(:)
01707
01708 ! add deterministic statement to exit the loop when conditions met
01709 !IF (k == 1) rMaxWHighIso(:) = strOut%rMaxW(iCnT, :) !PV This should be at the beginning of
the loop
01710 DO i = 1, 4
01711     quadrotateangle_new(i) = fang(r(i), rmaxwhighiso(i))
01712     IF (abs(quadrotateangle_new(i) - quadrotateangle(i)) > 0.2_sz) THEN
01713         vioflag(i) = .true.
01714     END IF
01715 END DO
01716 IF ((count(vioflag) >= 1) .AND. (iQuadrot < nQuadrot)) THEN
01717     quadrotateangle(:) = quadrotateangle_new(:)
01718 ELSE
01719     EXIT
01720 END IF
01721
01722 WHERE(.NOT. vioflag) irr = 0
01723 IF ((sum(irr(:)) == 0) .AND. (iQuadrot == 2)) EXIT
01724 END DO ! iQuadRot = 1, nQuadRot
01725
01726 IF ((iQuadrot >= nQuadrot) .AND. (k == 1) .AND. (sum(irr(:)) /= 0)) THEN
01727     !WRITE(*,*) "quadRotateAngle not fully converge, iCnt=", iCnT
01728     WRITE(*,*) "Converge issue at iCnT = ", icnt, " iQuadRot = ", iQuadrot
01729     WRITE(*,*) vioflag, irr

```

```

01730      WRITE(*, '(8(f6.3, x))') quadrotateangle(:, quadrotateangle_new(:)
01731      ELSE
01732          WRITE(scratchmessage,'(a, i0, "", 2x, a, i0)') 'Finished processing iCnt = ', icnt, &
01733                                         'iQuadRot = ', iquadrot
01734          CALL logmessage(info, scratchmessage)
01735      END IF
01736
01737      CASE default
01738          WRITE(*, *) "Wrong approach #, must be 1 or 2"
01739      END SELECT
01740
01741      CALL getvmaxesbl(vmaxesbltemp)
01742      strout%vMaxesBL(icnt, :) = vmaxesbltemp(:)
01743      strout%holBsl(icnt, 1:4) = getshapeparameters()
01744      phifactors(icnt, 1:4) = getphifactors()
01745
01746      ! Reset rmax to zero if there was a zero radius to the isotach for all
01747      ! isotachs EXCEPT the 34 kt isotach. in that case leave the radius that
01748      ! has been substituted.
01749      ! The isotach wind speed can sometimes be zero in cases
01750      ! where all radii are zero (this has been observed in the BEST
01751      ! track file for IGOR2010). Including this possibility in the if
01752      ! statement, so that we can avoid setting the quadrant Rmax to zero if ivr was zero.
01753      DO i=1,4
01754          IF ((strout%ivr(icnt) /= 34) .AND. (strout%ivr(icnt) /= 0) .AND. &
01755             (strout%ir(icnt,i) == 0) ) THEN
01756              strout%rMaxW(icnt, i) = 0.0
01757          END IF
01758      END DO
01759      END DO ! totRecPerCycle
01760  END DO ! iCyc (main do loop)
01761
01762  -----
01763  ! Now indicate which isotach quadrant radius
01764  ! that the user desires ADCIRC to read in
01765  ! for the final calculation of RMX in the
01766  ! Asymmetric Holland wind calculations
01767
01768  ! 34... - 0 0 0 0 ...
01769  ! 50... - 0 0 1 1 ...
01770  ! 64... - 1 1 0 0 ...
01771
01772  ! would indicate -
01773  ! use NO radii from the 34 kt isotach
01774  ! use the 3 & 4 radii form the 50 kt isotach
01775  ! use the 1 & 2 radii form the 64 kt isotach
01776
01777  ! users can then modify the input file
01778  ! to indicate which set of radii to use
01779  ! for each cycle
01780
01781  ! Loop through each cycle and choose
01782  ! the isotach radii to use
01783
01784  ! method 1
01785  ! use the 34kt isotach only (like original NWS=9)
01786
01787  ! method 2
01788  ! use the fancy way of taking the highest
01789  ! isotach Rmax that exists
01790
01791  ! method 3
01792  ! use preferably the 50kt isotach Rmax in each quadrant,
01793  ! if not available, use the 34kt one
01794
01795  ! method 4
01796  ! use all available isotach for each cycle,
01797  ! linearly weighted-combination will be performed in
01798  ! nws20get module
01799
01800  -----
01801  SELECT CASE(method)
01802    CASE(1) ! just use the Rmaxes from the 34kt isotach
01803        DO icnt = 1, numrec
01804            IF ((strout%ivr(icnt) == 34) .OR. (strout%ivr(icnt) == 0)) THEN
01805                strout%quadFlag(icnt, :) = 1
01806            ELSE
01807                strout%quadFlag(icnt, :) = 0
01808            END IF
01809        END DO
01810

```

```

01811 CASE(2) ! use the Rmax from the highest isotach in each quadrant
01812 DO icyc = 1, ncycles
01813   lastentry = sum(totrecpercycle(1:icyc))
01814   firstentry = lastentry - (totrecpercycle(icyc) - 1)
01815   IF (totrecpercycle(icyc) == 1) THEN
01816     strout%quadFlag(lastentry, :) = 1
01817   ELSE
01818     ! loop over quadrants
01819     DO i=1, 4
01820       numnonzero = count(strout%quadFlag(firstentry:lastentry, i) /= 0)
01821       SELECT CASE(numnonzero)
01822         CASE(0,1) ! none, or only 34kt isotach has a radius value
01823           strout%quadFlag(firstentry, i) = 1
01824         CASE(2) ! the 34kt and 50kt isotachs have radius value
01825           strout%quadFlag(firstentry, i) = 0
01826         CASE(3) ! the 34kt, 50kt, and 64kt isotachs have values
01827           strout%quadFlag(firstentry:firstentry + 1, i) = 0
01828         CASE default ! zero isotachs have been flagged
01829           WRITE(scratchmessage,'(i0, a)') numnonzero, ' isotachs were nonzero.'
01830           CALL allmessage(error, scratchmessage)
01831       END SELECT
01832     END DO
01833   END IF
01834 END DO
01835
01836 CASE(3) ! use preferably the Rmax from the 50kt isotach in each quadrant
01837   DO icyc = 1, ncycles
01838     lastentry = sum(totrecpercycle(1:icyc))
01839     firstentry = lastentry - (totrecpercycle(icyc) - 1)
01840     IF (totrecpercycle(icyc) == 1) THEN
01841       strout%quadFlag(lastentry, :) = 1
01842     ELSE
01843       ! loop over quadrants
01844       DO i = 1, 4
01845         numnonzero = count(strout%quadFlag(firstentry:lastentry, i) /= 0)
01846         SELECT CASE(numnonzero)
01847           CASE(0,1) ! none, or only 34kt isotach has a radius value
01848             strout%quadFlag(firstentry, i) = 1
01849           CASE(2) ! the 34kt and 50kt isotachs have radius value
01850             strout%quadFlag(firstentry, i) = 0
01851           CASE(3) ! the 34kt, 50kt, and 64kt isotachs have values
01852             strout%quadFlag(firstentry, i) = 0
01853             strout%quadFlag(lastentry,i) = 0
01854           CASE default ! zero isotachs have been flagged
01855             WRITE(scratchmessage,'(i0, a)') numnonzero, ' isotachs were nonzero.'
01856             CALL allmessage(error, scratchmessage)
01857         END SELECT
01858       END DO
01859     END IF
01860   END DO
01861
01862 CASE(4) ! use all available Rmaxes from multiple reported isotachs
01863   DO icyc = 1, ncycles
01864     lastentry = sum(totrecpercycle(1:icyc))
01865     firstentry = lastentry - (totrecpercycle(icyc) - 1)
01866     ! since strout%quadFlag is previously assigned 1 where (ir(iCnt,:)>0)
01867     ! here we only have to deal with situations when only 0 or 34
01868     ! isotach is reported and with missing ir values
01869     IF (totrecpercycle(icyc) == 1) THEN
01870       strout%quadFlag(lastentry, :) = 1
01871     ELSE
01872       ! loop over quadrants
01873       DO i=1,4
01874         numnonzero = count(strout%quadFlag(firstentry:lastentry, i) /= 0)
01875         SELECT CASE(numnonzero)
01876           CASE(0,1) ! none, or only 34kt isotach has a radius value
01877             strout%quadFlag(firstentry, i) = 1
01878           CASE(2, 3) ! the 34kt, 50kt, and/or 64kt isotachs have values
01879           CASE default
01880             WRITE(scratchmessage,'(i0, a)') numnonzero, ' isotachs were nonzero.'
01881             CALL allmessage(error, scratchmessage)
01882         END SELECT
01883       END DO
01884     END IF
01885   END DO
01886
01887 CASE default
01888   WRITE(scratchmessage,'(i0, a)') 'method = ', method, ' is not valid for setting rmax in
quadrants.'
01889   CALL allmessage(error, scratchmessage)
01890   CALL allmessage(error, 'Execution terminated.')

```

```

01891      END SELECT
01892
01893 ! Persist last good 34kt Rmax values if all radii are missing
01894 DO icyc = 1, ncycles
01895   IF (totrecpercycle(icyc) == 1) THEN
01896     icnt = sum(totrecpercycle(1:icyc))
01897     IF ((all(strout%ir(icnt, :) == 0)) .AND. (strout%iRmw(icnt) == 0)) THEN
01898       IF ((icyc - 1) >= 1) THEN
01899         strout%rMaxW(icnt, :) = strout%rMaxW(icnt - totrecpercycle(icyc - 1), :)
01900       ELSE
01901         strout%rMaxW(icnt, :) = 25 ! default value when all else fails
01902       END IF
01903     END IF
01904   END IF
01905 END DO
01906
01907 !-----
01908 ! Here convert the hurricane speeds to be stored in the data structure
01909 strout%iStormSpeed = nint(strout%stormSpeed)
01910 strout%stormSpeed = kt2ms * strout%stormSpeed
01911 strout%idir = nint(strout%dir)
01912 !
01913
01914 DO icnt = 1, numrec
01915   strout%isotachsPerCycle(icnt) = count(strout%quadFlag(icnt, 1:4) /= 0)
01916 END DO
01917
01918 DEALLOCATE(casttime)
01919 !
01920
01921 CALL writeasymmetricvortexdata(besttrackfilename(idtrfile), strout)
01922
01923 CALL unsetmessagesource()
01924
01925 END SUBROUTINE processasymmetricvortexdata
01926
01927 !=====
01928
01929 !-----
01930 ! S U B R O U T I N E   G E T   H O L L A N D   F I E L D S
01931 !
01932 !
01933
01934 SUBROUTINE gethollandfields(timeIDX)
01935
01936 USE pahm_mesh, ONLY : slam, sfea, np, ismeshok
01937 USE pahm_global, ONLY : rhoair, &
01938                           backgroundatmpress, windreduction, one2ten, &
01939                           deg2rad, rad2deg, basee, omega, mb2pa, mb2kpa, &
01940                           nbrfiles, besttrackfilename, &
01941                           noutdt, mdbegsimtime, mdoutdt, &
01942                           refyear, refmonth, refday, refhour, refmin, refsec, &
01943                           times, datetimes, &
01944                           wvelx, wvely, wpres
01945 USE utilities, ONLY : sphericaldistance, sphericalfracpoint, getlocandratio
01946 USE timedateutils, ONLY : juldaytogram, gregtjulday, gettimeconvsec, datetime2string
01947
01948 IMPLICIT NONE
01949
01950 INTEGER, INTENT(IN) :: timeIDX
01951
01952 INTEGER :: stormNumber ! storm identification number
01953 REAL(SZ) :: hLB ! Holland B parameter
01954 REAL(SZ) :: rrp ! radius of the last closed isobar (m)
01955 REAL(SZ) :: rmw ! radius of max winds (m)
01956 REAL(SZ) :: speed ! maximum sustained wind speed (m/s)
01957 REAL(SZ) :: cPress ! central pressure (Pa)
01958 REAL(SZ) :: cPressDef ! pressure deficit: Ambient Press - cPress
01959
01960 ! (Pa)
01961 REAL(SZ) :: trVX, trVY, trSPD ! storm translation velocities (m/s)
01962 REAL(SZ) :: trSpdX, trSpdY ! adjusted translation velocities (m/s)
01963 REAL(SZ) :: lon, lat ! current eye location
01964
01965 REAL(SZ), ALLOCATABLE :: rad(:) ! distance of nodal points from the eye
01966 location INTEGER, ALLOCATABLE :: radIDX(:) ! indices of nodal points such that rad <=
01967 rrp
01968 INTEGER :: maxRadIDX ! total number of radIDX elements
01969 REAL(SZ) :: windMultiplier ! for storm 2 in lpfs ensemble DO WE NEED
01970 THIS?
01971 REAL(SZ) :: dx, dy, theta
01972 REAL(SZ) :: wtRatio

```

```

01989      REAL(SZ)                      :: coriolis
01990
01991      REAL(SZ)                      :: sfPress          ! calculated surface MSL pressure (Pa)
01992      REAL(SZ)                      :: grVel           ! wind speed (m/s) at gradient level (top
01993      of ABL)
01993      REAL(SZ)                      :: sfVelX, sfVely   ! calculated surface (10-m above ground)
01994      wind velocities (m/s)
01995      INTEGER                        :: iCnt, stCnt, npCnt
01996      INTEGER                        :: i, j11, j12
01997      INTEGER                        :: status
01998
01999      REAL(SZ)                      :: jday
02000      INTEGER                        :: iYear, iMonth, iDay, iHour, iMin, iSec
02001
02002      CHARACTER(LEN=64)            :: tmpTimeStr, tmpStr1, tmpStr2
02003
02004      LOGICAL, SAVE                 :: firstCall = .true.
02005
02006      CALL setmessagesource("GetHollandFields")
02007
02008      ! Check if timeIDX is within bounds (1 <= timeIDX <= nOutDT). If it is not then exit the program.
02009      IF ((timeidx < 1) .OR. (timeidx > noutdt)) THEN
02010          WRITE(tmpstr1, '(a, i0)') 'timeIDX = ', timeidx
02011          WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
02012          WRITE(scratchmessage, '(a)') 'timeIDX should be: 1 <= timeIDX <= nOutDT :' // &
02013              trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
02014          CALL allmessage(error, scratchmessage)
02015
02016          CALL unsetmessagesource()
02017
02018          CALL terminate()
02019      END IF
02020
02021
02022 !#####
02023 !###    BEG:: FIRSTCALL BLOCK
02024 !###        This part of the code should only be executed once
02025 !#####
02026      IF (firstcall) THEN
02027          ! Check if the mesh variables are set and that nOutDT is greater than zero.
02028          IF (.NOT. ismeshok) THEN
02029              WRITE(scratchmessage, '(a)') 'The mesh variables are not established properly.' // &
02030                  'Call subroutine ReadMesh to read/create the mesh topology first.'
02031              CALL allmessage(error, scratchmessage)
02032
02033          CALL unsetmessagesource()
02034
02035          CALL terminate()
02036      ELSE
02037          IF ((np <= 0) .OR. (noutdt <= 0)) THEN
02038              WRITE(tmpstr1, '(a, i0)') 'np = ', np
02039              WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
02040              WRITE(scratchmessage, '(a)') 'Variables "np" or "nOutDT" are not defined properly: ' // &
02041                  trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
02042              CALL allmessage(error, scratchmessage)
02043
02044          CALL unsetmessagesource()
02045
02046          CALL terminate()
02047      END IF
02048  END IF
02049
02050
02051 !-----
02052 ! Allocate storage for the Times and DatesTimes arrays and populate them
02053 ! with the output times and ouput dates respectively.
02054 !-----
02055      ALLOCATE(times(noutdt))
02056      ALLOCATE(datestimes(noutdt))
02057
02058      DO icnt = 1, noutdt
02059          times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
02060          jday = (times(icnt) * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday, refhour,
02061          refmin, refsec)
02062          CALL juldaytoreg(jday, iyear, imonth, iday, ihour, imin, isec)
02063          datestimes(icnt) = trim(adjustl(datetime2string(iyear, imonth, iday, ihour, imin, isec, 0)))
02064      END DO
02065
02066 !-----

```

```

02067 !-----
02068 ! Allocate storage for the output atmospheric field arrays.
02069 ! These arrays share the same mesh with the ocean and wave model
02070 !-----
02071 ALLOCATE (wvelx(np))
02072 ALLOCATE (wvely(np))
02073 ALLOCATE (wpress(np))
02074 !-----
02075
02076
02077 !-----
02078 ! Allocate the Holland data structures and store the Holland
02079 ! data into the data structure array for subsequent use.
02080 ! The Holland structures are allocated by calling the ProcessHollandData
02081 ! subroutine.
02082 ! Process and store the "best track" data into the array of Holland structures
02083 ! for subsequent use. All required data to generate the P-W model wind fields
02084 ! are contained in these structures. We take into consideration that might be
02085 ! more than one "best track" file for the simulation period.
02086 !-----
02087 ALLOCATE (holstru(nbtrfiles))
02088
02089 DO stcnt = 1, nbtrfiles
02090   CALL processhollanddata(stcnt, holstru(stcnt), status)
02091
02092   IF (.NOT. holstru(stcnt)%loaded) THEN
02093     WRITE(scratchmessage, '(a)') 'There was an error loading the Holland data structure for the best
track file: ' // &
02094     trim(adjustl(besttrackfilename(stcnt)))
02095     CALL allmessage(error, scratchmessage)
02096
02097     CALL deallocohllstruct(holstru(stcnt))
02098     DEALLOCATE(holstru)
02099
02100     CALL unsetmessagesource()
02101
02102     CALL terminate()
02103   ELSE IF (status /= 0) THEN
02104     WRITE(scratchmessage, '(a)') 'There was an error processing the Holland data structure for the best
track file: ' // &
02105     trim(adjustl(besttrackfilename(stcnt)))
02106     CALL allmessage(error, scratchmessage)
02107
02108     CALL deallocohllstruct(holstru(stcnt))
02109     DEALLOCATE(holstru)
02110
02111     CALL unsetmessagesource()
02112
02113     CALL terminate()
02114   ELSE
02115     WRITE(scratchmessage, '(a)') 'Processing the Holland data structure for the best track file: '
// &
02116     trim(adjustl(besttrackfilename(stcnt)))
02117     CALL logmessage(info, scratchmessage)
02118   END IF
02119 END DO
02120 !-----
02121
02122
02123 firstcall = .false.
02124 END IF
02125 !#####
02126 !### END:: FIRSTCALL BLOCK
02127 !#####
02128
02129
02130 !-----
02131 ! Initialize the arrays. Here we are resetting the fields to their defaults.
02132 ! This subroutine is called repeatedly and each time the following
02133 ! atmospheric fields are recalculated.
02134 !-----
02135 wvelx = 0.0_sz
02136 wvely = wvelx
02137 wpress = backgroundatmpress * mb2pa
02138 !-----
02139
02140
02141 !-----
02142 ! THIS IS THE MAIN TIME LOOP
02143 ! IT USES "timeIDX" TO ADVANCE THE CALCULATIONS IN TIME
02144 !-----

```

```

02145      icnt = timeidx
02146      WRITE(tmpstr1, '(i5)') icnt
02147      WRITE(tmpstr2, '(i5)') noutdt
02148      tmpstr1 = '(' // trim(tmpstr1) // '//' // trim(adjustl(trimstr2)) // ')'
02149      !WRITE(tmpTimeStr, '(f20.3)') Times(iCnt)
02150      WRITE(tmptimestr, '(a)') datetimes(icnt)
02151      WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(trimstr1)) // " " //
02152      trim(adjustl(tmptimestr))
02153      CALL allmessage(scratchmessage)
02154
02155 DO stcnt = 1, nbtrfiles
02156   ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
02157   ! factor for the subsequent linear interpolation in time. In order for this to
02158   ! work, the array holStru%castTime should be ordered in ascending order.
02159   CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, j11, j12, wtratio)
02160
02161   ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
02162   ! by exiting this loop
02163   IF ((j11 <= 0) .OR. (j12 <= 0)) THEN
02164     WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(tmptimestr)) // &
02165           ', skipping generating data for this time'
02166     CALL logmessage(info, scratchmessage)
02167
02168   cycle
02169 END IF
02170
02171 ! Perform linear interpolation in time
02172 stormnumber = holstru(stcnt)%stormNumber(j11)
02173
02174 CALL sphericalfracpoint(holstru(stcnt)%lat(j11), holstru(stcnt)%lon(j11), &
02175           holstru(stcnt)%lat(j12), holstru(stcnt)%lon(j12), &
02176           wtratio, lat, lon)
02177 !lat    = holStru(stCnT)%lat(j11) + &
02178 !      wtRatio * (holStru(stCnT)%lat(j12) - holStru(stCnT)%lat(j11))
02179 !lon    = holStru(stCnT)%lon(j11) + &
02180 !      wtRatio * (holStru(stCnT)%lon(j12) - holStru(stCnT)%lon(j11))
02181
02182 ! Radius of the last closed isobar
02183 rrp = holstru(stcnt)%rrp(j11) + &
02184           wtratio * (holstru(stcnt)%rrp(j12) - holstru(stcnt)%rrp(j11))
02185
02186 ! Radius of maximum winds
02187 rmw = holstru(stcnt)%rmw(j11) + &
02188           wtratio * (holstru(stcnt)%rmw(j12) - holstru(stcnt)%rmw(j11))
02189
02190 ! Get all the distances of the mesh nodes from (lat, lon)
02191 rad   = sphericaldistance(sfea, slam, lat, lon)
02192 ! ... and the indices of the nodal points where rad <= rrp
02193 radiidx = pack([(i, i = 1, np)], rad <= rrp)
02194 maxradidx = SIZE(radiidx)
02195
02196 ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
02197 IF (maxradidx == 0) THEN
02198   WRITE(tmpstr1, '(f20.3)') rrp
02199   tmpstr1 = '(rrp = ' // trim(adjustl(tmpstr1)) // ' m)'
02200   WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar '
02201 // &
02202           trim(adjustl(tmpstr1)) // ' for storm: ' // &
02203           trim(adjustl(holstru(stcnt)%thisStorm))
02204   CALL logmessage(info, scratchmessage)
02205
02206 EXIT
02207 END IF
02208
02209 speed  = holstru(stcnt)%speed(j11) + &
02210           wtratio * (holstru(stcnt)%speed(j12) - holstru(stcnt)%speed(j11))
02211
02212 cpress = holstru(stcnt)%cPress(j11) + &
02213           wtratio * (holstru(stcnt)%cPress(j12) - holstru(stcnt)%cPress(j11))
02214
02215 trvx   = holstru(stcnt)%trVx(j11) + &
02216           wtratio * (holstru(stcnt)%trVx(j12) - holstru(stcnt)%trVx(j11))
02217
02218 trvy   = holstru(stcnt)%trVy(j11) + &
02219           wtratio * (holstru(stcnt)%trVy(j12) - holstru(stcnt)%trVy(j11))
02220
02221 ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
02222 ! background pressure and return. PV: check if this is actually needed
02223 IF (cpress < 0.0_sz) THEN
02224   wvelx = 0.0_sz
02225   wvely = wvelx
02226   wpress = backgroundatmpress * mb2pa

```

```

02224
02225     WRITE(scratchmessage, '(a)') 'Calm period found, generating zero atmospheric fields for this time'
02226     CALL logmessage(info, scratchmessage)
02227
02228     EXIT
02229 END IF
02230
02231 ! Calculate and limit central pressure deficit; some track files (e.g., Charley 2004)
02232 ! may have a central pressure greater than the ambient pressure that this subroutine assumes
02233 cpressdef = backgroundatmpress * mb2pa - cpress
02234 IF (cpressdef < 100.0_sz) cpressdef = 100.0_sz
02235
02236 ! Subtract the translational speed of the storm from the observed max wind speed to avoid
02237 ! distortion in the Holland curve fit. The translational speed will be added back later.
02238 trspd = sqrt(trvx * trvx + trvy * trvy)
02239 speed = speed - trspd
02240
02241 ! Convert wind speed from 10 meter altitude (which is what the
02242 ! NHC forecast contains) to wind speed at the top of the atmospheric
02243 ! boundary layer (which is what the Holland curve fit requires).
02244 speed = speed / windreduction
02245
02246 ! Calculate Holland parameters and limit the result to its appropriate range.
02247 hlb = rhoair * basee * (speed**2) / cpressdef
02248 IF (hlb < 1.0_sz) hlb = 1.0_sz
02249 IF (hlb > 2.5_sz) hlb = 2.5_sz
02250
02251 ! If we are running storm 2 in the Lake Pontchartrain !PV Do we need this?
02252 ! Forecast System ensemble, the final wind speeds should be multiplied by 1.2.
02253 windmultiplier = 1.0_sz
02254 IF (stormnumber == 2) windmultiplier = 1.2_sz
02255
02256 DO npcnt = 1, maxradidx
02257   i = radidx(npcnt)
02258
02259   !dx    = SphericalDistance(lat, lon, lat, slam(i))
02260   !dy    = SphericalDistance(lat, lon, sfea(i), lon)
02261   dx    = deg2rad * (slam(i) - lon)
02262   dy    = deg2rad * (sfea(i) - lat)
02263
02264   theta = atan2(dy, dx)
02265
02266   ! Compute coriolis
02267   coriolis = 2.0_sz * omega * sin(sfea(i) * deg2rad)
02268
02269   ! Compute the pressure (Pa) at a distance rad(i); all distances are in meters
02270   sfpress = cpress + cpressdef * exp(-(rmw / rad(i))**hlb)
02271
02272   ! Compute wind speed (speed - trSPD) at gradient level (m/s) and at a distance rad(i);
02273   ! all distances are in meters. Using absolute value for coriolis for Southern Hemisphere
02274   grvel = sqrt(speed**2 * (rmw / rad(i))**hlb * exp(1.0_sz - (rmw / rad(i))**hlb) +
02275             (rad(i) * abs(coriolis) / 2.0_sz)**2) -
02276             rad(i) * abs(coriolis) / 2.0_sz
02277
02278   ! Determine translation speed that should be added to final !PV CHECK ON THIS
02279   ! storm wind speed. This is tapered to zero as the storm tapers
02280   ! to zero toward the eye of the storm and at long distances from the storm.
02281   trspd_x = (abs(grvel) / speed) * trvx
02282   trspd_y = (abs(grvel) / speed) * trvy
02283
02284   ! Apply mutliplier for Storm #2 in LPFS ensemble.
02285   grvel = grvel * windmultiplier
02286
02287   ! Find the wind velocity components.
02288   sfvelx = -grvel * sin(theta)
02289   sfvely = grvel * cos(theta)
02290
02291   ! Convert wind velocity from the gradient level (top of atmospheric boundary layer)
02292   ! which, is what the Holland curve fit produces, to 10-m wind velocity.
02293   sfvelx = sfvelx * windreduction
02294   sfvely = sfvely * windreduction
02295
02296   ! Convert from 1-minute averaged winds to 10-minute averaged winds.
02297   sfvelx = sfvelx * one2ten
02298   sfvely = sfvely * one2ten
02299
02300   ! Add back the storm translation speed.
02301   sfvelx = sfvelx + trspd_x
02302   sfvely = sfvely + trspd_y
02303
02304   !PV Need to account for multiple storms in the basin

```

```

02305      ! Need to interpolate between storms if this nodal point
02306      ! is affected by more than one storm
02307      wpress(i) = sfpress
02308      wvelx(i)  = sfvelx
02309      wvely(i)  = sfvely
02310      END DO ! npCnt = 1, maxRadIDX
02311      END DO ! stCnt = 1, nBTrFiles
02312
02313
02314      !-----
02315      ! Deallocate the arrays
02316      !-----
02317      IF (ALLOCATED(rad)) DEALLOCATE(rad)
02318      IF (ALLOCATED(radidx)) DEALLOCATE(radidx)
02319      !DO iCnt = 1, nBTrFiles
02320      ! CALL DeAllocHollStruct(holStru(iCnt))
02321      !END DO
02322      !DEALLOCATE(holStru)
02323      !-----
02324
02325      CALL unsetmessagesource()
02326
02327      END SUBROUTINE gethollandfields
02328
02329 !=====
02330
02331      !-----
02332      ! S U B R O U T I N E   G E T   G A H M   F I E L D S
02333      !-----
02335      !
02336      SUBROUTINE getgahmfields(timeIDX)
02337
02338      USE pahm_mesh, ONLY      : slam, sfea, np, ismeshok
02339      USE pahm_global, ONLY    : backgroundatpress, one2ten,
02340                                deg2rad, rad2deg, basee, omega, kt2ms, mb2pa, mb2kpa, m2nm, nm2m, rearth, &
02341                                nbtrfiles, besttrackfilename,
02342                                noutdt, mdbegsimtime, mdoutdt,
02343                                refyear, refmonth, refday, refhour, refmin, refsec,
02344                                times, datestimes,
02345                                wvelx, wvely, wpress
02346      USE utilities, ONLY     : touppercase, sphericaldistance, sphericalfracpoint, getlocandratio
02347      USE timedateutils, ONLY : juldaytogreg, gregtojulday, gettimeconvsec, datetimetostring, timeconv
02348      USE pahm_vortex, ONLY    : spinterp, fitrmaxes4, rmaxes4, quadflag4, quadir4, bs4, vmb14, &
02349                                setvortex, uvpr
02350
02351      IMPLICIT NONE
02352
02353      INTEGER, INTENT(IN)      :: timeIDX
02354
02355      REAL(SZ)                 :: coriolis ! Coriolis force (1/s)
02356
02357      INTEGER                   :: iCnt, kCnt, stCnt
02358      INTEGER                   :: i, j11, j12
02359      INTEGER                   :: status
02360
02361      REAL(SZ)                 :: jday
02362      INTEGER                   :: iYear, iMonth, iDay, iHour, iMin, iSec
02363
02364      CHARACTER(LEN=64)         :: tmpTimeStr, tmpStr1, tmpStr2
02365
02366
02367      INTEGER                   :: maxTrackRecords
02368      INTEGER, SAVE              :: iCyc, iSot ! do we need to save this?
02369      INTEGER, DIMENSION(:, :, :, :, :), ALLOCATABLE, SAVE :: ir, quadFlag
02370      REAL(SZ), DIMENSION(:, :, :, :, :), ALLOCATABLE, SAVE :: rMaxW, hollBs, vMaxesBL
02371
02372      REAL(SZ)                 :: stormMotion ! portion of Vmax attributable to storm motion
02373      REAL(SZ)                 :: stormMotionU ! U portion of Vmax attributable to storm motion
02374      REAL(SZ)                 :: stormMotionV ! V portion of Vmax attributable to storm motion
02375
02376      CHARACTER(LEN = 10), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: stormName
02377      CHARACTER(LEN = 4), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: castType ! hindcast/nowcast or forecast?
02378      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: stormNumber ! storm identification number
02379      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: year, month, day, hour
02380      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: iRmw
02381      REAL(SZ), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: lat, lon
02382      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: iSpeed, iCPress
02383      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: fcstInc ! hours between forecasts
02384      REAL(SZ), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: hollB
02385      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: nCycles
02386      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: numCycle

```

```

02407    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE           :: isotachsPerCycle
02408
02409    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE           :: ipn
02410    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE           :: ivr
02411
02412    REAL(SZ), DIMENSION(:, :, ), ALLOCATABLE, SAVE          :: cycleTime
02413    REAL(SZ), DIMENSION(:, :, ), ALLOCATABLE, SAVE          :: uTrans, vTrans
02414    REAL(SZ), DIMENSION(:, :, ), ALLOCATABLE, SAVE          :: hSpeed, hDir
02415
02416    REAL(SZ), ALLOCATABLE, SAVE :: cHollBs1(:, ), cHollBs2(:, )
02417    REAL(SZ), ALLOCATABLE, SAVE :: cPhiFactor1(:, ), cPhiFactor2(:, )
02418    REAL(SZ), ALLOCATABLE, SAVE :: cVmwBL1(:, ), cVmwBL2(:, )
02419    REAL(SZ), ALLOCATABLE, SAVE :: crmaxwl(:, ), crmaxw2(:, )
02420    REAL(SZ), ALLOCATABLE, SAVE :: crmaxwTrue1(:, ), crmaxwTrue2(:, )
02421    REAL(SZ), ALLOCATABLE, SAVE :: cHollBs(:, )
02422    REAL(SZ), ALLOCATABLE, SAVE :: cPhiFactor(:, )
02423    REAL(SZ), ALLOCATABLE, SAVE :: cVmwBL(:, )
02424    REAL(SZ), ALLOCATABLE, SAVE :: crmaxw(:, )
02425    REAL(SZ), ALLOCATABLE, SAVE :: crmaxwTrue(:, )
02426
02427    REAL(SZ), SAVE           :: uTransNow, vTransNow ! time-interpolated overland speed, kts
02428    REAL(SZ), SAVE           :: dirNow ! Jie
02429
02430    REAL(SZ), SAVE           :: pn      ! Ambient surface pressure (mb)
02431    REAL(SZ), SAVE           :: pc      ! Surface pressure at center of storm (mb)
02432    REAL(SZ), SAVE           :: cLat, cLon ! Current eye location (degrees north, degrees
02433    east)
02434    REAL(SZ)                 :: wtRatio
02435
02436    REAL(SZ), DIMENSION(:, ), ALLOCATABLE, SAVE :: dx, dy, dist, azimuth
02437
02438    LOGICAL, SAVE             :: firstCall = .true.
02439
02440    CALL setmessagesource("GetGAHMFields")
02441
02442    ! Check if timeIDX is within bounds (1 <= timeIDX <= nOutDT). If it is not then exit the program.
02443    IF ((timeidx < 1) .OR. (timeidx > noutdt)) THEN
02444        WRITE(tmpstr1, '(a, i0)' ) 'timeIDX = ', timeidx
02445        WRITE(tmpstr2, '(a, i0)' ) 'nOutDT = ', noutdt
02446        WRITE(scratchmessage, '(a)' ) 'timeIDX should be: 1 <= timeIDX <= nOutDT :' // &
02447                    trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
02448        CALL allmessage(error, scratchmessage)
02449
02450        CALL unsetmessagesource()
02451
02452        CALL terminate()
02453
02454
02455 !#####
02456 !###    BEG::: FIRSTCALL BLOCK
02457 !###    This part of the code should only be executed once
02458 !#####
02459    IF (firstcall) THEN
02460        ! Check if the mesh variables are set and that nOutDT is greater than zero.
02461        IF (.NOT. ismeshok) THEN
02462            WRITE(scratchmessage, '(a)' ) 'The mesh variables are not established properly. ' // &
02463                                'Call subroutine ReadMesh to read/create the mesh topology first.'
02464            CALL allmessage(error, scratchmessage)
02465
02466            CALL unsetmessagesource()
02467
02468            CALL terminate()
02469
02470        ELSE
02471            IF ((np <= 0) .OR. (noutdt <= 0)) THEN
02472                WRITE(tmpstr1, '(a, i0)' ) 'np = ', np
02473                WRITE(tmpstr2, '(a, i0)' ) 'nOutDT = ', noutdt
02474                WRITE(scratchmessage, '(a)' ) 'Variables "np" or "nOutDT" are not defined properly: ' // &
02475                                trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
02476                CALL allmessage(error, scratchmessage)
02477
02478                CALL unsetmessagesource()
02479
02480            CALL terminate()
02481        END IF
02482
02483
02484    !-----
02485    ! Allocate storage for the Times and DatesTimes arrays and populate them
02486    ! with the output times and ouput dates respectively.

```

```

02487 !-----
02488 ALLOCATE(times(noutdt))
02489 ALLOCATE(datestimes(noutdt))
02490
02491
02492 DO icnt = 1, noutdt
02493   times(icnt) = mdbegsimtime + (icnt - 1) * mdoutdt
02494   jday = (times(icnt) * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday, refhour,
02495   refmin, refsec)
02496   CALL juldaytoreg(jday, iyear, imonth, iday, ihour, imin, isec)
02497   datestimes(icnt) = trim(adjustl(datetime2string(iyear, imonth, iday, ihour, imin, isec, 0)))
02498 END DO
02499 !-----
02500
02501 !-----
02502 ! Allocate storage for the output atmospheric field arrays.
02503 ! These arrays share the same mesh with the ocean and wave model
02504 !-----
02505 ALLOCATE(wvelx(np))
02506 ALLOCATE(wvely(np))
02507 ALLOCATE(wpress(np))
02508
02509 ALLOCATE(chollbs1(np), chollbs2(np))
02510 ALLOCATE(cphifactor1(np), cphifactor2(np))
02511 ALLOCATE(cvmwb1(np), cvmwbl2(np))
02512 ALLOCATE(crmawwl(np), crmaxw2(np))
02513 ALLOCATE(crmawtrue1(np), crmaxwtrue2(np))
02514 ALLOCATE(chollbs(np))
02515 ALLOCATE(cphifactor(np))
02516 ALLOCATE(cvmwb1(np))
02517 ALLOCATE(crmaw(np))
02518 ALLOCATE(crmawtrue(np))
02519 ALLOCATE(dx(np), dy(np), dist(np), azimuth(np))
02520 !-----
02521
02522
02523 !-----
02524 ! Allocate the asymmetric vortex data structures.
02525 ! The asymmetric vortex structures are allocated by calling the
02526 ! ProcessAsymmetricVortexData subroutine.
02527 ! Process and store the "best track" data into the array of asymmetric vortex structures
02528 ! for subsequent use. All required data to generate the P-W model wind fields
02529 ! are contained in these structures. We take into consideration that might be
02530 ! more than one "best track" file for the simulation period.
02531 !-----
02532 ALLOCATE(asyvortstru(nbtrfiles))
02533
02534 DO stcnt = 1, nbtrfiles
02535   CALL processasymmetricvortexdata(stcnt, asyvortstru(stcnt), status)
02536
02537   IF (.NOT. asyvortstru(stcnt)%loaded) THEN
02538     WRITE(scratchmessage, '(a)') 'There was an error loading the asymmetric vortex data structure' //
02539     &                                         ' for the best track file: ' //
02540     trim(adjustl(besttrackfilename(stcnt)))
02541     CALL alimessage(error, scratchmessage)
02542
02543     CALL deallocasymvortstruct(asyvortstru(stcnt))
02544     DEALLOCATE(asyvortstru)
02545
02546     CALL unsetmessagesource()
02547
02548     CALL terminate()
02549     ELSE IF (status /= 0) THEN
02550       WRITE(scratchmessage, '(a)') 'There was an error loading the asymmetric vortex data structure' //
02551     &                                         ' for the best track file: ' //
02552     trim(adjustl(besttrackfilename(stcnt)))
02553     CALL alimessage(error, scratchmessage)
02554
02555     CALL deallocasymvortstruct(asyvortstru(stcnt))
02556     DEALLOCATE(asyvortstru)
02557
02558     CALL unsetmessagesource()
02559
02560     CALL terminate()
02561     ELSE
02562       WRITE(scratchmessage, '(a)') 'Processing the asymmetric vortex data structure for the best track
file: ' // &
02563                                         trim(adjustl(besttrackfilename(stcnt)))

```

```

02562      CALL logmessage(info, scratchmessage)
02563      END IF
02564  END DO
02565
02566  maxtrackrecords = maxval(asyvortstru(1:nbtrfiles)%numRec)
02567
02568  ALLOCATE(casttype(nbtrfiles, maxtrackrecords))
02569  ALLOCATE(stormnumber(nbtrfiles, maxtrackrecords))
02570  ALLOCATE(year(nbtrfiles, maxtrackrecords))
02571  ALLOCATE(month(nbtrfiles, maxtrackrecords))
02572  ALLOCATE(day(nbtrfiles, maxtrackrecords))
02573  ALLOCATE(hour(nbtrfiles, maxtrackrecords))
02574  ALLOCATE(lat(nbtrfiles, maxtrackrecords))
02575  ALLOCATE(lon(nbtrfiles, maxtrackrecords))
02576  ALLOCATE(irmw(nbtrfiles, maxtrackrecords))
02577  ALLOCATE(ispeed(nbtrfiles, maxtrackrecords))
02578  ALLOCATE(icpress(nbtrfiles, maxtrackrecords))
02579  ALLOCATE(fcstinc(nbtrfiles, maxtrackrecords))
02580  ALLOCATE(hollb(nbtrfiles, maxtrackrecords))
02581  ALLOCATE(ncycles(nbtrfiles, maxtrackrecords))
02582  ALLOCATE(numcycle(nbtrfiles, maxtrackrecords))
02583  ALLOCATE(isotachspercycle(nbtrfiles, maxtrackrecords))
02584  ALLOCATE(ipn(nbtrfiles, maxtrackrecords))
02585  ALLOCATE(ivr(nbtrfiles, maxtrackrecords))
02586  ALLOCATE(cycletime(nbtrfiles, maxtrackrecords))
02587  ALLOCATE(utrans(nbtrfiles, maxtrackrecords))
02588  ALLOCATE(vtrans(nbtrfiles, maxtrackrecords))
02589  ALLOCATE(hspeed(nbtrfiles, maxtrackrecords))
02590  ALLOCATE(hdir(nbtrfiles, maxtrackrecords))
02591  ALLOCATE(stormname(nbtrfiles, maxtrackrecords))
02592
02593  ALLOCATE(ir(nbtrfiles, maxtrackrecords, 4, 4))
02594  ALLOCATE(rmaxw(nbtrfiles, maxtrackrecords, 4, 4))
02595  ALLOCATE(quadflag(nbtrfiles, maxtrackrecords, 4, 4))
02596  ALLOCATE(hollbs(nbtrfiles, maxtrackrecords, 4, 4))
02597  ALLOCATE(vmaxesbl(nbtrfiles, maxtrackrecords, 4, 4))
02598
02599
02600 !-----
02601 ! Initialize the variables
02602 !-----
02603 ir      = 0
02604 rmaxw   = 0.0_sz
02605 quadflag = 0
02606 hollbs  = 0.0_sz
02607 vmaxesbl = 0.0_sz
02608 !-----
02609
02610
02611 DO stcnt = 1, nbtrfiles
02612
02613 ! Loop through records in input data structure
02614 DO kcmt = 1, asyvortstru(stcnt)%numRec
02615   ! pick out the cycle data that the user wants to use
02616   ! by looping through quads
02617   IF (kcmt == 1) THEN
02618     icyc = 1
02619     isot = 1
02620
02621     stormnumber(stcnt, icyc) = asyvortstru(stcnt)%stormNumber(kcmt)
02622     year(stcnt, icyc)       = asyvortstru(stcnt)%year(kcmt)
02623     month(stcnt, icyc)      = asyvortstru(stcnt)%month(kcmt)
02624     day(stcnt, icyc)        = asyvortstru(stcnt)%day(kcmt)
02625     hour(stcnt, icyc)       = asyvortstru(stcnt)%hour(kcmt)
02626     casttype(stcnt, icyc)   = asyvortstru(stcnt)%castType(kcmt)
02627     fcstinc(stcnt, icyc)    = asyvortstru(stcnt)%fcstInc(kcmt)
02628     lat(stcnt, icyc)        = asyvortstru(stcnt)%lat(kcmt)
02629     lon(stcnt, icyc)        = asyvortstru(stcnt)%lon(kcmt)
02630     ispeed(stcnt, icyc)     = asyvortstru(stcnt)%iSpeed(kcmt)
02631     icpress(stcnt, icyc)    = asyvortstru(stcnt)%iCPress(kcmt)
02632     ivr(stcnt, icyc)        = asyvortstru(stcnt)%ivr(kcmt)
02633     ipn(stcnt, icyc)        = asyvortstru(stcnt)%iPrp(kcmt)
02634     irmw(stcnt, icyc)       = asyvortstru(stcnt)%iRmw(kcmt)
02635
02636     hdir(stcnt, icyc)       = asyvortstru(stcnt)%iDir(kcmt)
02637     hspeed(stcnt, icyc)     = asyvortstru(stcnt)%iStormSpeed(kcmt)
02638     stormname(stcnt, icyc)  = asyvortstru(stcnt)%stormName(kcmt)
02639     numcycle(stcnt, icyc)   = asyvortstru(stcnt)%numCycle(kcmt)
02640     hollb(stcnt, icyc)      = asyvortstru(stcnt)%hollB(kcmt)
02641
02642     utrans(stcnt, icyc)     = asyvortstru(stcnt)%trVx(kcmt)

```

```

02643     vtrans(stcnt, icyc)      = asyvortstru(stcnt)%trVy(kcnt)
02644
02645     CALL timeconv(year(stcnt, icyc), month(stcnt, icyc), day(stcnt, icyc), &
02646         0, 0, cycletime(stcnt, icyc))
02647
02648     isotachspercycle(stcnt, icyc) = asyvortstru(stcnt)%isotachsPerCycle(kcnt)
02649
02650     DO i = 1, 4
02651         IF (asyvortstru(stcnt)%quadFlag(kcnt, i) == 1) THEN
02652             ir(stcnt, icyc, i, isot)      = asyvortstru(stcnt)%ir(kcnt, i)
02653             rmaxw(stcnt, icyc, i, isot)   = asyvortstru(stcnt)%rMaxW(kcnt, i)
02654             quadflag(stcnt, icyc, i, isot) = asyvortstru(stcnt)%quadFlag(kcnt, i)
02655             hollbs(stcnt, icyc, i, isot)  = asyvortstru(stcnt)%hollBs(kcnt, i)
02656             vmaxesbl(stcnt, icyc, i, isot) = asyvortstru(stcnt)%vMaxesBL(kcnt, i)
02657
02658     END IF
02659     END DO
02660
02661     ELSE ! kCnt == 1
02662         IF (asyvortstru(stcnt)%numCycle(kcnt) == asyvortstru(stcnt)%numCycle(kcnt-1)) THEN
02663             IF (isot > 4) THEN
02664                 WRITE(scratchmessage,'(a, i0, a)')
02665                 'The GAHM asymmetric data structure has more than 4 iSotachs in cycle ', &
02666                 asyvortstru(stcnt)%numCycle(kcnt), '.'
02667                 CALL allmessage(error, scratchmessage)
02668                 CALL terminate()
02669             END IF
02670             isot = isot + 1 ! same iCyc, next iSotach
02671         ELSE
02672             isot = 1 ! initialize iSotach #
02673             IF (asyvortstru(stcnt)%fcstInc(kcnt) == 0 .AND. asyvortstru(stcnt)%fcstInc(icyc) == 0) THEN
02674                 icyc = icyc
02675             ELSE
02676                 icyc = icyc + 1
02677             END IF
02678         END IF
02679
02680         stormnumber(stcnt, icyc) = asyvortstru(stcnt)%stormNumber(kcnt)
02681         year(stcnt, icyc)        = asyvortstru(stcnt)%year(kcnt)
02682         month(stcnt, icyc)       = asyvortstru(stcnt)%month(kcnt)
02683         day(stcnt, icyc)        = asyvortstru(stcnt)%day(kcnt)
02684         casttype(stcnt, icyc)    = asyvortstru(stcnt)%castType(kcnt)
02685         fcstinc(stcnt, icyc)     = asyvortstru(stcnt)%fcstInc(kcnt)
02686         lat(stcnt, icyc)         = asyvortstru(stcnt)%lat(kcnt)
02687         lon(stcnt, icyc)         = asyvortstru(stcnt)%lon(kcnt)
02688         ispeed(stcnt, icyc)      = asyvortstru(stcnt)%iSpeed(kcnt)
02689         icpress(stcnt, icyc)     = asyvortstru(stcnt)%iCPress(kcnt)
02690         ivr(stcnt, icyc)         = asyvortstru(stcnt)%ivr(kcnt)
02691         ipn(stcnt, icyc)         = asyvortstru(stcnt)%iPrp(kcnt)
02692         irmw(stcnt, icyc)        = asyvortstru(stcnt)%iRmw(kcnt)
02693         hdir(stcnt, icyc)        = asyvortstru(stcnt)%iDir(kcnt)
02694         hspeed(stcnt, icyc)      = asyvortstru(stcnt)%iStormSpeed(kcnt)
02695         stormname(stcnt, icyc)    = asyvortstru(stcnt)%stormName(kcnt)
02696         numcycle(stcnt, icyc)     = asyvortstru(stcnt)%numCycle(kcnt)
02697         hollb(stcnt, icyc)        = asyvortstru(stcnt)%hollB(kcnt)
02698
02699         utrans(stcnt, icyc)       = asyvortstru(stcnt)%trVx(kcnt)
02700         vtrans(stcnt, icyc)       = asyvortstru(stcnt)%trVy(kcnt)
02701
02702         CALL timeconv(year(stcnt, icyc), month(stcnt, icyc), day(stcnt, icyc), &
02703             0, 0, cycletime(stcnt, icyc))
02704
02705         isotachspercycle(stcnt, icyc) = asyvortstru(stcnt)%isotachsPerCycle(kcnt)
02706
02707         DO i = 1, 4
02708             IF (asyvortstru(stcnt)%quadFlag(kcnt, i) == 1) THEN
02709                 ir(stcnt, icyc, i, isot)      = asyvortstru(stcnt)%ir(kcnt, i)
02710                 rmaxw(stcnt, icyc, i, isot)   = asyvortstru(stcnt)%rMaxW(kcnt, i)
02711                 quadflag(stcnt, icyc, i, isot) = asyvortstru(stcnt)%quadFlag(kcnt, i)
02712                 hollbs(stcnt, icyc, i, isot)  = asyvortstru(stcnt)%hollBs(kcnt, i)
02713                 vmaxesbl(stcnt, icyc, i, isot) = asyvortstru(stcnt)%vMaxesBL(kcnt, i)
02714
02715             END IF
02716         END DO
02717         END IF ! kCnt /= 1
02718     END DO ! asyVortStru(stCnT)%numRec
02719
02720     ncycles(stcnt) = asyvortstru(stcnt)%nCycles
02721
02722     firstcall = .false.
02723 END IF

```

```

02724 !#####
02725 !###      END:: FIRSTCALL BLOCK
02726 !#####
02727
02728 !-----
02729 ! Initialize the arrays. Here we are resetting the fields to their defaults.
02730 ! This subroutine is called repeatedly and each time the following
02731 ! atmospheric fields are recalculated.
02732 !-----
02733 wvelx = 0.0_sz
02734 wvely = wvelx
02735 wpress = backgroundatmpress * mb2pa
02736 !-----
02737
02738
02739 !-----
02740 ! THIS IS THE MAIN TIME LOOP
02741 ! IT USES "timeIDX" TO ADVANCE THE CALCULATIONS IN TIME
02742 !-----
02743 icnt = timeidx
02744     WRITE(tmpstr1, '(i5)') icnt
02745     WRITE(tmpstr2, '(i5)') noutdt
02746 tmpstr1 = (' // trim(tmpstr1) // ' // trim(adjustl(trimstr2)) // ')
02747     WRITE(tmpimestr, '(a)') datetimes(icnt)
02748     WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(trimstr1)) // " " //
02749 trim(adjustl(trimimestr))
02750     CALL allmessage(scratchmessage)
02751
02752 DO stcnt = 1, nbtrfiles
02753
02754 ! Get the bin interval where Times(iCnt) is bounded and the corresponding ratio
02755 ! factor for the subsequent linear interpolation in time. In order for this to
02756 ! work, the array asyVortStru%castTime should be ordered in ascending order.
02757 !PV (iCyc, iCyc - 1) = (j12, j11) j11: lower limit and j12: upper limit
02758     CALL getlocandratio(times(icnt), cycletime(stcnt, 1:ncycles(stcnt)), j11, j12, wtratio)
02759
02760 ! Skip the subsequent calculations if Times(iCnt) is outside the castTime range
02761 ! by exiting this loop
02762 IF ((j11 <= 0) .OR. (j12 <= 0)) THEN
02763     WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(trimimestr)) // &
02764             ', skipping generating data for this time'
02765     CALL logmessage(info, scratchmessage)
02766
02767     cycle
02768 END IF
02769
02770 IF ((touppercase(trim(adjustl(casttype(stcnt, j12)))) == 'CALM') .OR. &
02771     (touppercase(trim(adjustl(casttype(stcnt, j11)))) == 'CALM')) THEN
02772     wpress(:) = backgroundatmpress * mb2pa
02773     wvelx(:) = 0.0_sz
02774     wvely(:) = 0.0_sz
02775     cycle
02776 END IF
02777
02778 ! Perform linear interpolation in time
02779     CALL sphericalfracpoint(lat(stcnt, j11), lon(stcnt, j11), lat(stcnt, j12), lon(stcnt, j12), wtratio,
02780     clat, clon)
02781
02782 !---- Calculate distance/azimuth of points in CPP plane
02783 dx = deg2rad * rearth * (slam - clon) * cos(deg2rad * clat)
02784 dy = deg2rad * rearth * (sfea - clat)
02785 dist = sqrt(dx * dx + dy * dy) * m2nm
02786 azimuth = 360.0_sz + rad2deg * atan2(dx, dy)
02787 WHERE(azimuth > 360.0_sz) azimuth = azimuth - 360.0_sz
02788 !----
02789 quadflag4(2:5, 1:4) = quadflag(stcnt, j11, 1:4, 1:4)
02790 quadir4(2:5, 1:4) = real(ir(stcnt, j11, 1:4, 1:4))
02791 rmaxes4(2:5, 1:4) = rmaxw(stcnt, j11, 1:4, 1:4)
02792 bs4(2:5, 1:4) = hollbs(stcnt, j11, 1:4, 1:4)
02793 vmb14(2:5, 1:4) = vmaxesbl(stcnt, j11, 1:4, 1:4)
02794
02795     CALL fitrmaxes4()
02796
02797 DO i = 1, np
02798     crmaxwl(i) = spinterp(azimuth(i), dist(i), 1)
02799     crmaxwtrue1(i) = spinterp(azimuth(i), 1.0_sz, 1)
02800
02801 !an artificial number 1.0 is chosen to ensure only
02802 !rmax from the highest isotach is picked

```

```

02803     chollbs1(i)      = spinterp(azimuth(i), dist(i), 2)
02804     cvmwbl1(i)      = spinterp(azimuth(i), dist(i), 3)
02805   END DO
02806
02807     quadflag4(2:5, 1:4) = quadflag(stcnt, jl2, 1:4, 1:4)
02808     quadir4(2:5, 1:4)  = real(ir(stcnt, jl2, 1:4, 1:4))
02809     rmaxes4(2:5, 1:4)  = rmaxw(stcnt, jl2, 1:4, 1:4)
02810     bs4(2:5, 1:4)      = hollbs(stcnt, jl2, 1:4, 1:4)
02811     vmb14(2:5, 1:4)    = vmaxesbl(stcnt, jl2, 1:4, 1:4)
02812
02813   CALL fitrmaxes4()
02814
02815   DO i = 1, np
02816     crmaxw2(i)      = spinterp(azimuth(i), dist(i), 1)
02817     crmaxwtrue2(i) = spinterp(azimuth(i), 1.0_sz, 1)
02818
02819     !an artificial number 1.0 is chosen to ensure only
02820     !rmax from the highest isotach is picked
02821     chollbs2(i)      = spinterp(azimuth(i), dist(i), 2)
02822     cvmwbl2(i)      = spinterp(azimuth(i), dist(i), 3)
02823   END DO
02824
02825   pn      = 1.0_sz * (ipn(stcnt, jl1) + wtratio*(ipn(stcnt, jl2)-ipn(stcnt, jl1)))
02826   pc      = 1.0_sz * (icpress(stcnt, jl1) + &
02827                         wtratio * (icpress(stcnt, jl2) - icpress(stcnt, jl1)))
02828   crmaxw  = crmaxwl(:) + &
02829                         wtratio * (crmaxw2(:) - crmaxwl(:))
02830   crmaxwtrue = crmaxwtrue1(:) + &
02831                         wtratio * (crmaxwtrue2(:) - crmaxwtrue1(:))
02832   chollbs = chollbs1(:) + &
02833                         wtratio * (chollbs2(:) - chollbs1(:))
02834   cvmwbl  = cvmwbl1(:) + &
02835                         wtratio * (cmvwbl2(:) - cvmwbl1(:))
02836   coriolis = 2.0_sz * omega * sin(deg2rad * clat)
02837
02838   DO i = 1, np
02839     cphifactor(i) = 1 + cvmwbl(i) * kt2ms * crmaxw(i) * nm2m * coriolis / &
02840                           (chollbs(i)* ((cmvwbl(i) * kt2ms)**2 + &
02841                                         cvmwbl(i) * kt2ms * crmaxw(i) * nm2m * coriolis))
02842   END DO
02843
02844   utransnow = utrans(stcnt, jl1) + wtratio * (utrans(stcnt, jl2) - utrans(stcnt, jl1))
02845   vtransnow = vtrans(stcnt, jl1) + wtratio * (vtrans(stcnt, jl2) - vtrans(stcnt, jl1))
02846
02847 !-----
02848 ! Create a new asymmetric hurricane vortex.
02849 !
02850 ! Note: Subtract translational speed from Vmax, then
02851 ! scale (Vmax - Vt) and Vr up to the top of the surface,
02852 ! where the cyclostrophic wind balance is valid.
02853 !-----
02854 !-----
02855 ! Calculate wind and pressure fields at model nodal points.
02856 !
02857 ! Note: the asymmetric vortex wind speed is reduced from the
02858 ! top of the surface layer to the surface, then converted from
02859 ! a 1-minute (max sustained) to a 10-minute average prior to
02860 ! adding the translational velocity in subroutine uvpr.
02861 !-----
02862   stormmotion = 1.5_sz * (sqrt(utransnow**2.0_sz + vtransnow**2.0_sz))**0.63_sz
02863   dirnow = rad2deg * atan2(utransnow, vtransnow)
02864   IF (dirnow < 0.0_sz) dirnow = dirnow + 360.0_sz
02865   stormmotionu = sin(dirnow / rad2deg) * stormmotion
02866   stormmotionv = cos(dirnow / rad2deg) * stormmotion
02867   CALL setvortex(pn, pc, clat, clon)
02868
02869 !PV Need to account for multiple storms in the basin
02870   DO i = 1, np
02871     CALL uvpr(dist(i), azimuth(i), crmaxw(i), crmaxwtrue(i), &
02872               chollbs(i), cvmwbl(i), cphifactor(i), stormmotionu, &
02873               stormmotionv, geofactor, wvelx(i), wvely(i), wpress(i))
02874   END DO
02875
02876 END DO ! stCnt = 1, nBTrFiles
02877
02878 !-----
02879 ! Deallocate the arrays
02880 !-----
02881 !DO iCnT = 1, nBTrFiles
02882 !  CALL DeAllocAsymVortStruct(asyVortStru(iCnT))
02883 !END DO

```

```

02884 !DEALLOCATE(asyVortStru)
02885 !-----
02886
02887 CALL unsetmessagesource()
02888
02889 END SUBROUTINE getgahmfields
02890
02891 !=====
02892 !-----
02893 !----- S U B R O U T I N E W R I T E B E S T T R A C K D A T A -----
02894 !-----
02895 !-----
02912 SUBROUTINE writebesttrackdata(inpFile, trackStruc, suffix)
02913
02914 USE pahm_global, ONLY : lun_btrk, lun_btrkl
02915
02916 IMPLICIT NONE
02917
02918 ! Global variables
02919 CHARACTER(LEN=*) , INTENT(IN) :: inpFile
02920 TYPE(besttrackdata_t), INTENT(IN) :: trackStruc
02921 CHARACTER(LEN=*) , OPTIONAL, INTENT(IN) :: suffix
02922
02923 ! Local variables
02924 CHARACTER(LEN=FNAMELEN) :: outFile
02925 CHARACTER(LEN=64) :: fSuf
02926 INTEGER :: iCnt
02927 INTEGER :: iUnit, errIO
02928 CHARACTER(LEN=512) :: fmtStr
02929
02930
02931 !----- Initialize variables
02932 iunit = lun_btrk
02933 errio = 0
02934
02935 fmtstr = '(a2, "", 1x, i2.2, "", 1x, a10, "", 1x, i2, "", 1x, a4, "", 1x, i3, "",'
02936 fmtstr = trim(fmtstr) // ' 1x, i3, a1, "", 1x, i4, a1, "",'
02937 fmtstr = trim(fmtstr) // ' 1x, i3, "", 1x, i4, "", 1x, a2, "", 1x, i3, "", 1x, a3, "",'
02938 fmtstr = trim(fmtstr) // ' 4(1x, i4, ""), 1x, i4, "", 1x, i4, "", 1x, i3, "", 1x, i3,
",",
02939 fmtstr = trim(fmtstr) // ' 1x, a3,"", 1x, i3,"", 1x, a3, "", 1x, i3, "", 1x, i3, "",'
02940 fmtstr = trim(fmtstr) // ' 1x, a10, "", 1x, i4, "")'
02941 !-----
02942
02943 fsuf = '_adj'
02944 IF (PRESENT(suffix)) fsuf = adjustl(suffix)
02945
02946 CALL setmessagesource("WriteBestTrackData")
02947
02948 IF (.NOT. trackstruc%loaded) THEN
02949   WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be
written."
02950   CALL allmessage(info, scratchmessage)
02951
02952   RETURN
02953 END IF
02954
02955 outfile = trim(adjustl(inpfile)) // trim(fsuf)
02956
02957 WRITE(scratchmessage, '(a)') 'Writting the "adjusted" best track data to: ' // trim(adjustl(outfile))
02958 CALL logmessage(info, scratchmessage)
02959
02960 OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)
02961
02962 IF (errio /= 0) THEN
02963   WRITE(scratchmessage, '(a)') 'Error opening the outFile: ' // trim(outfile) // &
02964   ', skip writting the "adjusted" best track fields'
02965   CALL allmessage(error, scratchmessage)
02966
02967   RETURN
02968 END IF
02969
02970 DO icnt = 1, trackstruc%numRec
02971   WRITE(iunit, fmtstr) &
02972     trackstruc%basin(icnt), trackstruc%cyNum(icnt), &
02973     trackstruc%dtg(icnt), trackstruc%techNum(icnt), &
02974     trackstruc%tech(icnt), trackstruc%tau(icnt), &
02975     trackstruc%intLat(icnt), trackstruc%ns(icnt), &
02976     trackstruc%intLon(icnt), trackstruc%ew(icnt), &
02977     trackstruc%intVMax(icnt), trackstruc%intMsdp(icnt), &

```

```

02978      trackstruc%ty(icnt),           trackstruc%rad(icnt),          &
02979      trackstruc%windCode(icnt),    trackstruc%intRad1(icnt),        &
02980      trackstruc%intRad2(icnt),    trackstruc%intRad3(icnt),        &
02981      trackstruc%intRad4(icnt),    trackstruc%intPOuter(icnt),       &
02982      trackstruc%intROuter(icnt),  trackstruc%intRmw(icnt),         &
02983      trackstruc%gusts(icnt),     trackstruc%eye(icnt),           &
02984      trackstruc%subregion(icnt), trackstruc%maxseas(icnt),        &
02985      trackstruc%initials(icnt), trackstruc%dir(icnt),            &
02986      trackstruc%intSpeed(icnt),   trackstruc%stormName(icnt),       &
02987      trackstruc%cycleNum(icnt)
02988  END DO
02989
02990  CLOSE(iunit)
02991
02992  CALL unsetmessagesource()
02993
02994 END SUBROUTINE writebesttrackdata
02995
02996 !=====
02997
02998 !-----
02999 ! S U B R O U T I N E   W R I T E   A S Y M M E T R I C   V O R T E X   D A T A
03000 !-----
03016 !
03017 SUBROUTINE writeasymmetricvortexdata(inpFile, trackStruc, suffix)
03018
03019 USE pahm_global, ONLY : lun_btrk, lun_btrkl
03020
03021 IMPLICIT NONE
03022
03023 ! Global variables
03024 CHARACTER(LEN=*), INTENT(IN) :: inpFile
03025 TYPE(asymmetricvortexdata_t), INTENT(IN) :: trackStruc
03026 CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: suffix
03027
03028 ! Local variables
03029 INTEGER :: i
03030 CHARACTER(LEN=FNAMELEN) :: outFile
03031 CHARACTER(LEN=64) :: fsuf
03032 INTEGER :: iCnt
03033 INTEGER :: iUnit, errIO
03034 CHARACTER(LEN=512) :: fmtStr
03035
03036
03037 !----- Initialize variables
03038 iunit = lun_btrkl
03039 errio = 0
03040
03041 fmtstr = '(a2, "", 1x, i2.2, "", 1x, a10, "", 1x, i2, "", 1x, a4, "", 1x, i3, "",'
03042 fmtstr = trim(fmtstr) // ' 1x, i3, a1, "", 1x, i4, a1, "",'
03043 fmtstr = trim(fmtstr) // ' 1x, i3, "", 1x, i4, "", 1x, a2, "", 1x, i3, "", 1x, a3, "",'
03044 fmtstr = trim(fmtstr) // ' 4(1x, i4, ""), 1x, i4, "", 1x, i4, "", 1x, i3, "", 1x, i3,
03045 ",",
03046 fmtstr = trim(fmtstr) // ' 1x, a3, "", 1x, i3, "", 1x, a3, "", 1x, i3, "", 1x, i3, ","
03047 fmtstr = trim(fmtstr) // ' 1x, a10, "", 1x, i4, "",'
03048 fmtstr = trim(fmtstr) // ' 1x, i4, "", 4(1x, i1, ""), 2x, 4(1x, f6.1, ""), 9(1x, f8.4, "")'
03049 !-----
03050
03051 fsuf = '_asymvort'
03052 IF (PRESENT(suffix)) fsuf = adjustl(suffix)
03053
03054 CALL setmessagesource("WriteAsymmetricVortexData")
03055
03056 IF (.NOT. trackstruc%loaded) THEN
03057   WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be
written."
03058   CALL allmessage(info, scratchmessage)
03059
03060   RETURN
03061 END IF
03062
03063 outfile = trim(adjustl(inpfile)) // trim(fsuf)
03064
03065 WRITE(scratchmessage, '(a)') 'Writting the "extended" best track data to: ' // trim(adjustl(outfile))
03066 CALL logmessage(info, scratchmessage)
03067
03068 OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)
03069
03070 IF (errio /= 0) THEN
03071   WRITE(scratchmessage, '(a)') 'Error opening the outFile: ' // trim(outfile) // &
03072   ', skip writting the "extended" best track fields'

```

```

03072     CALL allmessage(error, scratchmessage)
03073
03074     RETURN
03075 END IF
03076
03077 DO icnt = 1, trackstruc%numRec
03078     WRITE(iunit, fmtstr)
03079         trackstruc%basin(icnt), trackstruc%stormNumber(icnt),
03080         trackstruc%dtg(icnt), trackstruc%castTypeNum(icnt),
03081         trackstruc%castType(icnt), trackstruc%fcstInc(icnt),
03082         trackstruc%ilat(icnt), trackstruc%ns(icnt),
03083         trackstruc%ilon(icnt), trackstruc%ew(icnt),
03084         trackstruc%ispeed(icnt), trackstruc%iCPress(icnt),
03085         trackstruc%ty(icnt), trackstruc%ivr(icnt),
03086         trackstruc%windCode(icnt), (trackstruc%ir(icnt, i), i = 1, 4),
03087         trackstruc%ipr(icnt), trackstruc%irp(icnt),
03088         trackstruc%irmw(icnt),
03089         trackstruc%gusts(icnt), trackstruc%eye(icnt),
03090         trackstruc%subregion(icnt), trackstruc%maxseas(icnt),
03091         trackstruc%initials(icnt),
03092         trackstruc%idir(icnt), trackstruc%istormSpeed(icnt),
03093         trackstruc%stormName(icnt), trackstruc%numCycle(icnt),
03094         trackstruc%isotachsPerCycle(trackstruc%numCycle(icnt)),
03095         (trackstruc%quadFlag(icnt, i), i = 1, 4), (trackstruc%rMaxW(icnt, i), i = 1, 4),
03096         trackstruc%hollB(icnt), (trackstruc%hollBs(icnt, i), i = 1, 4),
03097         (trackstruc%vMaxesBL(icnt, i), i = 1, 4)
03098     END DO
03099
03100     CLOSE(iunit)
03101
03102     CALL unsetmessagesource()
03103
03104 END SUBROUTINE writeasymmetricvortexdata
03105
03106 !=====
03107
03108 !-----+
03109 ! S U B R O U T I N E   A L L O C   B T R   S T R U C T
03110 !-----+
03111
03112 !-----+
03113
03114 SUBROUTINE allocbtrstruct(str, nRec)
03115
03116     IMPLICIT NONE
03117
03118     TYPE(besttrackdata_t), INTENT(INOUT) :: str
03119     INTEGER, INTENT(IN) :: nRec
03120
03121     str%numRec = nrec
03122     str%loaded = .false.
03123
03124     !---- Input parameters
03125     IF (.NOT. ALLOCATED(str%basin)) ALLOCATE(str%basin(nrec))
03126     IF (.NOT. ALLOCATED(str%cyNum)) ALLOCATE(str%cyNum(nrec))
03127     IF (.NOT. ALLOCATED(str%dtg)) ALLOCATE(str%dtg(nrec))
03128     IF (.NOT. ALLOCATED(str%techNum)) ALLOCATE(str%techNum(nrec))
03129     IF (.NOT. ALLOCATED(str%tech)) ALLOCATE(str%tech(nrec))
03130     IF (.NOT. ALLOCATED(str%tau)) ALLOCATE(str%tau(nrec))
03131     IF (.NOT. ALLOCATED(str%intLat)) ALLOCATE(str%intLat(nrec))
03132     IF (.NOT. ALLOCATED(str%intLon)) ALLOCATE(str%intLon(nrec))
03133     IF (.NOT. ALLOCATED(str%ew)) ALLOCATE(str%ew(nrec))
03134     IF (.NOT. ALLOCATED(str%ns)) ALLOCATE(str%ns(nrec))
03135     IF (.NOT. ALLOCATED(str%intVMax)) ALLOCATE(str%intVMax(nrec))
03136     IF (.NOT. ALLOCATED(str%intMslp)) ALLOCATE(str%intMslp(nrec))
03137     IF (.NOT. ALLOCATED(str%ty)) ALLOCATE(str%ty(nrec))
03138     IF (.NOT. ALLOCATED(str%rad)) ALLOCATE(str%rad(nrec))
03139     IF (.NOT. ALLOCATED(str%windCode)) ALLOCATE(str%windCode(nrec))
03140     IF (.NOT. ALLOCATED(str%intRad1)) ALLOCATE(str%intRad1(nrec))
03141     IF (.NOT. ALLOCATED(str%intRad2)) ALLOCATE(str%intRad2(nrec))
03142     IF (.NOT. ALLOCATED(str%intRad3)) ALLOCATE(str%intRad3(nrec))
03143     IF (.NOT. ALLOCATED(str%intRad4)) ALLOCATE(str%intRad4(nrec))
03144     IF (.NOT. ALLOCATED(str%intPOuter)) ALLOCATE(str%intPOuter(nrec))
03145     IF (.NOT. ALLOCATED(str%intROuter)) ALLOCATE(str%intROuter(nrec))
03146     IF (.NOT. ALLOCATED(str%intRmw)) ALLOCATE(str%intRmw(nrec))
03147     IF (.NOT. ALLOCATED(str%gusts)) ALLOCATE(str%gusts(nrec))
03148     IF (.NOT. ALLOCATED(str%eye)) ALLOCATE(str%eye(nrec))
03149     IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
03150     IF (.NOT. ALLOCATED(str%maxseas)) ALLOCATE(str%maxseas(nrec))
03151     IF (.NOT. ALLOCATED(str%initials)) ALLOCATE(str%initials(nrec))
03152     IF (.NOT. ALLOCATED(str%dir)) ALLOCATE(str%dir(nrec))
03153     IF (.NOT. ALLOCATED(str%intSpeed)) ALLOCATE(str%intSpeed(nrec))
03154     IF (.NOT. ALLOCATED(str%stormName)) ALLOCATE(str%stormName(nrec))

```

```

03165    IF (.NOT. ALLOCATED(str%cycleNum))  ALLOCATE(str%cycleNum(nrec))
03166
03167 !----- Converted parameters
03168    IF (.NOT. ALLOCATED(str%year))      ALLOCATE(str%year(nrec))
03169    IF (.NOT. ALLOCATED(str%month))     ALLOCATE(str%month(nrec))
03170    IF (.NOT. ALLOCATED(str%day))       ALLOCATE(str%day(nrec))
03171    IF (.NOT. ALLOCATED(str%hour))     ALLOCATE(str%hour(nrec))
03172    IF (.NOT. ALLOCATED(str%lat))      ALLOCATE(str%lat(nrec))
03173    IF (.NOT. ALLOCATED(str%lon))      ALLOCATE(str%lon(nrec))
03174
03175 END SUBROUTINE allocbtrstruct
03176
03177 !=====
03178 !
03179 !----- S U B R O U T I N E   D E A L L O C   B T R   S T R U C T
03180 !
03181 !
03182 !
03183 SUBROUTINE deallocbtrstruct(str)
03184
03185 IMPLICIT NONE
03186
03187 TYPE(besttrackdata_t), INTENT(INOUT) :: str
03188
03189 str%numRec = -1
03190 str%loaded = .false.
03191
03192 !----- Input parameters
03193 IF (ALLOCATED(str%basin))      DEALLOCATE(str%basin)
03194 IF (ALLOCATED(str%cyNum))      DEALLOCATE(str%cyNum)
03195 IF (ALLOCATED(str%dtg))       DEALLOCATE(str%dtg)
03196 IF (ALLOCATED(str%techNum))    DEALLOCATE(str%techNum)
03197 IF (ALLOCATED(str%tech))      DEALLOCATE(str%tech)
03198 IF (ALLOCATED(str%tau))       DEALLOCATE(str%tau)
03199 IF (ALLOCATED(str%intLat))    DEALLOCATE(str%intLat)
03200 IF (ALLOCATED(str%intLon))    DEALLOCATE(str%intLon)
03201 IF (ALLOCATED(str%ew))        DEALLOCATE(str%ew)
03202 IF (ALLOCATED(str%ns))        DEALLOCATE(str%ns)
03203 IF (ALLOCATED(str%intVMax))   DEALLOCATE(str%intVMax)
03204 IF (ALLOCATED(str%intMslp))   DEALLOCATE(str%intMslp)
03205 IF (ALLOCATED(str%ty))        DEALLOCATE(str%ty)
03206 IF (ALLOCATED(str%rad))      DEALLOCATE(str%rad)
03207 IF (ALLOCATED(str%windCode))  DEALLOCATE(str%windCode)
03208 IF (ALLOCATED(str%intRad1))   DEALLOCATE(str%intRad1)
03209 IF (ALLOCATED(str%intRad2))   DEALLOCATE(str%intRad2)
03210 IF (ALLOCATED(str%intRad3))   DEALLOCATE(str%intRad3)
03211 IF (ALLOCATED(str%intRad4))   DEALLOCATE(str%intRad4)
03212 IF (ALLOCATED(str%intPOuter)) DEALLOCATE(str%intPOuter)
03213 IF (ALLOCATED(str%intROuter)) DEALLOCATE(str%intROuter)
03214 IF (ALLOCATED(str%intRmw))   DEALLOCATE(str%intRmw)
03215 IF (ALLOCATED(str%gusts))    DEALLOCATE(str%gusts)
03216 IF (ALLOCATED(str%eye))      DEALLOCATE(str%eye)
03217 IF (ALLOCATED(str%subregion)) DEALLOCATE(str%subregion)
03218 IF (ALLOCATED(str%maxseas))  DEALLOCATE(str%maxseas)
03219 IF (ALLOCATED(str%initials)) DEALLOCATE(str%initials)
03220 IF (ALLOCATED(str%dir))      DEALLOCATE(str%dir)
03221 IF (ALLOCATED(str%intSpeed))  DEALLOCATE(str%intSpeed)
03222 IF (ALLOCATED(str%stormName)) DEALLOCATE(str%stormName)
03223 IF (ALLOCATED(str%cycleNum))  DEALLOCATE(str%cycleNum)
03224
03225 !----- Converted parameters
03226 IF (ALLOCATED(str%year))      DEALLOCATE(str%year)
03227 IF (ALLOCATED(str%month))     DEALLOCATE(str%month)
03228 IF (ALLOCATED(str%day))       DEALLOCATE(str%day)
03229 IF (ALLOCATED(str%hour))     DEALLOCATE(str%hour)
03230 IF (ALLOCATED(str%lat))      DEALLOCATE(str%lat)
03231 IF (ALLOCATED(str%lon))      DEALLOCATE(str%lon)
03232
03233 END SUBROUTINE deallocbtrstruct
03234
03235 !=====
03236 !
03237 !----- S U B R O U T I N E   A L L O C   H O L L   S T R U C T
03238 !
03239 !
03240 !
03241 !
03242 !
03243 !
03244 !
03245 !
03246 !
03247 !
03248 !
03249 !
03250 !
03251 !
03252 !
03253 SUBROUTINE allocollstruct(str, nRec)
03254
03255 IMPLICIT NONE
03256
03257 TYPE(hollanddata_t), INTENT(INOUT) :: str

```

```

03268      INTEGER, INTENT(IN) :: nRec
03269
03270      str%numRec = nrec
03271      str%loaded = .false.
03272
03273      !---- Input parameters
03274      IF (.NOT. ALLOCATED(str%basin))           ALLOCATE(str%basin(nrec))
03275
03276      IF (.NOT. ALLOCATED(str%dtg))              ALLOCATE(str%dtg(nrec))
03277      IF (.NOT. ALLOCATED(str%stormNumber))       ALLOCATE(str%stormNumber(nrec))
03278      IF (.NOT. ALLOCATED(str%year))              ALLOCATE(str%year(nrec))
03279      IF (.NOT. ALLOCATED(str%month))             ALLOCATE(str%month(nrec))
03280      IF (.NOT. ALLOCATED(str%day))               ALLOCATE(str%day(nrec))
03281      IF (.NOT. ALLOCATED(str%hour))              ALLOCATE(str%hour(nrec))
03282
03283      IF (.NOT. ALLOCATED(str%castTime))          ALLOCATE(str%castTime(nrec))
03284      IF (.NOT. ALLOCATED(str%castType))          ALLOCATE(str%castType(nrec))
03285      IF (.NOT. ALLOCATED(str%fcstInc))           ALLOCATE(str%fcstInc(nrec))
03286
03287      IF (.NOT. ALLOCATED(str%iLat))              ALLOCATE(str%iLat(nrec))
03288      IF (.NOT. ALLOCATED(str%lat))               ALLOCATE(str%lat(nrec))
03289      IF (.NOT. ALLOCATED(str%ilon))              ALLOCATE(str%ilon(nrec))
03290      IF (.NOT. ALLOCATED(str%lon))               ALLOCATE(str%lon(nrec))
03291
03292      IF (.NOT. ALLOCATED(str%iSpeed))            ALLOCATE(str%iSpeed(nrec))
03293      IF (.NOT. ALLOCATED(str%speed))             ALLOCATE(str%speed(nrec))
03294
03295      IF (.NOT. ALLOCATED(str%icPress))           ALLOCATE(str%icPress(nrec))
03296      IF (.NOT. ALLOCATED(str%cPress))            ALLOCATE(str%cPress(nrec))
03297
03298      IF (.NOT. ALLOCATED(str%irrp))              ALLOCATE(str%irrp(nrec))
03299      IF (.NOT. ALLOCATED(str%rrp))               ALLOCATE(str%rrp(nrec))
03300
03301      IF (.NOT. ALLOCATED(str%irmw))              ALLOCATE(str%irmw(nrec))
03302      IF (.NOT. ALLOCATED(str%rmw))               ALLOCATE(str%rmw(nrec))
03303
03304      IF (.NOT. ALLOCATED(str%cPrDt))             ALLOCATE(str%cPrDt(nrec))
03305
03306      IF (.NOT. ALLOCATED(str%trVx))              ALLOCATE(str%trVx(nrec))
03307      IF (.NOT. ALLOCATED(str%trVy))              ALLOCATE(str%trVy(nrec))
03308
03309  END SUBROUTINE allochollstruct
03310
03311 !=====
03312
03313 !-----+
03314 ! S U B R O U T I N E   D E A L L O C   H O L L   S T R U C T
03315 !-----+
03316
03317 SUBROUTINE deallochollstruct(str)
03318
03319     IMPLICIT NONE
03320
03321     TYPE(hollanddata_t), INTENT(INOUT) :: str
03322
03323     str%numRec = -1
03324     str%loaded = .false.
03325
03326     !---- Input parameters
03327     IF (ALLOCATED(str%basin))           DEALLOCATE(str%basin)
03328
03329     IF (ALLOCATED(str%dtg))              DEALLOCATE(str%dtg)
03330     IF (ALLOCATED(str%stormNumber))     DEALLOCATE(str%stormNumber)
03331     IF (ALLOCATED(str%year))             DEALLOCATE(str%year)
03332     IF (ALLOCATED(str%month))            DEALLOCATE(str%month)
03333     IF (ALLOCATED(str%day))              DEALLOCATE(str%day)
03334     IF (ALLOCATED(str%hour))              DEALLOCATE(str%hour)
03335
03336     IF (ALLOCATED(str%castTime))         DEALLOCATE(str%castTime)
03337     IF (ALLOCATED(str%castType))         DEALLOCATE(str%castType)
03338     IF (ALLOCATED(str%fcstInc))          DEALLOCATE(str%fcstInc)
03339
03340     IF (ALLOCATED(str%iLat))             DEALLOCATE(str%iLat)
03341     IF (ALLOCATED(str%lat))              DEALLOCATE(str%lat)
03342     IF (ALLOCATED(str%ilon))             DEALLOCATE(str%ilon)
03343     IF (ALLOCATED(str%lon))              DEALLOCATE(str%lon)
03344
03345     IF (ALLOCATED(str%icPress))          DEALLOCATE(str%icPress)
03346
03347     IF (ALLOCATED(str%irrp))             DEALLOCATE(str%irrp)
03348     IF (ALLOCATED(str%rrp))              DEALLOCATE(str%rrp)
03349
03350     IF (ALLOCATED(str%irmw))             DEALLOCATE(str%irmw)
03351     IF (ALLOCATED(str%rmw))              DEALLOCATE(str%rmw)
03352
03353     IF (ALLOCATED(str%cPrDt))            DEALLOCATE(str%cPrDt)
03354
03355     IF (ALLOCATED(str%trVx))             DEALLOCATE(str%trVx)
03356     IF (ALLOCATED(str%trVy))              DEALLOCATE(str%trVy)
03357
03358     IF (ALLOCATED(str%trVz))              DEALLOCATE(str%trVz)

```

```

03359   IF (ALLOCATED(str%cPress))      DEALLOCATE(str%cPress)
03360
03361   IF (ALLOCATED(str%iRrp))        DEALLOCATE(str%iRrp)
03362   IF (ALLOCATED(str%rrp))        DEALLOCATE(str%rrp)
03363
03364   IF (ALLOCATED(str%iRmw))        DEALLOCATE(str%iRmw)
03365   IF (ALLOCATED(str%rmw))        DEALLOCATE(str%rmw)
03366
03367   IF (ALLOCATED(str%cPrDt))      DEALLOCATE(str%cPrDt)
03368
03369   IF (ALLOCATED(str%trVx))       DEALLOCATE(str%trVx)
03370   IF (ALLOCATED(str%trVy))       DEALLOCATE(str%trVy)
03371
03372 END SUBROUTINE deallochollstruct
03373
03374 !=====
03375
03376 !-----
03377 ! S U B R O U T I N E   A L L O C   A S Y M   V O R T   S T R U C T
03378 !-----
03391 !
03392 SUBROUTINE allocasymvortstruct(str, nRec)
03393
03394 IMPLICIT NONE
03395
03396 TYPE(asymmetricvortexdata_t), INTENT(INOUT) :: str
03397 INTEGER, INTENT(IN)                      :: nRec
03398
03399 str%numRec = nrec
03400 str%loaded = .false.
03401
03402 !---- Input parameters
03403 IF (.NOT. ALLOCATED(str%basin))          ALLOCATE(str%basin(nrec))
03404
03405 IF (.NOT. ALLOCATED(str%dtg))            ALLOCATE(str%dtg(nrec))
03406 IF (.NOT. ALLOCATED(str%stormNumber))     ALLOCATE(str%stormNumber(nrec))
03407 IF (.NOT. ALLOCATED(str%year))           ALLOCATE(str%year(nrec))
03408 IF (.NOT. ALLOCATED(str%month))          ALLOCATE(str%month(nrec))
03409 IF (.NOT. ALLOCATED(str%day))            ALLOCATE(str%day(nrec))
03410 IF (.NOT. ALLOCATED(str%hour))           ALLOCATE(str%hour(nrec))
03411
03412 IF (.NOT. ALLOCATED(str%castTime))       ALLOCATE(str%castTime(nrec))
03413 IF (.NOT. ALLOCATED(str%castTypeNum))    ALLOCATE(str%castTypeNum(nrec))
03414 IF (.NOT. ALLOCATED(str%castType))       ALLOCATE(str%castType(nrec))
03415 IF (.NOT. ALLOCATED(str%fcstInc))        ALLOCATE(str%fcstInc(nrec))
03416
03417 IF (.NOT. ALLOCATED(str%iLat))           ALLOCATE(str%iLat(nrec))
03418 IF (.NOT. ALLOCATED(str%lat))            ALLOCATE(str%lat(nrec))
03419 IF (.NOT. ALLOCATED(str%ilLon))          ALLOCATE(str%ilLon(nrec))
03420 IF (.NOT. ALLOCATED(str%lon))            ALLOCATE(str%lon(nrec))
03421 IF (.NOT. ALLOCATED(str%ew))             ALLOCATE(str%ew(nrec))
03422 IF (.NOT. ALLOCATED(str%ns))             ALLOCATE(str%ns(nrec))
03423
03424 IF (.NOT. ALLOCATED(str%isSpeed))        ALLOCATE(str%isSpeed(nrec))
03425 IF (.NOT. ALLOCATED(str%speed))          ALLOCATE(str%speed(nrec))
03426
03427 IF (.NOT. ALLOCATED(str%icPress))        ALLOCATE(str%icPress(nrec))
03428 IF (.NOT. ALLOCATED(str%cPress))          ALLOCATE(str%cPress(nrec))
03429
03430 IF (.NOT. ALLOCATED(str%ty))              ALLOCATE(str%ty(nrec))
03431
03432 IF (.NOT. ALLOCATED(str%ivr))            ALLOCATE(str%ivr(nrec))
03433 IF (.NOT. ALLOCATED(str%windCode))        ALLOCATE(str%windCode(nrec))
03434 IF (.NOT. ALLOCATED(str%ir))              ALLOCATE(str%ir(nrec, 4))
03435
03436 IF (.NOT. ALLOCATED(str%iprP))           ALLOCATE(str%iprP(nrec))
03437 IF (.NOT. ALLOCATED(str%prp))            ALLOCATE(str%prp(nrec))
03438 IF (.NOT. ALLOCATED(str%irRp))           ALLOCATE(str%irRp(nrec))
03439 IF (.NOT. ALLOCATED(str%rrp))            ALLOCATE(str%rrp(nrec))
03440
03441 IF (.NOT. ALLOCATED(str%iRmw))           ALLOCATE(str%iRmw(nrec))
03442 IF (.NOT. ALLOCATED(str%rmw))            ALLOCATE(str%rmw(nrec))
03443
03444 IF (.NOT. ALLOCATED(str%gusts))          ALLOCATE(str%gusts(nrec))
03445 IF (.NOT. ALLOCATED(str%eye))             ALLOCATE(str%eye(nrec))
03446 IF (.NOT. ALLOCATED(str%subregion))      ALLOCATE(str%subregion(nrec))
03447 IF (.NOT. ALLOCATED(str%maxseas))         ALLOCATE(str%maxseas(nrec))
03448 IF (.NOT. ALLOCATED(str%initials))        ALLOCATE(str%initials(nrec))
03449
03450 IF (.NOT. ALLOCATED(str%trVx))           ALLOCATE(str%trVx(nrec))
03451 IF (.NOT. ALLOCATED(str%trVy))           ALLOCATE(str%trVy(nrec))

```

```

03452
03453   IF (.NOT. ALLOCATED(str%idir))           ALLOCATE(str%idir(nrec))
03454   IF (.NOT. ALLOCATED(str%dir))             ALLOCATE(str%dir(nrec))
03455   IF (.NOT. ALLOCATED(str%iStormSpeed))     ALLOCATE(str%iStormSpeed(nrec))
03456   IF (.NOT. ALLOCATED(str%stormSpeed))       ALLOCATE(str%stormSpeed(nrec))
03457   IF (.NOT. ALLOCATED(str%stormName))        ALLOCATE(str%stormName(nrec))
03458
03459   IF (.NOT. ALLOCATED(str%numCycle))         ALLOCATE(str%numCycle(nrec))
03460   IF (.NOT. ALLOCATED(str%isotachsPerCycle)) ALLOCATE(str%isotachsPerCycle(nrec))
03461   IF (.NOT. ALLOCATED(str%quadFlag))          ALLOCATE(str%quadFlag(nrec, 4))
03462   IF (.NOT. ALLOCATED(str%rMaxW))            ALLOCATE(str%rMaxW(nrec, 4))
03463   IF (.NOT. ALLOCATED(str%hollB))             ALLOCATE(str%hollB(nrec))
03464   IF (.NOT. ALLOCATED(str%hollBs))            ALLOCATE(str%hollBs(nrec, 4))
03465   IF (.NOT. ALLOCATED(str%vMaxesBL))          ALLOCATE(str%vMaxesBL(nrec, 4))
03466
03467 END SUBROUTINE allocasymvortstruct
03468
03469 !=====
03470
03471 !-----
03472 !----- S U B R O U T I N E   D E A L L O C   A S Y M   V O R T   S T R U C T
03473 !-----
03474 !-----
03475 SUBROUTINE deallocasymvortstruct(str)
03476
03477 IMPLICIT NONE
03478
03479 TYPE(asymmetricvortexdata_t), INTENT(INOUT) :: str
03480
03481 str%numRec = -1
03482 str%loaded = .false.
03483
03484 !----- Input parameters
03485 IF (ALLOCATED(str%basin))           DEALLOCATE(str%basin)
03486
03487 IF (ALLOCATED(str%dtg))             DEALLOCATE(str%dtg)
03488 IF (ALLOCATED(str%stormNumber))     DEALLOCATE(str%stormNumber)
03489 IF (ALLOCATED(str%year))            DEALLOCATE(str%year)
03490 IF (ALLOCATED(str%month))           DEALLOCATE(str%month)
03491 IF (ALLOCATED(str%day))             DEALLOCATE(str%day)
03492 IF (ALLOCATED(str%hour))           DEALLOCATE(str%hour)
03493
03494 IF (ALLOCATED(str%castTime))       DEALLOCATE(str%castTime)
03495 IF (ALLOCATED(str%castTypeNum))    DEALLOCATE(str%castTypeNum)
03496 IF (ALLOCATED(str%castType))       DEALLOCATE(str%castType)
03497 IF (ALLOCATED(str%fcstInc))        DEALLOCATE(str%fcstInc)
03498
03499 IF (ALLOCATED(str%iLat))           DEALLOCATE(str%iLat)
03500 IF (ALLOCATED(str%lat))            DEALLOCATE(str%lat)
03501 IF (ALLOCATED(str%iLon))           DEALLOCATE(str%iLon)
03502 IF (ALLOCATED(str%lon))            DEALLOCATE(str%lon)
03503 IF (ALLOCATED(str%ew))             DEALLOCATE(str%ew)
03504 IF (ALLOCATED(str%ns))             DEALLOCATE(str%ns)
03505
03506 IF (ALLOCATED(str%iSpeed))         DEALLOCATE(str%iSpeed)
03507 IF (ALLOCATED(str%speed))          DEALLOCATE(str%speed)
03508
03509 IF (ALLOCATED(str%icPress))        DEALLOCATE(str%icPress)
03510 IF (ALLOCATED(str%cPress))          DEALLOCATE(str%cPress)
03511
03512 IF (ALLOCATED(str%qty))            DEALLOCATE(str%qty)
03513
03514 IF (ALLOCATED(str%iprp))           DEALLOCATE(str%iprp)
03515 IF (ALLOCATED(str%prp))             DEALLOCATE(str%prp)
03516 IF (ALLOCATED(str%irrp))            DEALLOCATE(str%irrp)
03517 IF (ALLOCATED(str%rrp))             DEALLOCATE(str%rrp)
03518
03519 IF (ALLOCATED(str%irmw))           DEALLOCATE(str%irmw)
03520 IF (ALLOCATED(str%rmw))             DEALLOCATE(str%rmw)
03521
03522 IF (ALLOCATED(str%gusts))          DEALLOCATE(str%gusts)
03523 IF (ALLOCATED(str%eye))             DEALLOCATE(str%eye)
03524 IF (ALLOCATED(str%subregion))      DEALLOCATE(str%subregion)
03525 IF (ALLOCATED(str%maxseas))         DEALLOCATE(str%maxseas)
03526 IF (ALLOCATED(str%initials))       DEALLOCATE(str%initials)
03527
03528
03529
03530
03531
03532
03533
03534
03535
03536
03537
03538
03539
03540
03541
03542

```

```
03543    IF (.NOT. ALLOCATED(str%trVx))          DEALLOCATE(str%trVx)
03544    IF (.NOT. ALLOCATED(str%trVy))          DEALLOCATE(str%trVy)
03545
03546    IF (ALLOCATED(str%idir))                 DEALLOCATE(str%idir)
03547    IF (ALLOCATED(str%dir))                  DEALLOCATE(str%dir)
03548    IF (ALLOCATED(str%iStormSpeed))         DEALLOCATE(str%iStormSpeed)
03549    IF (ALLOCATED(str%stormSpeed))          DEALLOCATE(str%stormSpeed)
03550    IF (ALLOCATED(str%stormName))           DEALLOCATE(str%stormName)
03551
03552    IF (ALLOCATED(str%numCycle))            DEALLOCATE(str%numCycle)
03553    IF (ALLOCATED(str%isotachsPerCycle))   DEALLOCATE(str%isotachsPerCycle)
03554    IF (ALLOCATED(str%quadFlag))           DEALLOCATE(str%quadFlag)
03555    IF (ALLOCATED(str%rMaxW))              DEALLOCATE(str%rMaxW)
03556    IF (ALLOCATED(str%hollB))              DEALLOCATE(str%hollB)
03557    IF (ALLOCATED(str%hollBs))             DEALLOCATE(str%hollBs)
03558    IF (ALLOCATED(str%vMaxesBL))           DEALLOCATE(str%vMaxesBL)
03559
03560 END SUBROUTINE deallocasymvortstruct
03561
03562 !=====
03563
03564 END MODULE parwind
```

20.35 /home/takis/CSDL/parwinds-doc/src/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::comparedoublerials](#) (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::fixnearwholedoubleral](#) (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

20.35.1 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](#).

20.36 sizes.F90

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

```

00001 !-----  

00002 !----- M O D U L E   S I Z E S  

00003 !-----  

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 :: nbytes = 4
00055 #else
00056  INTEGER, PARAMETER :: sz = hp
00057  INTEGER, PARAMETER :: nbytes = 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 = -9999999.0_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
00072  CONTAINS
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  END IF
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 !-----+
00147 !-----+
00148 INTEGER FUNCTION comparesinglereals(rVal1, rVal2, eps) RESULT(myValOut)
00149
00150    IMPLICIT NONE
00151
00152    ! Global variables
00153    REAL(sp), INTENT(IN) :: rval1, rval2
00154    REAL(sp), OPTIONAL, INTENT(IN) :: eps
00155
00156    ! Local variables
00157    REAL(sp) :: epssys, epsusr, value
00158
00159    epssys = 2.0_sp * epsilon(rval1)
00160
00161    IF (PRESENT(eps)) THEN
00162        epsusr = abs(eps)
00163    ELSE
00164        epsusr = epssys
00165    ENDIF
00166
00167    IF ((abs(rval1) < 1.0_sp) .OR. (abs(rval2) < 1.0_sp)) THEN
00168        value = rval1 - rval2
00169    ELSE
00170        value = (rval1 - rval2) / max(rval1, rval2)
00171        IF (abs(value) < 1.0_sp) value = rval1 - rval2
00172    END IF
00173
00174    IF (abs(value) < epsusr) THEN
00175        myvalout = 0
00176    ELSE IF (rval1 < rval2) THEN
00177        myvalout = -1
00178    ELSE
00179        myvalout = 1
00180    END IF
00181
00182    RETURN
00183
00184 END FUNCTION comparesinglereals
00185
00186 !=====
00187
00188 !-----+
00189 ! FUNCTION FIX_NEAR_WHOLE_DOUBLE_REAL
00190 !-----+
00191 !-----+
00192 !-----+
00193 !-----+
00194 !-----+
00195 !-----+
00196 !-----+
00197 !-----+
00198 !-----+
00199 !-----+
00200 !-----+
00201 !-----+
00202 !-----+
00203 !-----+
00204 !-----+
00205 !-----+
00206 !-----+
00207 !=====

00208
00209 !-----+
00210 !-----+
00211 !-----+
00212 !-----+
00213 !-----+
00214 !-----+
00215 !-----+
00216 !-----+
00217 !-----+
00218 !-----+
00219 !-----+
00220 !-----+
00221 !-----+
00222 !-----+
00223 !-----+
00224 !-----+
00225 !-----+
00226 !-----+
00227 !-----+
00228 !-----+
00229 !-----+
00230 !-----+
00231 !-----+
00232 !-----+
00233 !-----+
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    epssys = 2.0_hp * epsilon(rval)
00247
00248    IF (PRESENT(eps)) THEN
00249        epsusr = abs(eps)
00250    ELSE
00251        epsusr = epssys
00252    ENDIF
00253
00254

```

```
00255     myvalout = rval
00256     value     = anint(myvalout)
00257     IF (comparereals(myvalout, value, epsusr) == 0) myvalout = value
00258
00259     RETURN
00260
00261 END FUNCTION fixnearwholedoublereal
00262
00263 !=====
00264
00265 !-----  
00266 ! F U N C T I O N   F I X   N E A R   W H O L E   S I N G L E   R E A L
00267 !-----  
00268 !-----  
00269 !-----  
00270 REAL(sp) function fixnearwholesinglereal(rval, eps) result(myvalout)
00271
00272 IMPLICIT NONE
00273
00274 ! Global Variables
00275 REAL(sp), INTENT(IN)          :: rval
00276 REAL(sp), OPTIONAL, INTENT(IN) :: eps
00277
00278 ! Local Variables
00279 REAL(sp)                      :: epssys, epsusr, value
00280
00281
00282 epssys = 2.0_sp * epsilon(rval)
00283
00284 IF (PRESENT(eps)) THEN
00285     epsusr = abs(eps)
00286 ELSE
00287     epsusr = epssys
00288 ENDIF
00289
00290 myvalout = rval
00291 value     = anint(myvalout)
00292 IF (comparereals(myvalout, value, epsusr) == 0) myvalout = value
00293
00294 RETURN
00295
00296 END FUNCTION fixnearwholesinglereal
00297
00298 !=====
00299
00300 END MODULE pahm_sizes
```

20.37 /home/takis/CSDL/parwinds-doc/src/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)

Swaps the contents of i and j (integer).
- 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::arraycopsingle` (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::arrayqualsingle` (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).
- subroutine `sortutils::swapsingle` (a, b, mask)

Swaps the contents of a and b (single precision).
- subroutine `sortutils::swapdouble` (a, b, mask)

Swaps the contents of a and b (double precision).
- subroutine `sortutils::swapintvec` (a, b, mask)

Swaps the contents of a and b (integer).
- subroutine `sortutils::swapsinglevec` (a, b, mask)

Swaps the contents of a and b (single precision).
- subroutine `sortutils::swapdoublevec` (a, b, mask)

Swaps the contents of a and b (double precision).
- 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").

20.37.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [sortutils.F90](#).

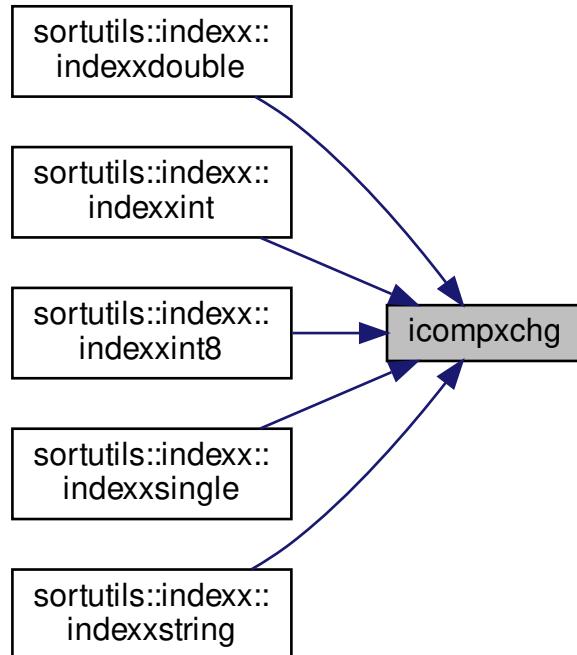
20.37.2 Function/Subroutine Documentation

20.37.2.1 `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::indexxit\(\)](#), [sortutils::indexx::indexxit8\(\)](#), [sortutils::indexx::indexxsingle\(\)](#) and [sortutils::indexx::indexxstring\(\)](#).

Here is the caller graph for this function:



20.38 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 indexxit  
00026   MODULE PROCEDURE indexxit8  
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 arraycopsingle  
00041   MODULE PROCEDURE arraycopydouble  
00042 END INTERFACE arraycopy  
00043  
00044 INTERFACE arrayequal  
00045   MODULE PROCEDURE arrayequalint  
00046   MODULE PROCEDURE arrayqualsingle  
00047   MODULE PROCEDURE arrayqualdouble  
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 ! S U B R O U T I N E   I N D E X X   I N T  
00066 !-----  
00084 !-----  
00085 SUBROUTINE indexxit(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("Indexxit")
```

```

00104
00105 IF (PRESENT(status)) status = 0
00106
00107 narr = SIZE(arrld, 1)
00108 nidx = SIZE(idxld, 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 idxld = 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 = l + 1, r
00133       tmpidx = idxld(j)
00134       a = arrld(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 icompchg(idxld(l), idxld(r))
00156   CALL icompchg(idxld(l + 1), idxld(r))
00157   CALL icompchg(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
00212      CONTAINS
00213
00214      SUBROUTINE icompchg(i, j)
00215
00216      IMPLICIT NONE
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 !-----  
S U B R O U T I N E   I N D E X X   I N T   8  
!-----  
00237
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
00259      CALL setmessagesource("IndexxInt8")
00260
00261      IF (PRESENT(status)) status = 0
00262
00263      narr = SIZE(arr1D, 1)
00264      nidx = SIZE(idx1D, 1)
00265
00266      IF (narr /= nidx) THEN

```

```

00283 WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00284 WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00285 WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00286     trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00287
00288 CALL allmessage(error, scratchmessage)
00289 CALL unsetmessagesource()
00290
00291 IF (PRESENT(status)) status = 1
00292
00293 RETURN
00294 END IF
00295
00296 idx1d = arth(1, 1, narr)
00297
00298 ist = 0
00299 l = 1
00300 r = narr
00301
00302 DO
00303   IF (r - l < nn) THEN
00304     DO j = l + 1, r
00305       tmpidx = idx1d(j)
00306       a = arr1d(tmpidx)
00307       DO i = j - 1, l, -1
00308         IF (arr1d(idx1d(i)) <= a) EXIT
00309         idx1d(i + 1) = idx1d(i)
00310       END DO
00311       idx1d(i + 1) = tmpidx
00312     END DO
00313
00314   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 icompxchg(idx1d(l), idx1d(r))
00328   CALL icompxchg(idx1d(l + 1), idx1d(r))
00329   CALL icompxchg(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      RETURN
00365
00366      END IF
00367
00368      IF (r - i + 1 >= j - 1) THEN
00369          stack(ist) = r
00370          stack(ist - 1) = i
00371          r = j - 1
00372
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
00384 CONTAINS
00385
00386 SUBROUTINE icompxchg(i, j)
00387
00388     IMPLICIT NONE
00389
00390     ! Global variables
00391     INTEGER, INTENT(INOUT) :: i, j
00392
00393     ! Local variables
00394     INTEGER :: swp
00395
00396     IF (arr1D(j) < arr1D(i)) THEN
00397         swp = i
00398         i = j
00399         j = swp
00400     END IF
00401
00402 END SUBROUTINE icompxchg
00403
00404 END SUBROUTINE indexxint8
00405
00406 !=====
00407 !-----+
00408 !-----+ S U B R O U T I N E   I N D E X X   S T R I N G
00409 !-----+
00410 !-----+
00429 !
00430 SUBROUTINE indexxstring(arr1D, idx1D, status, caseSens)
00431
00432     IMPLICIT NONE
00433
00434     ! Global variables
00435     CHARACTER(LEN=*) , DIMENSION(:), INTENT(IN) :: arr1D
00436     LOGICAL, OPTIONAL, INTENT(IN)                  :: caseSens
00437     INTEGER, DIMENSION(:), INTENT(OUT)             :: idx1D
00438     INTEGER, OPTIONAL, INTENT(OUT)                 :: status
00439
00440     ! Local variables
00441     INTEGER, PARAMETER                           :: NN = 15, nstack = 50
00442     CHARACTER(LEN=LEN(arr1D(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(arr1D, 1)
00458     nidx = SIZE(idx1D, 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 arr1D and idx1D 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      idx1d = 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 = idx1d(j)
00484                  a = arr1d(tmpidx)
00485                  DO i = j - 1, l, -1
00486                      IF (stringlexcomp(arr1d(idx1d(i)), a, sflag) <= 0) EXIT
00487                      idx1d(i + 1) = idx1d(i)
00488                  END DO
00489                  idx1d(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 icompxchg(idx1d(l), idx1d(r))
00506          CALL icompxchg(idx1d(l + 1), idx1d(r))
00507          CALL icompxchg(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
00545      END IF
00546
00547      IF (r - i + 1 >= j - 1) THEN
00548          stack(ist) = r
00549          stack(ist - 1) = i
00550          r = j - 1
00551      ELSE
00552          stack(ist) = j - 1
00553          stack(ist - 1) = l
00554          l = i
00555      END IF
00556      END IF
00557  END DO
00558
00559  CALL unsetmessagesource()
00560
00561
00562  CONTAINS
00563
00564  SUBROUTINE icompxchg(i, j)
00565
00566      IMPLICIT NONE
00567
00568      ! Global variables
00569      INTEGER, INTENT(INOUT) :: i, j
00570
00571      ! Local variables
00572      INTEGER :: swp
00573
00574      IF (stringlexcomp(arr1d(j), arr1d(i), sflag) < 0) THEN
00575          swp = i
00576          i = j
00577          j = swp
00578      END IF
00579
00580  END SUBROUTINE icompxchg
00581
00582  END SUBROUTINE indexxstring
00583
00584 !=====
00585
00586 !-----
00587 !----- S U B R O U T I N E   I N D E X X   S I N G L E
00588 !-----
00589 !-----
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      RETURN
00643  END IF
00645
00646  idxld = 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 = idxld(j)
00656        a = arrld(tmpidx)
00657        DO i = j - 1, l, -1
00658          IF (arrld(idxld(i)) <= a) EXIT
00659          idxld(i + 1) = idxld(i)
00660        END DO
00661        idxld(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(idxld(k), idxld(l + 1))
00677   CALL icompxchg(idxld(l), idxld(r))
00678   CALL icompxchg(idxld(l + 1), idxld(r))
00679   CALL icompxchg(idxld(l), idxld(l + 1))
00680
00681   i = l + 1
00682   j = r
00683   tmpidx = idxld(l + 1)
00684   a = arrld(tmpidx)
00685
00686   DO
00687     DO
00688       i = i + 1
00689       IF (arrld(idxld(i)) > a) EXIT
00690     END DO
00691
00692     DO
00693       j = j - 1
00694       IF (arrld(idxld(j)) < a) EXIT
00695     END DO
00696
00697     IF (j < i) EXIT
00698     CALL swap(idxld(i), idxld(j))
00699   END DO
00700
00701   idxld(l + 1) = idxld(j)
00702   idxld(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
00734 CONTAINS
00735
00736 SUBROUTINE icompxchg(i, j)
00737
00738 IMPLICIT NONE
00739
00740 ! Global variables
00741 INTEGER, INTENT(INOUT) :: i, j
00742
00743 ! Local variables
00744 INTEGER :: swp
00745
00746 IF (arr1d(j) < arr1d(i)) THEN
00747     swp = i
00748     i = j
00749     j = swp
00750 END IF
00751
00752 END SUBROUTINE icompxchg
00753
00754 END SUBROUTINE indexxsingle
00755
00756 !=====
00757
00758 !-----
00759 ! S U B R O U T I N E   I N D E X X   D O U B L E
00760 !-----
00761 !-----
00762 !-----
00763
00764 SUBROUTINE indexxdouble(arr1D, idx1D, status)
00765
00766 IMPLICIT NONE
00767
00768 ! Global variables
00769 REAL(HP), DIMENSION(:), INTENT(IN) :: arr1D
00770 INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00771 INTEGER, OPTIONAL, INTENT(OUT) :: status
00772
00773 ! Local variables
00774 INTEGER, PARAMETER :: NN = 15, nstack = 50
00775 REAL(HP) :: a
00776 INTEGER :: nARR, nIDX, tmpIDX
00777 INTEGER :: k, i, j, l, r
00778 INTEGER :: ist, stack(NSTACK)
00779 CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00780
00781
00782
00783
00784
00785
00786
00787
00788
00789
00790
00791
00792
00793
00794
00795
00796
00797
00798
00799
00800
00801
00802
00803
00804
00805
00806
00807
00808
00809
00810
00811
00812
00813
00814
00815
00816
00817
00818
00819
00820
00821
00822
00823
00824
00825
00826
00827
00828
00829
00830
00831
00832
00833
00834
00835
00836
00837
00838
00839
00840
00841
00842
00843
00844
00845
00846
00847
00848
00849
00850
00851
00852
00853
00854
00855
00856
00857
00858
00859
00860
00861
00862
00863
00864
00865
00866
00867
00868
00869
00870
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888
00889
00890
00891
00892
00893
00894
00895
00896
00897
00898
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917
00918
00919
00920
00921
00922
00923
00924
00925
00926
00927
00928
00929
00930
00931
00932
00933
00934
00935
00936
00937
00938
00939
00940
00941
00942
00943
00944
00945
00946
00947
00948
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968
00969
00970
00971
00972
00973
00974
00975
00976
00977
00978
00979
00980
00981
00982
00983
00984
00985
00986
00987
00988
00989
00990
00991
00992
00993
00994
00995
00996
00997
00998
00999
01000
01001
01002
01003
01004
01005
01006
01007
01008
01009
01010
01011
01012
01013
01014
01015
01016
01017
01018
01019
01020
01021
01022
01023
01024
01025
01026
01027
01028
01029
01030
01031
01032
01033
01034
01035
01036
01037
01038
01039
01040
01041
01042
01043
01044
01045
01046
01047
01048
01049
01050
01051
01052
01053
01054
01055
01056
01057
01058
01059
01060
01061
01062
01063
01064
01065
01066
01067
01068
01069
01070
01071
01072
01073
01074
01075
01076
01077
01078
01079
01080
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099
01100
01101
01102
01103
01104
01105
01106
01107
01108
01109
01110
01111
01112
01113
01114
01115
01116
01117
01118
01119
01120
01121
01122
01123
01124
01125
01126
01127
01128
01129
01130
01131
01132
01133
01134
01135
01136
01137
01138
01139
01140
01141
01142
01143
01144
01145
01146
01147
01148
01149
01150
01151
01152
01153
01154
01155
01156
01157
01158
01159
01160
01161
01162
01163
01164
01165
01166
01167
01168
01169
01170
01171
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186
01187
01188
01189
01190
01191
01192
01193
01194
01195
01196
01197
01198
01199
01200
01201
01202
01203
01204
01205
01206
01207
01208
01209
01210
01211
01212
01213
01214
01215
01216
01217
01218
01219
01220
01221
01222
01223
01224
01225
01226
01227
01228
01229
01230
01231
01232
01233
01234
01235
01236
01237
01238
01239
01240
01241
01242
01243
01244
01245
01246
01247
01248
01249
01250
01251
01252
01253
01254
01255
01256
01257
01258
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275
01276
01277
01278
01279
01280
01281
01282
01283
01284
01285
01286
01287
01288
01289
01290
01291
01292
01293
01294
01295
01296
01297
01298
01299
01300
01301
01302
01303
01304
01305
01306
01307
01308
01309
01310
01311
01312
01313
01314
01315
01316
01317
01318
01319
01320
01321
01322
01323
01324
01325
01326
01327
01328
01329
01330
01331
01332
01333
01334
01335
01336
01337
01338
01339
01340
01341
01342
01343
01344
01345
01346
01347
01348
01349
01350
01351
01352
01353
01354
01355
01356
01357
01358
01359
01360
01361
01362
01363
01364
01365
01366
01367
01368
01369
01370
01371
01372
01373
01374
01375
01376
01377
01378
01379
01380
01381
01382
01383
01384
01385
01386
01387
01388
01389
01390
01391
01392
01393
01394
01395
01396
01397
01398
01399
01400
01401
01402
01403
01404
01405
01406
01407
01408
01409
01410
01411
01412
01413
01414
01415
01416
01417
01418
01419
01420
01421
01422
01423
01424
01425
01426
01427
01428
01429
01430
01431
01432
01433
01434
01435
01436
01437
01438
01439
01440
01441
01442
01443
01444
01445
01446
01447
01448
01449
01450
01451
01452
01453
01454
01455
01456
01457
01458
01459
01460
01461
01462
01463
01464
01465
01466
01467
01468
01469
01470
01471
01472
01473
01474
01475
01476
01477
01478
01479
01480
01481
01482
01483
01484
01485
01486
01487
01488
01489
01490
01491
01492
01493
01494
01495
01496
01497
01498
01499
01500
01501
01502
01503
01504
01505
01506
01507
01508
01509
01510
01511
01512
01513
01514
01515
01516
01517
01518
01519
01520
01521
01522
01523
01524
01525
01526
01527
01528
01529
01530
01531
01532
01533
01534
01535
01536
01537
01538
01539
01540
01541
01542
01543
01544
01545
01546
01547
01548
01549
01550
01551
01552
01553
01554
01555
01556
01557
01558
01559
01560
01561
01562
01563
01564
01565
01566
01567
01568
01569
01570
01571
01572
01573
01574
01575
01576
01577
01578
01579
01580
01581
01582
01583
01584
01585
01586
01587
01588
01589
01590
01591
01592
01593
01594
01595
01596
01597
01598
01599
01600
01601
01602
01603
01604
01605
01606
01607
01608
01609
01610
01611
01612
01613
01614
01615
01616
01617
01618
01619
01620
01621
01622
01623
01624
01625
01626
01627
01628
01629
01630
01631
01632
01633
01634
01635
01636
01637
01638
01639
01640
01641
01642
01643
01644
01645
01646
01647
01648
01649
01650
01651
01652
01653
01654
01655
01656
01657
01658
01659
01660
01661
01662
01663
01664
01665
01666
01667
01668
01669
01670
01671
01672
01673
01674
01675
01676
01677
01678
01679
01680
01681
01682
01683
01684
01685
01686
01687
01688
01689
01690
01691
01692
01693
01694
01695
01696
01697
01698
01699
01700
01701
01702
01703
01704
01705
01706
01707
01708
01709
01710
01711
01712
01713
01714
01715
01716
01717
01718
01719
01720
01721
01722
01723
01724
01725
01726
01727
01728
01729
01730
01731
01732
01733
01734
01735
01736
01737
01738
01739
01740
01741
01742
01743
01744
01745
01746
01747
01748
01749
01750
01751
01752
01753
01754
01755
01756
01757
01758
01759
01760
01761
01762
01763
01764
01765
01766
01767
01768
01769
01770
01771
01772
01773
01774
01775
01776
01777
01778
01779
01780
01781
01782
01783
01784
01785
01786
01787
01788
01789
01790
01791
01792
01793
01794
01795
01796
01797
01798
01799
01800
01801
01802
01803
01804
01805
01806
01807
01808
01809
01810
01811
01812
01813
01814
01815
01816
01817
01818
01819
01820
01821
01822
01823
01824
01825
01826
01827
01828
01829
01830
01831
01832
01833
01834
01835
01836
01837
01838
01839
01840
01841
01842
01843
01844
01845
01846
01847
01848
01849
01850
01851
01852
01853
01854
01855
01856
01857
01858
01859
01860
01861
01862
01863
01864
01865
01866
01867
01868
01869
01870
01871
01872
01873
01874
01875
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887
01888
01889
01890
01891
01892
01893
01894
01895
01896
01897
01898
01899
01900
01901
01902
01903
01904
01905
01906
01907
01908
01909
01910
01911
01912
01913
01914
01915
01916
01917
01918
01919
01920
01921
01922
01923
01924
01925
01926
01927
01928
01929
01930
01931
01932
01933
01934
01935
01936
01937
01938
01939
01940
01941
01942
01943
01944
01945
01946
01947
01948
01949
01950
01951
01952
01953
01954
01955
01956
01957
01958
01959
01960
01961
01962
01963
01964
01965
01966
01967
01968
01969
01970
01971
01972
01973
01974
01975
01976
01977
01978
01979
01980
01981
01982
01983
01984
01985
01986
01987
01988
01989
01990
01991
01992
01993
01994
01995
01996
01997
01998
01999
02000
02001
02002
02003
02004
02005
02006
02007
02008
02009
02010
02011
02012
02013
02014
02015
02016
02017
02018
02019
02020
02021
02022
02023
02024
02025
02026
02027
02028
02029
02030
02031
02032
02033
02034
02035
02036
02037
02038
02039
02040
02041
02042
02043
02044
02045
02046
02047
02048
02049
02050
02051
02052
02053
02054
02055
02056
02057
02058
02059
02060
02061
02062
02063
02064
02065
02066
02067
02068
02069
02070
02071
02072
02073
02074
02075
02076
02077
02078
02079
02080
02081
02082
02083
02084
02085
02086
02087
02088
02089
02090
02091
02092
02093
02094
02095
02096
02097
02098
02099
02100
02101
02102
02103
02104
02105
02106
02107
02108
02109
02110
02111
02112
02113
02114
02115
02116
02117
02118
02119
02120
02121
02122
02123
02124
02125
02126
02127
02128
02129
02130
02131
02132
02133
02134
02135
02136
02137
02138
02139
02140
02141
02142
02143
02144
02145
02146
02147
02148
02149
02150
02151
02152
02153
02154
02155
02156
02157
02158
02159
02160
02161
02162
02163
02164
02165
02166
02167
02168
02169
02170
02171
02172
02173
02174
02175
02176
02177
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206
02207
02208
02209
02210
02211
02212
02213
02214
02215
02216
02217
02218
02219
02220
02221
02222
02223
02224
02225
02226
02227
02228
02229
02230
02231
02232
02233
02234
02235
02236
02237
02238
02239
02240
02241
02242
02243
02244
02245
02246
02247
02248
02249
02250
02251
02252
02253
02254
02255
02256
02257
02258
02259
02260
02261
02262
02263
02264
02265
02266
02267
02268
02269
02270
02271
02272
02273
02274
02275
02276
02277
02278
02279
02280
02281
02282
02283
02284
02285
02286
02287
02288
02289
02290
02291
02292
02293
02294
02295
02296
02297
02298
02299
02300
02301
02302
02303
02304
02305
02306
02307
02308
02309
02310
02311
02312
02313
02314
02315
02316
02317
02318
02319
02320
02321
02322
02323
02324
02325
02326
02327
02328
02329
02330
02331
02332
02333
02334
02335
02336
02337
02338
02339
02340
02341
02342
02343
02344
02345
02346
02347
02348
02349
02350
02351
02352
02353
02354
02355
02356
02357
02358
02359
02360
02361
02362
02363
02364
02365
02366
02367
02368
02369
02370
02371
02372
02373
02374
02375
02376
02377
02378
02379
02380
02381
02382
02383
02384
02385
02386
02387
02388
02389
02390
02391
02392
02393
02394
02395
02396
02397
02398
02399
02400
02401
02402
02403
02404
02405
02406
02407
02408
02409
02410
02411
02412
02413
02414
02415
02416
02417
02418
02419
02420
02421
02422
02423
02424
02425
02426
02427
02428
02429
02430
02431
02432
02433
02434
02435
02436
02437
02438
02439
02440
02441
02442
02443
02444
02445
02446
02447
02448
02449
02450
02451
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
02470
02471
02472
02473
02474
02475
02476
02477
02478
02479
02480
02481
02482
02483
02484
02485
02486
02487
02488
02489
02490
02491
02492
02493
02494
02495
02496
02497
02498
02499
02500
02501
02502
02503
02504
02505
02506
02507
02508
02509
02510
02511
02512
02513
02514
02515
02516
02517
02518
02519
02520
02521
02522
02523
02524
02525
02526
02527
02528
02529
02530
02531
02532
02533
02534
02535
02536
02537
02538
02539
02540
02541
02542
02543
02544
02545
02546
02547
02548
02549
02550
02551
02552
02553
02554
02555
02556
02557
0
```

```

00821      l    = 1
00822      r    = narr
00823
00824      DO
00825          IF (r - l < nn) THEN
00826              DO j = l + 1, r
00827                  tmpidx = idxld(j)
00828                  a = arrld(tmpidx)
00829                  DO i = j - 1, l, -1
00830                      IF (arrld(idxld(i)) <= a) EXIT
00831                      idxld(i + 1) = idxld(i)
00832                  END DO
00833                  idxld(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(idxld(k), idxld(l + 1))
00849          CALL icompxchg(idxld(l), idxld(r))
00850          CALL icompxchg(idxld(l + 1), idxld(r))
00851          CALL icompxchg(idxld(l), idxld(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
00906     CONTAINS
00907
00908     SUBROUTINE icompxchg(i, j)
00909
00910         IMPLICIT NONE
00911
00912         ! Global variables
00913         INTEGER, INTENT(INOUT) :: i, j
00914
00915         ! Local variables
00916         INTEGER :: swp
00917
00918         IF (arr1d(j) < arr1d(i)) THEN
00919             swp = i
00920             i   = j
00921             j   = swp
00922         END IF
00923
00924     END SUBROUTINE icompxchg
00925
00926     END SUBROUTINE indexxdouble
00927
00928 !=====
00929 !-----
00930 !-----S U B R O U T I N E   Q U I C K   S O R T
00931 !-----
00932 !-----
00933 !-----
00934 !-----SUBROUTINE quicksort(arr1D, status)
00935
00936     IMPLICIT NONE
00937
00938     ! Global variables
00939     REAL(SZ), DIMENSION(:, ), INTENT(INOUT) :: arr1D
00940     INTEGER, OPTIONAL, INTENT(OUT)       :: status
00941
00942     ! Local variables
00943     INTEGER, PARAMETER                  :: NN = 15, nstack = 50
00944     REAL(SZ)                           :: a
00945     INTEGER                           :: nARR
00946     INTEGER                           :: k, i, j, l, r
00947     INTEGER                           :: ist, stack(NSTACK)
00948     CHARACTER(LEN=64)                 :: tmpStr1
00949
00950
00951
00952
00953
00954
00955
00956
00957
00958
00959
00960
00961
00962
00963
00964
00965
00966
00967
00968     CALL setmessagesource("QuickSort")
00969
00970     IF (PRESENT(status)) status = 0
00971
00972     narr = size(arr1D, 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 = arr1d(j)
00983                 DO i = j - 1, l, -1
00984                     IF (arr1d(i) <= a) EXIT
00985                     arr1d(i + 1) = arr1d(i)
00986                 END DO
00987                 arr1d(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
01004      ELSE
01005          k = (l + r) / 2
01006
01007          CALL swap(arrld(k), arrld(l + 1))
01008          CALL swap(arrld(l), arrld(r), arrld(l) > arrld(r))
01009          CALL swap(arrld(l + 1), arrld(r), arrld(l + 1) > arrld(r))
01010          CALL swap(arrld(l), arrld(l + 1), arrld(l) > arrld(l + 1))
01011
01012          ! Initialize pointers for partitioning
01013          i = l + 1
01014          j = r
01015          a = arrld(l + 1) ! Partitioning element.
01016
01017          DO ! Here is the meat.
01018              ! Scan up to find element >= a
01019              DO
01020                  i = i + 1
01021                  IF (arrld(i) > a) EXIT
01022              END DO
01023
01024              ! Scan down to find element <= a
01025              DO
01026                  j = j - 1
01027                  IF (arrld(j) < a) EXIT
01028              END DO
01029
01030              ! Pointers crossed. Exit with partitioning complete.
01031              IF (j < i) EXIT
01032
01033              CALL swap(arrld(i), arrld(j)) !Exchange elements.
01034
01035              ! Insert partitioning element
01036              arrld(l + 1) = arrld(j)
01037              arrld(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
01053              END IF
01054
01055              IF (r - i + 1 >= j - 1) THEN
01056                  stack(ist) = r
01057                  stack(ist - 1) = i
01058                  r = j - 1
01059              ELSE
01060                  stack(ist) = j - 1
01061                  stack(ist - 1) = l
01062                  l = i
01063              END IF
01064          END IF
01065      END DO
01066
01067      CALL unsetmessagesource()
01068
01069  END SUBROUTINE quicksort
01070
01071  =====
01072
01073  -----
01074  ! S U B R O U T I N E   S O R T  2
01075  -----
01076
01077  -----
01078  SUBROUTINE sort2(arr1D, slv1D, status)
01079
01080      IMPLICIT NONE
01081
01082
01083
01084
01085
01086
01087
01088
01089
01090
01091
01092
01093
01094
01095
01096
01097
01098
01099

```

```

01099 ! Global variables
01100 REAL(SZ), DIMENSION(:), INTENT(INOUT) :: arr1D, slv1D
01101 INTEGER, OPTIONAL, INTENT(OUT)       :: status
01102
01103 ! Local variables
01104 INTEGER                      :: nARR, nSLV
01105 INTEGER, DIMENSION(SIZE(arr1D)) :: idx1D
01106 CHARACTER(LEN=64)            :: tmpStr1, tmpStr2
01107
01108
01109 CALL setmessagesource("Sort2")
01110
01111 narr = SIZE(arr1D, 1)
01112 nslv = SIZE(slv1D, 1)
01113
01114 IF (narr /= nslv) THEN
01115   WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
01116   WRITE(tmpstr2, '(a, i0)') 'nSLV = ', nslv
01117   WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and slv1D is not the same: ' // &
01118   trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
01119
01120 CALL logmessage(error, scratchmessage)
01121 CALL unsetmessagesource()
01122
01123 IF (PRESENT(status)) status = 1
01124
01125 RETURN
01126 END IF
01127
01128 ! Make the index array
01129 CALL indexx(arr1D, idx1D, status)
01130
01131 ! Sort the array
01132 arr1D = arr1D(idx1D)
01133
01134 ! Rearrange slave
01135 slv1D = slv1D(idx1D)
01136
01137 CALL unsetmessagesource()
01138
01139 END SUBROUTINE sort2
01140
01141 !=====
01142
01143
01144 !-----
01145 ! S U B R O U T I N E   A R R A Y   C O P Y   I N T
01146 !-----
01147 !-----
01148
01149 SUBROUTINE arraycopyint(src, dest, nCP, nNCP)
01150
01151 IMPLICIT NONE
01152
01153 ! Global variables
01154 INTEGER, DIMENSION(:, ), INTENT(IN)  :: src
01155 INTEGER, DIMENSION(:, ), INTENT(OUT) :: dest
01156 INTEGER, INTENT(OUT)                 :: nCP, nNCP
01157
01158 ncp = min(SIZE(src), SIZE(dest))
01159 nncp = SIZE(src) - ncp
01160 dest(1:ncp) = src(1:ncp)
01161
01162
01163 END SUBROUTINE arraycopyint
01164
01165 !=====
01166
01167 !-----
01168 ! S U B R O U T I N E   A R R A Y   C O P Y   S I N G L E
01169 !-----
01170 !-----
01171
01172 SUBROUTINE arraycopiesingle(src, dest, nCP, nNCP)
01173
01174 IMPLICIT NONE
01175
01176 ! Global variables
01177 REAL(SP), DIMENSION(:, ), INTENT(IN)  :: src
01178 REAL(SP), DIMENSION(:, ), INTENT(OUT) :: dest
01179 INTEGER, INTENT(OUT)                 :: nCP, nNCP
01180
01181 ncp = min(SIZE(src), SIZE(dest))
01182 nncp = SIZE(src) - ncp
01183
01184 !-----
01185 !-----
01186
01187 ! S U B R O U T I N E   A R R A Y   C O P Y   S I N G L E
01188 !-----
01189 !-----
01190
01191 SUBROUTINE arraycopiesingle(src, dest, nCP, nNCP)
01192
01193 IMPLICIT NONE
01194
01195 ! Global variables
01196 REAL(SP), DIMENSION(:, ), INTENT(IN)  :: src
01197 REAL(SP), DIMENSION(:, ), INTENT(OUT) :: dest
01198 INTEGER, INTENT(OUT)                 :: nCP, nNCP
01199
01200 ncp = min(SIZE(src), SIZE(dest))
01201 nncp = SIZE(src) - ncp
01202
01203
01204 !-----
01205 !-----
01206 !-----
```

```

01216      dest(1:ncp) = src(1:ncp)
01217
01218  END SUBROUTINE arraycopsingle
01219
01220 !=====
01221
01222 !-----+
01223 ! S U B R O U T I N E   A R R A Y   C O P Y   D O U B L E
01224 !
01225 !-----
01226
01227 SUBROUTINE arraycopydouble(src, dest, nCP, nNCP)
01228
01229     IMPLICIT NONE
01230
01231     ! Global variables
01232     REAL(HP), DIMENSION(:), INTENT(IN) :: src
01233     REAL(HP), DIMENSION(:), INTENT(OUT) :: dest
01234     INTEGER, INTENT(OUT) :: nCP, nNCP
01235
01236     ncp = min(SIZE(src), SIZE(dest))
01237     nnccp = SIZE(src) - ncp
01238     dest(1:ncp) = src(1:ncp)
01239
01240  END SUBROUTINE arraycopydouble
01241
01242 !=====
01243
01244 !-----+
01245 ! F U N C T I O N   A R R A Y   E Q U A L   I N T
01246 !
01247 !-----
01248
01249 LOGICAL FUNCTION arrayequalint(arr1, arr2) RESULT(myValOut)
01250
01251     IMPLICIT NONE
01252
01253     ! Global variables
01254     INTEGER, DIMENSION(:), INTENT(IN) :: arr1, arr2
01255
01256     IF (SIZE(arr1) /= SIZE(arr2)) THEN
01257         myvalout = .false.
01258
01259         RETURN
01260     END IF
01261
01262     myvalout = .true.
01263     IF (any(arr1 - arr2 /= 0)) myvalout = .false.
01264
01265     RETURN
01266
01267  END FUNCTION arrayequalint
01268
01269 !=====
01270
01271 !-----+
01272 ! F U N C T I O N   A R R A Y   E Q U A L   S I N G L E
01273 !
01274 !-----
01275
01276 LOGICAL FUNCTION arrayqualsingle(arr1, arr2) RESULT(myValOut)
01277
01278     IMPLICIT NONE
01279
01280     ! Global variables
01281     REAL(sp), DIMENSION(:), INTENT(IN) :: arr1, arr2
01282
01283     ! Local variables
01284     INTEGER :: i
01285
01286     IF (SIZE(arr1) /= SIZE(arr2)) THEN
01287         myvalout = .false.
01288
01289         RETURN
01290     END IF
01291
01292     myvalout = .true.
01293
01294     DO i = 1, SIZE(arr1, 1)
01295         IF (comparereals(arr1(i), arr2(i), 0.0000001_sp) /= 0) THEN
01296             myvalout = .false.
01297
01298     END DO
01299
01300  END FUNCTION arrayqualsingle
01301
01302 !=====
01303
01304 !-----+
01305 ! F U N C T I O N   A R R A Y   E Q U A L   S I N G L E
01306 !
01307 !-----
01308
01309 LOGICAL FUNCTION comparereals(x, y, tolerance)
01310
01311     IMPLICIT NONE
01312
01313     ! Global variables
01314     REAL(sp), INTENT(IN) :: x, y
01315
01316     ! Local variables
01317     REAL(sp) :: diff, abs_x, abs_y
01318
01319     diff = abs(x - y)
01320     abs_x = abs(x)
01321     abs_y = abs(y)
01322
01323     IF (abs_x == 0.0_sp) THEN
01324         IF (abs_y == 0.0_sp) THEN
01325             comparereals = .true.
01326         ELSE
01327             comparereals = .false.
01328         END IF
01329     ELSE
01330         IF (abs_y == 0.0_sp) THEN
01331             comparereals = .true.
01332         ELSE
01333             comparereals = (diff / abs_y) <= tolerance
01334         END IF
01335     END IF
01336
01337  END FUNCTION comparereals
01338
01339 !-----+
01340 !-----+
01341 !-----+
01342 !-----+
01343 !-----+
01344 !-----+
01345 !-----+
01346 !-----+
01347 !-----+
01348 !-----+

```

```
01349      EXIT
01350      END IF
01351      END DO
01352
01353      RETURN
01354
01355  END FUNCTION arrayqualsingle
01356
01357 !=====
01358 !-----+
01359 !-----+
01360 !-----+
01361 !-----+
01362 !-----+
01363 LOGICAL FUNCTION arrayequaldouble(arr1, arr2) RESULT(myValOut)
01364
01365      IMPLICIT NONE
01366
01367      ! Global variables
01368      REAL(hp), DIMENSION(:), INTENT(IN) :: arr1, arr2
01369
01370      ! Local variables
01371      INTEGER :: i
01372
01373      IF (SIZE(arr1) /= SIZE(arr2)) THEN
01374          myvalout = .false.
01375
01376          RETURN
01377      END IF
01378
01379      myvalout = .true.
01380
01381      DO i = 1, SIZE(arr1, 1)
01382          IF (comparereals(arr1(i), arr2(i), 0.000000000001_hp) /= 0) THEN
01383              myvalout = .false.
01384
01385          EXIT
01386      END IF
01387      END DO
01388
01389      RETURN
01390
01391  END FUNCTION arrayequaldouble
01392
01393 !=====
01394 !-----+
01395 !-----+
01396 !-----+
01397 !-----+
01398 INTEGER FUNCTION stringlexcomp(str1, str2, msensitive) RESULT(myValOut)
01399
01400      USE utilities, ONLY : touppercase
01401
01402      IMPLICIT NONE
01403
01404      ! Global variables
01405      CHARACTER(LEN=*), INTENT(IN) :: str1, str2
01406      LOGICAL, OPTIONAL, INTENT(IN) :: msensitive
01407
01408      ! Local variables
01409      LOGICAL :: sflag
01410
01411      sflag = .true.
01412      IF (PRESENT(msensitive)) sflag = msensitive
01413
01414      IF (sflag) THEN
01415          IF (trim(str1) == trim(str2)) THEN
01416              myvalout = 0
01417          ELSE IF (trim(str1) < trim(str2)) THEN
01418              myvalout = -1
01419          ELSE
01420              myvalout = 1
01421          END IF
01422      ELSE
01423          IF (touppercase(trim(str1)) == touppercase(trim(str2))) THEN
01424              myvalout = 0
01425          ELSE IF (touppercase(trim(str1)) < touppercase(trim(str2))) THEN
01426              myvalout = -1
01427          ELSE
01428
```

```

01470      myvalout = 1
01471      END IF
01472  END IF
01473
01474  RETURN
01475
01476 END FUNCTION stringlexcomp
01477
01478 !=====
01479
01480 !-----  

01481 ! S U B R O U T I N E   S W A P   I N T
01482 !-----  

01504 !-----
01505 SUBROUTINE swapint(a, b, mask)
01506
01507  IMPLICIT NONE
01508
01509  ! Global variables
01510  INTEGER, INTENT(INOUT)      :: a, b
01511  LOGICAL, OPTIONAL, INTENT(IN) :: mask
01512
01513  ! Local variables
01514  INTEGER :: dum
01515  LOGICAL :: mFlag
01516
01517
01518  mflag = .true.
01519  IF (PRESENT(mask)) mflag = mask
01520
01521  IF (mflag) THEN
01522    dum = a
01523    a   = b
01524    b   = dum
01525  END IF
01526
01527 END SUBROUTINE swapint
01528
01529 !=====
01530
01531 !-----  

01532 ! S U B R O U T I N E   S W A P   S I N G L E
01533 !-----  

01555 !
01556 SUBROUTINE swapsingle(a, b, mask)
01557
01558  IMPLICIT NONE
01559
01560  ! Global variables
01561  REAL(SP), INTENT(INOUT)      :: a, b
01562  LOGICAL, OPTIONAL, INTENT(IN) :: mask
01563
01564  ! Local variables
01565  REAL(SP) :: dum
01566  LOGICAL :: mFlag
01567
01568
01569  mflag = .true.
01570  IF (PRESENT(mask)) mflag = mask
01571
01572  IF (mflag) THEN
01573    dum = a
01574    a   = b
01575    b   = dum
01576  END IF
01577
01578 END SUBROUTINE swapsingle
01579
01580 !=====
01581
01582 !-----  

01583 ! S U B R O U T I N E   S W A P   D O U B L E
01584 !-----  

01606 !
01607 SUBROUTINE swapdouble(a, b, mask)
01608
01609  IMPLICIT NONE
01610
01611  ! Global variables
01612  REAL(HP), INTENT(INOUT)      :: a, b
01613  LOGICAL, OPTIONAL, INTENT(IN) :: mask

```

```
01614      ! Local variables
01615      REAL(HP) :: dum
01616      LOGICAL :: mFlag
01617
01618
01619      mflag = .true.
01620      IF (PRESENT(mask)) mflag = mask
01621
01622      IF (mflag) THEN
01623          dum = a
01624          a   = b
01625          b   = dum
01626      END IF
01627
01628
01629      END SUBROUTINE swapdouble
01630
01631 !=====
01632
01633 !-----
01634 ! S U B R O U T I N E   S W A P   I N T   V E C
01635 !-----
01636
01637
01638 SUBROUTINE swapintvec(a, b, mask)
01639
01640     IMPLICIT NONE
01641
01642     ! Global variables
01643     INTEGER, DIMENSION(:, ), INTENT(INOUT) :: a, b
01644     LOGICAL, OPTIONAL, INTENT(IN)          :: mask
01645
01646     ! Local variables
01647     INTEGER, DIMENSION(SIZE(a)) :: dum
01648     LOGICAL                      :: mFlag
01649
01650
01651     mflag = .true.
01652     IF (PRESENT(mask)) mflag = mask
01653
01654     IF (mflag) THEN
01655         dum = a
01656         a   = b
01657         b   = dum
01658     END IF
01659
01660
01661     END SUBROUTINE swapintvec
01662
01663 !=====
01664
01665 !-----
01666 ! S U B R O U T I N E   S W A P   S I N G L E   V E C
01667 !-----
01668
01669
01670 SUBROUTINE swapsinglevec(a, b, mask)
01671
01672     IMPLICIT NONE
01673
01674     ! Global variables
01675     REAL(SP), DIMENSION(:, ), INTENT(INOUT) :: a, b
01676     LOGICAL, OPTIONAL, INTENT(IN)          :: mask
01677
01678     ! Local variables
01679     REAL(SP), DIMENSION(SIZE(a)) :: dum
01680     LOGICAL                      :: mFlag
01681
01682
01683     !-----
01684
01685 ! S U B R O U T I N E   S W A P   D O U B L E   V E C
01686 !-----
01687
01688
01689 SUBROUTINE swapdoublevec(a, b, mask)
01690
01691     IMPLICIT NONE
01692
01693     ! Global variables
01694     REAL(DP), DIMENSION(:, ), INTENT(INOUT) :: a, b
01695     LOGICAL, OPTIONAL, INTENT(IN)          :: mask
01696
01697     ! Local variables
01698     REAL(DP), DIMENSION(SIZE(a)) :: dum
01699     LOGICAL                      :: mFlag
01700
01701
01702     mflag = .true.
01703     IF (PRESENT(mask)) mflag = mask
01704
01705     IF (mflag) THEN
01706         dum = a
01707         a   = b
01708         b   = dum
01709     END IF
01710
01711
01712     END SUBROUTINE swapdoublevec
01713
01714 !=====
01715
01716 !-----
```

```

01737 !-----
01759 !-----
01760 SUBROUTINE swapdoublevec(a, b, mask)
01761
01762     IMPLICIT NONE
01763
01764     ! Global variables
01765     REAL(HP), DIMENSION(:, ), INTENT(INOUT) :: a, b
01766     LOGICAL, OPTIONAL, INTENT(IN)           :: mask
01767
01768     ! Local variables
01769     REAL(HP), DIMENSION(SIZE(a)) :: dum
01770     LOGICAL                      :: mFlag
01771
01772
01773     mflag = .true.
01774     IF (PRESENT(mask)) mflag = mask
01775
01776     IF (mflag) THEN
01777         dum = a
01778         a   = b
01779         b   = dum
01780     END IF
01781
01782 END SUBROUTINE swapdoublevec
01783
01784 !=====
01785 !-----
01786 !-----  

01787 ! F U N C T I O N   A R T H   I N T
01788 !-----
01789 !-----
01808
01809 pure FUNCTION arthint(first, increment, n) RESULT(arthOut)
01810
01811     IMPLICIT NONE
01812
01813     ! Global variables
01814     INTEGER, INTENT(IN)    :: first, increment
01815     INTEGER, INTENT(IN)    :: n
01816     INTEGER, DIMENSION(n) :: arthout
01817
01818     ! Local variables
01819     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01820     INTEGER :: k, k2
01821     INTEGER :: temp
01822
01823
01824     IF (n > 0) arthout(1) = first
01825
01826     IF (n <= nparth) THEN
01827         DO k = 2, n
01828             arthout(k) = arthout(k - 1) + increment
01829         END DO
01830     ELSE
01831         DO k = 2, nparth2
01832             arthout(k) = arthout(k - 1) + increment
01833         END DO
01834
01835         temp = increment * nparth2
01836         k = nparth2
01837
01838         DO
01839             IF (k >= n) EXIT
01840             k2 = k + k
01841             arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01842             temp = temp + temp
01843             k = k2
01844         END DO
01845     END IF
01846
01847     RETURN
01848
01849 END FUNCTION arthint
01850
01851 !=====
01852 !-----
01853 !-----  

01854 ! F U N C T I O N   A R T H   S I N G L E
01855 !-----
01875 !-----
01876 pure FUNCTION arthsingle(first, increment, n) RESULT(arthOut)

```

```

01877
01878     IMPLICIT NONE
01879
01880     ! Global variables
01881     REAL(sp), INTENT(IN)    :: first, increment
01882     INTEGER, INTENT(IN)    :: n
01883     REAL(sp), DIMENSION(n) :: arthout
01884
01885     ! Local variables
01886     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01887     INTEGER :: k, k2
01888     REAL(sp) :: temp
01889
01890
01891     IF (n > 0) arthout(1) = first
01892
01893     IF (n <= nparth) THEN
01894         DO k = 2, n
01895             arthout(k) = arthout(k - 1) + increment
01896         END DO
01897     ELSE
01898         DO k = 2, nparth2
01899             arthout(k) = arthout(k - 1) + increment
01900         END DO
01901
01902     temp = increment * nparth2
01903     k = nparth2
01904
01905     DO
01906         IF (k >= n) EXIT
01907         k2 = k + k
01908         arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01909         temp = temp + temp
01910         k = k2
01911     END DO
01912 END IF
01913
01914     RETURN
01915
01916 END FUNCTION arthsingle
01917
01918 !=====
01919
01920 !-----
01921 ! F U N C T I O N   A R T H D O U B L E
01922 !-----
01923
01943 pure FUNCTION arthdouble(first, increment, n) RESULT(arthOut)
01944
01945     IMPLICIT NONE
01946
01947     ! Global variables
01948     REAL(hp), INTENT(IN)    :: first, increment
01949     INTEGER, INTENT(IN)    :: n
01950     REAL(hp), DIMENSION(n) :: arthout
01951
01952     ! Local variables
01953     INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01954     INTEGER :: k, k2
01955     REAL(hp) :: temp
01956
01957
01958     IF (n > 0) arthout(1) = first
01959
01960     IF (n <= nparth) THEN
01961         DO k = 2, n
01962             arthout(k) = arthout(k - 1) + increment
01963         END DO
01964     ELSE
01965         DO k = 2, nparth2
01966             arthout(k) = arthout(k - 1) + increment
01967         END DO
01968
01969     temp = increment * nparth2
01970     k = nparth2
01971
01972     DO
01973         IF (k >= n) EXIT
01974         k2 = k + k
01975         arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01976         temp = temp + temp

```

```
01977      k = k2
01978      END DO
01979      END IF
01980
01981      RETURN
01982
01983      END FUNCTION arthdouble
01984
01985 !=====
01986
01987 END MODULE sortutils
```

20.39 /home/takis/CSDL/parwinds-doc/src/timedateutils.F90 File Reference

Data Types

- interface [timedateutils::timeconv](#)
- interface [timedateutils::gregtojulday](#)
- interface [timedateutils::splittimetimestamp](#)

Modules

- module [timedateutils](#)

Functions/Subroutines

- subroutine [timedateutils::timeconvise](#) (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::gregtojuldayise](#) (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::splittimetimestamp](#) (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)

- subroutine `timedateutils::splitdatetimestring2` (`inDateTime`, `iDate`, `iTime`)
 - Splits a date string into 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` (`iDate`, `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.*
- character(`len(inpstring)`) function, private `timedateutils::upp` (`inpString`)
 - Convert a string to upper-case.*

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`

20.39.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `timedateutils.F90`.

20.40 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 timeconviseC  

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 ! S U B R O U T I N E   T I M E   C O N V   I S E C
00097 !-----
00124 !-----
00125 SUBROUTINE timeconviseC(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 timeconviseC
00167 !=====
00168 !-----
00170 !----- S U B R O U T I N E   T I M E   C O N V   R S E C
00171 !-----
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 ! S U B R O U T I N E   T I M E   C O N V   A D C I R C <- TO BE DELETED
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 ! F U N C T I O N   L E A P   Y E A R
00299 !-----
00314 !-----
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 ! F U N C T I O N   Y E A R   D A Y S
00350 !-----
00351 !-----
00352
00353      INTEGER FUNCTION yeardays(iYear) RESULT(myVal)
00354
00355      IMPLICIT NONE
00356
00357      INTEGER, INTENT(IN) :: iyear
00358
00359      !----- START CALCULATIONS -----
00360
00361      myval = 365
00362      IF (leapyear(iyear)) myval = 366
00363
00364      RETURN
00365  END FUNCTION yeardays
00366
00367 !=====
00368
00369 !-----
00370 ! F U N C T I O N   M O N T H   D A Y S
00371 !-----
00372 !-----
00373
00374
00375      INTEGER FUNCTION monthdays(iYear, iMonth) RESULT(myVal)
00376
00377      IMPLICIT NONE
00378
00379      ! Global variables
00380      INTEGER, INTENT(IN) :: iyear, imonth
00381
00382      ! Local variables
00383      INTEGER :: leap, monlen(12, 2)
00384
00385      !----- START CALCULATIONS -----
00386
00387      IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12)) THEN
00388          myval = imissv
00389
00390          RETURN
00391      END IF
00392
00393      ! Initialize lengths of months:
00394      monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31, &
00395                      31, 29, 31, 30, 31, 31, 31, 30, 31, 30, 31 /), &
00396                      (/ 12, 2 /))
00397
00398      leap = 1
00399      IF (leapyear(iyear)) leap = 2
00400
00401
00402
00403
00404
00405
00406
00407
00408
00409
00410
00411
00412
00413
00414
00415
00416
00417
00418
00419
00420
00421
00422
00423
00424
00425
00426
00427
00428

```

```

00429     myval = monlen(imonth, leap)
00430
00431     RETURN
00432 END FUNCTION monthdays
00433
00434 !=====
00435 !-----
00436 ! F U N C T I O N   D A Y   O F   Y E A R
00437 !-----
00438 !-----
00439 !-----
00440 INTEGER FUNCTION dayofyear(iYear, iMonth, iDay) RESULT(myVal)
00441
00442     IMPLICIT NONE
00443
00444     ! Global variables
00445     INTEGER, INTENT(IN) :: iyear, imonth, iday
00446
00447     ! Local variables
00448     REAL(sz) :: jd0, jd1
00449
00450     !---- START CALCULATIONS ----
00451
00452     jd0 = gregtojulday(iyear, 1, 1, 0, 0, 0)
00453     jd1 = gregtojulday(iyear, imonth, iday, 0, 0, 0)
00454
00455     IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
00456         myval = imissv
00457
00458         RETURN
00459     END IF
00460
00461     myval = int(jd1 - jd0 + 1.0_sz)
00462
00463     RETURN
00464 END FUNCTION dayofyear
00465
00466 !=====
00467 !-----
00468 ! F U N C T I O N   G R E G   T O   J U L   D A Y   I S E C
00469 !-----
00470 !-----
00471 !-----
00472 REAL(sz) function gregtojuldayisec(iyear, imonth, iday, ihour, imin, isec, mjd) result(myval)
00473
00474     IMPLICIT NONE
00475
00476     ! Global variables
00477     INTEGER, INTENT(IN)          :: iyear, imonth, iday, ihour, imin, isec
00478     INTEGER, OPTIONAL, INTENT(IN) :: mjd
00479
00480     ! Local variables
00481     INTEGER :: leap, monlen(12, 2)
00482     LOGICAL :: modjul
00483     REAL(hp) :: templ, temp2
00484
00485     !---- START CALCULATIONS ----
00486
00487     modjul = .false.
00488     IF (PRESENT(mjd)) THEN
00489         modjul = (mjd > 0)
00490     ELSE
00491         modjul = (usemodjulday > 0)
00492     END IF
00493
00494     ! Initialize lengths of months:
00495     monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 30, 31, 30, 31, 31, &
00496                      31, 29, 31, 30, 31, 31, 31, 30, 31, 30, 31 /), &
00497                      (/ 12, 2 /))
00498
00499     ! This function intentionally works on Gregorian dates only. For modeling
00500     ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00501     ! it is not necessary to go beyond that date.
00502
00503     ! Is this a LEAP year?
00504     leap = 1
00505     IF (leapyear(iyear)) leap = 2
00506
00507     IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
00508         myval = rmissv
00509
00510
00511
00512
00513
00514
00515
00516
00517
00518
00519
00520
00521
00522
00523
00524
00525
00526
00527
00528
00529
00530
00531
00532
00533
00534
00535
00536
00537
00538
00539
00540
00541
00542
00543
00544
00545
00546
00547
00548
00549
00550
00551
00552
00553
00554
00555
00556
00557
00558
00559
00560
00561
00562
00563
00564
00565
00566
00567
00568
00569
00570
00571
00572
00573

```

```

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     (rsec < 0) .OR. (rsec > 60)) THEN
00580     myval = rmissv
00581
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(rsec, hp) - 43200.0_hp
00592
00593     IF (modjul) THEN
00594         myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
00595     ELSE
00596         myval = temp2 + (temp1 / 86400.0_hp)
00597     END IF
00598 END IF
00599
00600     RETURN
00601 END FUNCTION gregtojuldayisec
00602
00603 !=====
00604
00605 !-----+
00606 ! F U N C T I O N   G R E G   T O   J U L   D A Y   R S E C
00607 !-----+
00653 !-
00654 REAL(sz) function gregtojuldayrsec(iyear, imonth, iday, ihour, imin, rsec, mjd) result(myval)
00655
00656 IMPLICIT NONE
00657
00658 ! Global variables
00659 INTEGER, INTENT(IN) :: iyear, imonth, iday, ihour, imin
00660 REAL(sz), INTENT(IN) :: rsec
00661 INTEGER, OPTIONAL, INTENT(IN) :: mjd
00662
00663 ! Local variables
00664 INTEGER :: leap, monlen(12, 2)
00665 LOGICAL :: modjul
00666 REAL(hp) :: temp1, temp2
00667
00668 !---- START CALCULATIONS ----
00669
00670 modjul = .false.
00671 IF (PRESENT(mjd)) THEN
00672     modjul = (mjd > 0)
00673 ELSE
00674     modjul = (usemodjulday > 0)
00675 END IF
00676
00677 ! Initialize lengths of months:
00678 monlen = reshape(/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, &
00679                 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 /), &
00680             (/ 12, 2 /))
00681
00682 ! This function intentionally works on Gregorian dates only. For modeling
00683 ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00684 ! it is not necessary to go beyond that date.
00685
00686 ! Is this a LEAP year?
00687 leap = 1
00688 IF (leapyear(iyear)) leap = 2
00689
00690 IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
00691     myval = rmissv
00692
00693     RETURN
00694 ELSE IF ((imonth < 1) .OR. (imonth > 12)) .OR. &
00695     (iday < 1) .OR. (iday > monlen(imonth, leap)) .OR. &
00696     (ihour < 0) .OR. (ihour > 23) .OR. &
00697     (imin < 0) .OR. (imin > 59) .OR. &
00698     (rsec < 0) .OR. (rsec > 60)) THEN
00699     myval = rmissv

```

```

00700      RETURN
00701
00702  ELSE
00703    templ = int((imonth - 14.0_hp) / 12.0_hp)
00704    temp2 = iday - 32075.0_hp
00705    + int(1461.0_hp * (iyear + 4800.0_hp + templ) / 4.0_hp) &
00706    + int(367.0_hp * (imonth - 2.0_hp - templ * 12.0_hp) / 12.0_hp) &
00707    - int(3.0_hp * int((iyear + 4900.0_hp + templ) / 100.0_hp) / 4.0_hp)
00708    templ = real(ihour, hp) * 3600.0_hp &
00709    + real(imin, hp) * 60.0_hp &
00710    + real(rsec, hp) - 43200.0_hp
00711
00712  IF (modjul) THEN
00713    myval = temp2 + (templ / 86400.0_hp) - mdjoffset
00714  ELSE
00715    myval = temp2 + (templ / 86400.0_hp)
00716  END IF
00717 END IF
00718
00719  RETURN
00720 END FUNCTION gregtojuldaysec
00721
00722 !=====
00723
00724 !-----+
00725 ! F U N C T I O N   G R E G   T O   J U L   D A Y   I S E C   2
00726 !-----+
00774 !-----+
00775 REAL(sz) function gregtojulday2(idate, itime, mjd) result(myval)
00776
00777 IMPLICIT NONE
00778
00779 ! Global variables
00780 INTEGER, INTENT(IN) :: idate, itime
00781 INTEGER, OPTIONAL, INTENT(IN) :: mjd
00782
00783 ! Local variables
00784 INTEGER :: iyear, imonth, iday, ihour, imin, isec
00785 INTEGER :: leap, monlen(12, 2)
00786 LOGICAL :: modjul
00787 REAL(hp) :: templ, temp2
00788
00789 !---- START CALCULATIONS -----
00790
00791 modjul = .false.
00792 IF (PRESENT(mjd)) THEN
00793   modjul = (mjd > 0)
00794 ELSE
00795   modjul = (usemodjulday > 0)
00796 END IF
00797
00798 ! Initialize lengths of months:
00799 monlen = reshape((/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, &
00800           31, 29, 31, 30, 31, 30, 31, 30, 31, 30, 31, 30, 31 /), &
00801           (/ 12, 2 /))
00802
00803 ! This function intentionally works on Gregorian dates only. For modeling
00804 ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00805 ! it is not necessary to go beyond that date.
00806
00807 CALL splitdate(idate, iyear, imonth, iday)
00808 CALL splitdate(itime, ihour, imin, isec)
00809
00810 ! Is this a LEAP year?
00811 leap = 1
00812 IF (leapyear(iyear)) leap = 2
00813
00814 IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12) &
00815           .OR. (iday < 1) .OR. (iday > monlen(imonth, leap)) &
00816           .OR. (ihour < 0) .OR. (ihour > 23) &
00817           .OR. (imin < 0) .OR. (imin > 59) &
00818           .OR. (isec < 0) .OR. (isec > 60)) THEN
00819   myval = rmissv
00820
00821   RETURN
00822 ELSE
00823   IF (idate < firstgregdate) THEN
00824     myval = rmissv
00825
00826   RETURN
00827 ELSE

```

```

00828      temp1 = int((imonth - 14.0_hp) / 12.0_hp)
00829      temp2 = iday - 32075.0_hp
00830          + int(1461.0_hp * (iyear + 4800.0_hp + temp1) / 4.0_hp)
00831          + int(367.0_hp * (imonth - 2.0_hp - temp1 * 12.0_hp) / 12.0_hp)
00832          - int(3.0_hp * int((iyear + 4900.0_hp + temp1) / 100.0_hp) / 4.0_hp)
00833      temp1 = real(ihour, hp) * 3600.0_hp    &
00834          + real(imin, hp) * 60.0_hp        &
00835          + real(isec, hp) - 43200.0_hp
00836
00837      IF (modjul) THEN
00838          myval = temp2 + (temp1 / 86400.0_hp) - mdjoffset
00839      ELSE
00840          myval = temp2 + (temp1 / 86400.0_hp)
00841      END IF
00842      END IF
00843  END IF
00844
00845      RETURN
00846 END FUNCTION gregtojulday2
00847
00848 !=====
00849 !-----!
00850 ! S U B R O U T I N E   J U L   D A Y   T O   G R E G
00851 !-----!
00852 !-----!
00853 SUBROUTINE juldaytogram(julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
00854
00855     IMPLICIT NONE
00856
00857     ! Global Variables
00858     REAL(SZ), INTENT(IN)      :: julDay
00859     INTEGER, OPTIONAL, INTENT(IN) :: mJD
00860     INTEGER, INTENT(OUT)       :: iYear, iMonth, iDay, iHour, iMin, iSec
00861
00862     ! Local Variables
00863     REAL(HP) :: temp1, temp2, temp3, temp4, temp5
00864     REAL(HP) :: thisJulDay, myJulDay, delta
00865     INTEGER   :: nTry
00866     LOGICAL   :: modJul
00867
00868     !---- START CALCULATIONS ----
00869
00870     modJul = .false.
00871     IF (PRESENT(mjd)) THEN
00872         modJul = (mjd > 0)
00873     ELSE
00874         modJul = (usemodjulday > 0)
00875     END IF
00876
00877     IF (modJul) THEN
00878         thisJulDay = julDay + mdjoffset
00879     ELSE
00880         thisJulDay = julDay
00881     END IF
00882
00883     ! Check for valid Julian day (Gregorian calendar only)
00884     IF (thisJulDay < offfirstgregday) THEN
00885         iyear = imissv
00886         imonth = imissv
00887         iday = imissv
00888         ihour = imissv
00889         imin = imissv
00890         isec = imissv
00891
00892         RETURN
00893     END IF
00894
00895     delta = 0.0_hp
00896     ntry = 1
00897     DO WHILE (ntry <= 2)
00898         myJulDay= thisJulDay + delta
00899         temp4 = myJulDay
00900         temp5 = dmod(myJulDay, 1.0_hp)
00901
00902         IF (temp5 < 0.5) THEN
00903             temp3 = 0.5_hp + temp5
00904             temp4 = aint(temp4)
00905         ELSE
00906             temp3 = temp5 - 0.5_hp
00907             temp4 = aint(temp4) + 1.0_hp
00908
00909     END DO
00910
00911     iYear = int(myJulDay)
00912     iMonth = int((myJulDay - iYear) * 12.0_hp)
00913     iDay = int((myJulDay - iYear - iMonth * 1.0_hp) * 31.0_hp)
00914
00915     iHour = int((myJulDay - iYear - iMonth * 1.0_hp - iDay * 1.0_hp) * 24.0_hp)
00916     iMin = int((myJulDay - iYear - iMonth * 1.0_hp - iDay * 1.0_hp - iHour * 1.0_hp) * 60.0_hp)
00917     iSec = int((myJulDay - iYear - iMonth * 1.0_hp - iDay * 1.0_hp - iHour * 1.0_hp - iMin * 1.0_hp) * 60.0_hp)
00918
00919     RETURN
00920
00921 END SUBROUTINE juldaytogram

```

```

00953     END IF
00954
00955     temp1 = temp4 + 68569.0
00956     temp2 = aint(4.0_hp * temp1 / 146097.0_hp)
00957     temp1 = temp1 - aint((146097.0_hp * temp2 + 3.0_hp) / 4.0_hp)
00958     iyear = int(4000.0_hp * (temp1 + 1.0_hp) / 1461001.0_hp)
00959     temp1 = temp1 - aint((1461.0_hp * iyear) / 4.0_hp) + 31.0_hp
00960     imonth = int(80.0_hp * temp1 / 2447.0_hp)
00961     iday = int(temp1 - aint(2447.0_hp * imonth / 80.0_hp))
00962     temp1 = aint(imonth / 11.0_hp)
00963     imonth = int(imonth + 2.0 - 12.0_hp * temp1)
00964     iyear = int(100.0_hp * (temp2 - 49.0_hp) + iyear + temp1)
00965     ihour = int(temp3 * 24.0_hp)
00966     imin = int(temp3 * 1440.0_hp - 60.0_hp * ihour)
00967     isec = nint(temp3 * 86400.0_hp - 3600.0_hp * ihour - 60.0_hp * imin)
00968
00969     IF (isec >= 60) THEN
00970         IF (ntry < 2) THEN
00971             delta = 0.49999_hp / 86400.0_hp
00972             ntry = ntry + 1
00973         ELSE
00974             iyear = imissv
00975             EXIT
00976         END IF
00977     ELSE
00978         EXIT
00979     END IF
00980 END DO
00981
00982 END SUBROUTINE juldaytogreg
00983
00984 !=====
00985
00986 !-----
00987 ! S U B R O U T I N E   D A Y   O F   Y E A R   T O   G R E G
00988 !-----
01009 !
01010 SUBROUTINE dayofyeartogreg(inYR, inDY, iYear, iMonth, iDay)
01011
01012     IMPLICIT NONE
01013
01014     ! Global Variables
01015     INTEGER, INTENT(IN) :: inYR, inDY
01016     INTEGER, INTENT(OUT) :: iYear, iMonth, iDay
01017
01018     ! Local Variables
01019     REAL(SZ) :: julDay
01020     INTEGER :: yr, mo, da, hh, mm, ss
01021
01022     !---- START CALCULATIONS ----
01023
01024     ! Check for valid day of year (Gregorian calendar only)
01025     IF ((inyr < 1582) .OR. (indy < 1) .OR. (indy > 366) ) THEN
01026         iyear = imissv
01027         imonth = imissv
01028         iday = imissv
01029
01030         RETURN
01031     END IF
01032
01033     julday = gregojulday(inyr, 1, 1, 0, 0, 0) + (indy - 1) * 1.0_hp
01034
01035     CALL juldaytogreg(julday, yr, mo, da, hh, mm, ss)
01036
01037     iyear = yr
01038     imonth = mo
01039     iday = da
01040
01041 END SUBROUTINE dayofyeartogreg
01042
01043 !=====
01044
01045 !-----
01046 ! S U B R O U T I N E   S P L I T   D A T E   T I M E   S T R I N G
01047 !-----
01071 !
01072 SUBROUTINE splitdatetimestring(inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
01073
01074     IMPLICIT NONE
01075
01076     ! Global Variables

```

```

01077      CHARACTER(LEN=*) , INTENT(IN)    :: inDateTime
01078      INTEGER, INTENT(OUT)          :: iYear, iMonth, iDay, iHour, iMin, iSec
01079
01080      ! Local Variables
01081      CHARACTER(LEN=LEN(inDateTime)) :: tmpDateStr
01082      INTEGER                      :: errIO
01083
01084      !----- START CALCULATIONS -----
01085
01086      tmpdatestr = preprocessdatetimestring(indatetime)
01087
01088      IF (trim(tmpdatestr) == "") THEN
01089          iyear = imissv
01090          imonth = 0
01091          iday = 0
01092          ihour = 0
01093          imin = 0
01094          isec = 0
01095
01096      RETURN
01097  END IF
01098
01099      READ(tmpdatestr(1:4), '(I4.4)', iostat=errio) iyear
01100      IF ((errio /= 0) .OR. (iyear < 1582)) iyear = imissv
01101
01102      READ(tmpdatestr(5:6), '(I2.2)', iostat=errio) imonth
01103      IF ((errio /= 0) .OR. (imonth < 1) .OR. (imonth > 12)) imonth = 0
01104
01105      READ(tmpdatestr(7:8), '(I2.2)', iostat=errio) iday
01106      IF ((errio /= 0) .OR. (iday < 0) .OR. (iday > monthdays(iyear, imonth))) iday = 0
01107
01108      READ(tmpdatestr(9:10), '(I2.2)', iostat=errio) ihour
01109      IF ((errio /= 0) .OR. (ihour < 0) .OR. (ihour >= 23)) ihour = 0
01110
01111      READ(tmpdatestr(11:12), '(I2.2)', iostat=errio) imin
01112      IF ((errio /= 0) .OR. (imin < 0) .OR. (imin >= 60)) imin = 0
01113
01114      READ(tmpdatestr(13:14), '(I2.2)', iostat=errio) isec
01115      IF ((errio /= 0) .OR. (isec < 0) .OR. (isec >= 60)) isec = 0
01116
01117  END SUBROUTINE splitdatetimestring
01118
01119 !=====
01120
01121 !-----
01122 ! S U B R O U T I N E   S P L I T   D A T E   T I M E   S T R I N G _ 2
01123 !-----
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 !----- FUNCTION preprocessdatetimestring(indateTime) Result(myValOut)
01159
01160
01161
01162
01163
01164
01165 !=====
01166
01167 !-----
01168 ! F U N C T I O N   P R E   P R O C E S S   D A T E   T I M E   S T R I N G
01169 !-----
01170
01171 !-----
01172
01173
01174
01175
01176
01177
01178
01179
01180
01181
01182
01183
01184
01185
01186

```

```

01187      IMPLICIT NONE
01188
01189      ! Global Variables
01190      CHARACTER(LEN=*) , INTENT(IN)    :: indatetime
01191      CHARACTER(LEN=LEN(indatetime)) :: myvalout
01192
01193      ! Local Variables
01194      CHARACTER(LEN=1)                :: c
01195      INTEGER                         :: i, ipos
01196
01197      !----- START CALCULATIONS -----
01198
01199      myvalout = blank
01200      ipos = 1
01201
01202      DO i = 1, len(indatetime)
01203          c = indatetime(i:i)
01204          IF ((48 <= ichar(c)) .AND. (ichar(c) <= 57)) THEN
01205              myvalout(ipos:ipos) = c
01206              ipos = ipos + 1
01207          ENDIF
01208      END DO
01209
01210      RETURN
01211
01212  END FUNCTION preprocessdatetimestring
01213
01214 !=====
01215 !
01216 !-----+
01217 ! F U N C T I O N   J O I N   D A T E
01218 !
01219 !-----+
01220 INTEGER FUNCTION joindate(iYear, iMonth, iDay) RESULT(myVal)
01221
01222      IMPLICIT NONE
01223
01224      ! Global Variables
01225      INTEGER, INTENT(IN) :: iyear, imonth, iday
01226
01227      !----- START CALCULATIONS -----
01228
01229      myval = iyear * 10000 + imonth * 100 + iday
01230
01231  END FUNCTION joindate
01232
01233 !=====
01234 !
01235 !-----+
01236 ! S U B R O U T I N E   S P L I T   D A T E
01237 !
01238 !-----+
01239 SUBROUTINE splitdate(inDate, iYear, iMonth, iDay)
01240
01241      IMPLICIT NONE
01242
01243      ! Global Variables
01244      INTEGER, INTENT(IN)  :: inDate
01245      INTEGER, INTENT(OUT) :: iYear, iMonth, iDay
01246
01247      !----- START CALCULATIONS -----
01248
01249      iyear = indate / 10000
01250      imonth = indate / 100 - iyear * 100
01251      iday = indate - imonth * 100 - iyear * 10000
01252
01253  END SUBROUTINE splitdate
01254
01255 !=====
01256 !
01257 !-----+
01258 !-----+
01259 !-----+
01260 !-----+
01261 !-----+
01262 !-----+
01263 !-----+
01264 !-----+
01265 !-----+
01266 !-----+
01267 !-----+
01268 !-----+
01269 !-----+
01270 !-----+
01271 !-----+
01272 !-----+
01273 !-----+
01274 !-----+
01275 !-----+
01276 !-----+
01277 !-----+
01278 !-----+
01279 !-----+
01280 !-----+
01281 !-----+
01282 !-----+
01283 !-----+
01284 !-----+
01285 !-----+
01286 !-----+
01287 !-----+
01288 !-----+
01289 !-----+
01290 !-----+
01291 !-----+
01292 !-----+
01293 !-----+
01294 !-----+
01295 !-----+
01296 !=====
```

```

01340      INTEGER, OPTIONAL, INTENT(IN)          :: sep, hour, min, sec
01341      CHARACTER(LEN=*) , OPTIONAL, INTENT(IN) :: units, zone
01342      INTEGER, OPTIONAL, INTENT(OUT)         :: err ! Error status, 0 if success, nonzero in case of format
01343      error.
01344      ! The resulting date time string. Considering using trim() on it.
01345      CHARACTER(LEN=64) :: myvalout
01346      CHARACTER(LEN=20) :: myunits, myzone
01347      CHARACTER(LEN=1)  :: mytimesep
01348      INTEGER           :: myhour, mymin, mysec, myerr
01349
01350      myhour = 0
01351      IF (PRESENT(hour)) myhour = hour
01352      mymin = 0
01353      IF (PRESENT(min))  mymin = min
01354      mysec = 0
01355      IF (PRESENT(sec))  mysec = sec
01356
01357      mytimesep = ' '
01358      IF (PRESENT(sep)) THEN
01359          IF (sep > 0) mytimesep = 'T'
01360          IF (sep <= 0) mytimesep = ' '
01361      END IF
01362
01363      IF (PRESENT(units)) THEN
01364          SELECT CASE(trim(adjustl(upp(units))))
01365              CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01366                  myunits = 'seconds since'
01367              CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01368                  myunits = 'minutes since'
01369              CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01370                  myunits = 'hours since'
01371              CASE('DAYS', 'DAY', 'DA', 'D')
01372                  myunits = 'days since'
01373              CASE('WEEKS', 'WEEK', 'WE', 'W')
01374                  myunits = 'weeks since'
01375              CASE DEFAULT
01376                  myvalout = ' '
01377          END SELECT
01378      ELSE
01379          myunits = ' '
01380      END IF
01381
01382      IF (PRESENT(zone)) THEN
01383          myzone = adjustl(zone)
01384      ELSE
01385          myzone = ' '
01386      END IF
01387
01388      !WRITE(myValOut, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":" , i2.2, ":" , i2.2, "Z")', IOSTAT=myErr) &
01389      !           year, month, day, myTimeSep, myHour, myMin, mySec
01390      WRITE(myvalout, '(i4.4, "-", i2.2, "-", i2.2, a1, i2.2, ":" , i2.2, ":" , i2.2)', iostat=myerr) &
01391      !           year, month, day, mytimesep, myhour, mymin, mysec
01392
01393      IF (len_trim(myunits) /= 0) THEN
01394          myvalout = trim(myunits) // " " // trim(myvalout)
01395      END IF
01396
01397      IF (len_trim(myzone) /= 0) THEN
01398          myvalout = trim(myvalout) // " " // trim(myzone)
01399      END IF
01400
01401      IF (PRESENT(err)) err = myerr
01402
01403      RETURN
01404
01405  END FUNCTION datetime2string
01406
01407 !=====
01408
01409 !-----
01410 ! F U N C T I O N   G E T   T I M E   C O N V   S E C
01411 !-----
01429 !-----
01430 REAL(sz) function gettimeconvsec(units, invert) result(myvalout)
01431
01432     IMPLICIT NONE
01433
01434     CHARACTER(LEN=*) , INTENT(IN)  :: units
01435     INTEGER, OPTIONAL, INTENT(IN) :: invert
01436

```

```

01437     INTEGER :: myinvert
01438     CHARACTER(LEN=LEN(units)) :: myunits
01439     REAL(sz), PARAMETER :: minsecs = 60.0_sz
01440     REAL(sz), PARAMETER :: hoursecs = 3600.0_sz
01441     REAL(sz), PARAMETER :: daysecs = 86400.0_sz
01442     REAL(sz), PARAMETER :: weeksecs = 604800.0_sz
01443
01444
01445     myinvert = 0
01446     IF (PRESENT(invert)) THEN
01447         IF (invert > 0) myinvert = 1
01448         IF (invert <= 0) myinvert = 0
01449     END IF
01450
01451     myunits = adjust1(units)
01452     IF (myinvert == 0) THEN
01453         SELECT CASE(trim(upp(myunits)))
01454             CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01455                 myvalout = 1.0_sz
01456             CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01457                 myvalout = minsecs
01458             CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01459                 myvalout = hoursecs
01460             CASE('DAYS', 'DAY', 'DA', 'D')
01461                 myvalout = daysecs
01462             CASE('WEEKS', 'WEEK', 'WE', 'W')
01463                 myvalout = weeksecs
01464             CASE DEFAULT
01465                 myvalout = 1.0_sz
01466         END SELECT
01467     ELSE
01468         SELECT CASE(trim(upp(myunits)))
01469             CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01470                 myvalout = 1.0_sz
01471             CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01472                 myvalout = 1.0_sz / minsecs
01473             CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01474                 myvalout = 1.0_sz / hoursecs
01475             CASE('DAYS', 'DAY', 'DA', 'D')
01476                 myvalout = 1.0_sz / daysecs
01477             CASE('WEEKS', 'WEEK', 'WE', 'W')
01478                 myvalout = 1.0_sz / weeksecs
01479             CASE DEFAULT
01480                 myvalout = 1.0_sz
01481         END SELECT
01482     END IF
01483
01484     RETURN
01485
01486 END FUNCTION gettimeconvsec
01487
01488 !=====
01489 !-----!
01490 !-----!
01491 ! FUNCTION ELAPSED SECS
01492 !-----!
01515 !-----!
01516 REAL(sz) function elapsedsecs(intime1, intime2, inunits) result(myval)
01517
01518 IMPLICIT NONE
01519
01520 ! Global Variables
01521 REAL(sz), INTENT(IN) :: intime1, intime2
01522 CHARACTER(LEN=*) , OPTIONAL, INTENT(IN) :: inunits
01523
01524 ! Local Variables
01525 REAL(sz) :: uconfac
01526 CHARACTER(LEN=:), ALLOCATABLE :: unitsval
01527
01528 !---- START CALCULATIONS ----
01529
01530 IF (PRESENT(inunits)) THEN
01531     ALLOCATE(CHARACTER(LEN=LEN(inUnits)) :: unitsval)
01532     unitsval = inunits
01533 ELSE
01534     ALLOCATE(CHARACTER(LEN=1) :: unitsval)
01535     unitsval = 'S'
01536 END IF
01537
01538 uconfac = gettimeconvsec(unitsval)
01539

```

```

01540     myval = (intime2 - intime1) * uconfac
01541     myval = fixnearwholereal(myval, 0.001_sz)
01542
01543     DEALLOCATE(unitsval)
01544
01545     RETURN
01546
01547 END FUNCTION elapsedsecs
01548
01549 !=====
01550
01551 !-----+
01552 !-----+ F U N C T I O N   U P P
01553 !-----+
01554 !
01555 !
01556 FUNCTION upp(inpString) RESULT(outString)
01557
01558 CHARACTER(*), INTENT(IN) :: inpstring
01559
01560 INTEGER, PARAMETER :: duc = ichar('A') - ichar('a')
01561 CHARACTER(LEN(inpString)) :: outstring
01562 CHARACTER :: ch
01563 INTEGER :: i
01564
01565 DO i = 1, len(inpstring)
01566     ch = inpstring(i:i)
01567     IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
01568     outstring(i:i) = ch
01569 END DO
01570
01571 RETURN
01572
01573 END FUNCTION upp
01574
01575 !=====
01576
01577 END MODULE timedateutils

```

20.41 /home/takis/CSDL/parwinds-doc/src/utilities.F90 File Reference

Data Types

- interface [utilities::geotocpp](#)
- interface [utilities::cpptogeoo](#)
- 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::cpptogeoo_scalar](#) (x, y, lat0, lon0, lat, lon)
Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- subroutine [utilities::cpptogeoo_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::sphericaldistancehav](#) (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.
- integer function [utilities::realsscan](#) (String, Pos, Value)
Scans string looking for the leading single precision real numeric string.
- integer function [utilities::drealsscan](#) (String, Pos, Value)
Scans string looking for the leading double precision real numeric string.
- integer function [utilities::intsscan](#) (String, Pos, Signed, Value)
Scans string looking for the leading integer numeric string.

Variables

- integer, private `utilities::numbtfiles` = 0
- real(sz), parameter `utilities::closetol` = 0.001_SZ

20.41.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `utilities.F90`.

20.42 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_1d  

00032 END INTERFACE geotocpp  

00033  

00034 INTERFACE cptogeo  

00035   MODULE PROCEDURE cptogeo_scalar  

00036   MODULE PROCEDURE cptogeo_1d  

00037 END INTERFACE cptogeo  

00038  

00039 INTERFACE sphericaldistance  

00040   MODULE PROCEDURE sphericaldistance_scalar  

00041   MODULE PROCEDURE sphericaldistance_1d  

00042   MODULE PROCEDURE sphericaldistance_2d  

00043 END INTERFACE sphericaldistance  

00044 !-----  

00045  

00046  

00047 CONTAINS  

00048  

00049  

00050 !-----  

00051 !      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  

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
00104    WRITE(scratchmessage, '("The file : ", a, " was found. The file will be opened.")')
00105    trim(tmpfilename)
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 !     S U B R O U T I N E   R E A D   C O N T R O L   F I L E
00135 !-----+
00136
00137 !-----+
00138 SUBROUTINE readcontrolfile(inpFile)
00139
00140
00141    USE pahm_global
00142    USE pahm_messages
00143    USE timedateutils, ONLY : timeconv, splitdatetimestamp, joindate, gregtojulday, juldaytoggreg, &
00144                           gettimeconvsec, datetime2string
00145
00146    IMPLICIT NONE
00147
00148    ! Global variables
00149    CHARACTER(LEN=*, INTENT(IN))      :: inpFile
00150
00151    ! Local variables
00152    INTEGER, PARAMETER               :: maxNumELEM = 200
00153    INTEGER, PARAMETER               :: maxStrLEN = 512
00154
00155    LOGICAL                          :: fileFound   ! .TRUE. if the file is present
00156    CHARACTER(LEN=LEN(inpFile))      :: tmpFileName
00157    CHARACTER(LEN=maxStrLEN)         :: inpLine, outLine
00158    CHARACTER(LEN=40)                :: keyWord

```

```

00173
00174     INTEGER                      :: iUnit, errIO, status
00175
00176     INTEGER                      :: nPnts
00177     INTEGER                      :: nVal
00178     REAL(SZ), ALLOCATABLE        :: realVal(:)
00179     CHARACTER(LEN=maxStrLEN), ALLOCATABLE :: charVal(:)
00180     CHARACTER(LEN=maxStrLEN)      :: tmpCharVal
00181
00182     INTEGER                      :: iValOut(1)
00183     REAL(SZ)                     :: rValOut(1)
00184
00185     LOGICAL                      :: gotNBTRFILES = .false.
00186
00187     CHARACTER(LEN=maxStrLEN)      :: cntlFmtStr, fmtDimParInvalid, fmtParNotFound
00188     CHARACTER(LEN=FNAMELEN)       :: tmpStr
00189     REAL(SZ)                     :: jday
00190
00191
00192     ALLOCATE(realval(maxnumelem))
00193     ALLOCATE(charval(maxnumelem))
00194
00195
00196 !----- Initialize variables
00197 ! Global variables
00198     numbtfiles                  = 0
00199
00200 ! Local variables
00201     inline                       = blank
00202     outline                      = blank
00203     keyword                      = blank
00204     charval                      = blank
00205     cntlfmtstr                   = blank
00206     fmtdimparinvalid             = blank
00207     fmtparnotfound               = blank
00208     tmpstr                       = blank
00209
00210     iunit = lun_ctrl
00211     errio = 0
00212 !-----
00213
00214
00215     CALL setmessagesource("ReadControlFile")
00216
00217 !----- Establish the format variables
00218     cntlfmtstr = ' in control file ' // "<" // trim(adjustl(inpfile)) // ">" // "'"
00219
00220     fmtdimparinvalid = '(" Invalid dimension parameter: ", a, 1x, i0, 2x, '
00221     fmtdimparinvalid = trim(fmtdimparinvalid) // trim(cntlfmtstr) // ', 1x, a)'
00222
00223     fmtparnotfound = '(" Could not find input parameter: ", a, 1x, '
00224     fmtparnotfound = trim(fmtparnotfound) // trim(cntlfmtstr) // ', 1x, a)'
00225 !-----
00226
00227     tmpfilename = adjustl(inpfile)
00228
00229     INQUIRE(file=trim(tmpfilename), exist=filefound)
00230     IF (.NOT. filefound) THEN
00231         WRITE(lun_screen, '("The control file : ", a, " was not found, cannot continue.")')
00232     trim(tmpfilename)
00233
00234         stop ! file not found
00235     ELSE
00236         WRITE(lun_screen, '("The contol file : ", a, " was found and will be opened for reading.")')
00237     trim(tmpfilename)
00238     END IF
00239
00240         ! Open existing file
00241     OPEN(unit=iunit, file=trim(tmpfilename), status='OLD', action='READ', iostat=errio)
00242     IF (errio /= 0) THEN
00243         WRITE(lun_screen, '("Could not open the contol file: ", a, ".")') trim(tmpfilename)
00244
00245         stop ! file found but could not be opened
00246     END IF
00247
00248     DO WHILE (.true.)
00249         READ(unit=iunit, fmt='(a)', err=10, END=20) inpline
00250         status = parseline(inpline, outline, keyword, nval, charval, realval)
00251
00252         IF (status > 0) THEN
00253             SELECT CASE (touppercase(trim(keyword)))

```

```

00252      !----- CASE
00253      CASE ('TITLE')
00254          IF (nval == 1) THEN
00255              title = trim(adjustl(charval(nval)))
00256          ELSE
00257              WRITE(title, '(a, 1x, a)') trim(adjustl(title)), trim(adjustl(charval(nval)))
00258          END IF
00259
00260      !----- CASE
00261      CASE ('LOGFILENAME')
00262          IF (nval == 1) THEN
00263              logfilename = trim(adjustl(charval(nval)))
00264          ELSE
00265              IF (trim(adjustl(logfilename)) == "") THEN
00266                  logfilename = trim(adjustl(charval(nval)))
00267              END IF
00268          END IF
00269
00270      !----- CASE
00271      CASE ('WRITEPARAMS')
00272          npnts = loadintvar(nval, realval, 1, ivalout)
00273          IF (ivalout(1) > 0) THEN
00274              writeparams = .true.
00275          ELSE
00276              writeparams = .false.
00277          END IF
00278
00279      !----- CASE
00280      CASE ('NBTRFILES')
00281          npnts = loadintvar(nval, realval, 1, ivalout)
00282          nbtrfiles = ivalout(1)
00283          IF (nbtrfiles > 0) THEN
00284              ALLOCATE(besttrackfilename(nbtrfiles))
00285              besttrackfilename = blank
00286          END IF
00287          gotnbtrfiles = .true.
00288
00289      !----- CASE
00290      CASE ('BESTTRACKFILENAME')
00291          IF (.NOT. gotnbtrfiles) THEN
00292              WRITE(scratchmessage, fmtparnotfound) 'nBTrFiles', '(add the "nBTrFiles" keyword before
00293 "bestTrackFileName").'
00294              CALL allmessage(error, scratchmessage)
00295          ELSE
00296              IF (ALLOCATED(besttrackfilename)) THEN
00297                  tmpstr = adjustl(charval(nval))
00298                  IF (trim(tmpstr) == "") THEN
00299                      nval = nval - 1
00300                  ELSE
00301                      IF (nval <= nbtrfiles) THEN ! because bestTrackFileName has been allocated this way
00302                          numbtfiles = numbtfiles + 1
00303                          besttrackfilename(numbtfiles) = trim(tmpstr)
00304                          besttrackfilenamespecified = .true.
00305                      END IF
00306                  END IF
00307              END IF
00308
00309      !----- CASE
00310      CASE ('MESHFILETYPE')
00311          IF (nval == 1) THEN
00312              meshfiletype = trim(adjustl(charval(nval)))
00313          ELSE
00314              IF (trim(adjustl(meshfiletype)) == "") THEN
00315                  meshfiletype = trim(adjustl(charval(nval)))
00316              END IF
00317          END IF
00318
00319      !----- CASE
00320      CASE ('MESHFILENAME')
00321          IF (nval == 1) THEN
00322              meshfilename = trim(adjustl(charval(nval)))
00323          ELSE
00324              IF (trim(adjustl(meshfilename)) == "") THEN
00325                  meshfilename = trim(adjustl(charval(nval)))
00326              END IF
00327          END IF
00328          IF (trim(adjustl(meshfilename)) /= "") meshfilenamespecified = .true.
00329
00330      !----- CASE

```

```

00331      CASE ('MESHFILEFORM')
00332          IF (nval == 1) THEN
00333              meshfileform = trim(adjustl(charval(nval)))
00334          ELSE
00335              IF (trim(adjustl(meshfileform)) == "") THEN
00336                  meshfileform = trim(adjustl(charval(nval)))
00337              END IF
00338          END IF
00339
00340      !----- CASE
00341      CASE ('GRAVITY')
00342          npnts = loadrealvar(nval, realval, 1, rvalout)
00343          gravity = rvalout(1)
00344
00345      !----- CASE
00346      CASE ('RHOWATER')
00347          npnts = loadrealvar(nval, realval, 1, rvalout)
00348          rhewater = rvalout(1)
00349
00350      !----- CASE
00351      CASE ('RHOAIR')
00352          npnts = loadrealvar(nval, realval, 1, rvalout)
00353          rhoair = rvalout(1)
00354
00355      !----- CASE
00356      CASE ('BACKGROUNDATMPRESS')
00357          npnts = loadrealvar(nval, realval, 1, rvalout)
00358          backgroundatmpress = rvalout(1)
00359
00360      !----- CASE
00361      CASE ('WINDREDUCTION')
00362          npnts = loadrealvar(nval, realval, 1, rvalout)
00363          windreduction = rvalout(1)
00364
00365      !----- CASE
00366      CASE ('REFDATETIME')
00367          IF (nval == 1) THEN
00368              refdatetime = trim(adjustl(charval(nval)))
00369          ELSE
00370              IF (trim(adjustl(refdatetime)) == "") THEN
00371                  refdatetime = trim(adjustl(charval(nval)))
00372              END IF
00373          END IF
00374
00375          CALL splittimetimestring(refdatetime, refyear, refmonth, refday, refhour, refmin, refsec)
00376
00377          IF ((refyear == imissv) .OR. (refmonth <= 0) .OR. (refday <= 0)) THEN
00378              refdate = imissv
00379          ELSE
00380              refdate = joindate(refyear, refmonth, refday)
00381          END IF
00382          reftime = joindate(refhour, refmin, refsec)
00383          refdatespecified = .true.
00384
00385      !----- CASE
00386      CASE ('UNITTIME')
00387          IF (begdatespecified .OR. begsimsspecified .OR.
00388              enddatespecified .OR. endsimsspecified) THEN
00389              scratchmessage = 'add the "unitTime" keyword before the ' // &
00390                  '"begDateTime"/"begTime" and "endDateTime"/"endTime" keywords'
00391              CALL allmessage(error, scratchmessage)
00392              CALL terminate()
00393
00394          ELSE
00395              IF (nval == 1) THEN
00396                  tmpcharval = touppercase(trim(adjustl(charval(nval))))
00397                  unittime = tmpcharval(1:1)
00398              ELSE
00399                  IF (trim(adjustl(unittime)) == "") THEN
00400                      tmpcharval = touppercase(trim(adjustl(charval(nval))))
00401                      unittime = tmpcharval(1:1)
00402                  END IF
00403              END IF
00404
00405      !----- CASE
00406      CASE ('BEGDATETIME')
00407          IF (begdatespecified .OR. begsimsspecified) THEN
00408              scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
00409              CALL allmessage(error, scratchmessage)
00410
00411          begdatespecified = .false.

```

```

00412      ELSE
00413          IF (.NOT. refdatespecified) THEN
00414              scratchmessage = 'Add the "refDateTime" keyword before "begDateTime".'
00415              CALL allmessage(error, scratchmessage)
00416      ELSE
00417          IF (nval == 1) THEN
00418              begdatetime = trim(adjustl(charval(nval)))
00419      ELSE
00420          IF (trim(adjustl(begdatetime)) == "") THEN
00421              begdatetime = trim(adjustl(charval(nval)))
00422          END IF
00423      END IF
00424
00425      CALL splitdatetimestring(begdatetime, begyear, begmonth, begday, beghour, begmin, begsec)
00426
00427      IF ((begyear == imissv) .OR. (begmonth <= 0) .OR. (begday <= 0)) THEN
00428          begdate = imissv
00429      ELSE
00430          begdate = joindate(begyear, begmonth, begday)
00431      END IF
00432      begtime = joindate(beghour, begmin, begsec)
00433
00434      CALL timeconv(begyear, begmonth, begday, beghour, begmin, begsec, mdbegsimtime)
00435      begsimtime = mdbegsimtime * gettimeconvsec(unittime, 1)
00436
00437      begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)
00438
00439      begdatespecified = .true.
00440      begsimspecified = .true.
00441      END IF
00442  END IF
00443
00444 !----- CASE
00445 CASE ('ENDDATETIME')
00446     IF (enddatespecified .OR. endsimspecified) THEN
00447         scratchmessage = 'Only one of "endDateTime" or "endSimTime" can be specified'
00448         CALL allmessage(error, scratchmessage)
00449
00450         enddatespecified = .false.
00451     ELSE
00452         IF (.NOT. refdatespecified) THEN
00453             scratchmessage = 'Add the "refDateTime" keyword before "endDateTime".'
00454             CALL allmessage(error, scratchmessage)
00455     ELSE
00456         IF (nval == 1) THEN
00457             enddatetime = trim(adjustl(charval(nval)))
00458         ELSE
00459             IF (trim(adjustl(enddatetime)) == "") THEN
00460                 enddatetime = trim(adjustl(charval(nval)))
00461             END IF
00462         END IF
00463
00464         CALL splitdatetimestring(enddatetime, endyear, endmonth, endday, endhour, endmin, endsec)
00465
00466         IF ((endyear == imissv) .OR. (endmonth <= 0) .OR. (endday <= 0)) THEN
00467             enddate = imissv
00468         ELSE
00469             enddate = joindate(endyear, endmonth, endday)
00470         END IF
00471         endtime = joindate(endhour, endmin, endsec)
00472
00473         CALL timeconv(endyear, endmonth, endday, endhour, endmin, endsec, mdendsimtime)
00474         endsimtime = mdendsimtime * gettimeconvsec(unittime, 1)
00475
00476         enddatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)
00477
00478         enddatespecified = .true.
00479         endsimspecified = .true.
00480     END IF
00481  END IF
00482
00483 !----- CASE
00484 CASE ('OUTDT')
00485     npnts = loadrealvar(nval, realval, 1, rvalout)
00486     outdt = rvalout(1)
00487     mdoutdt = fixnearwholereal(outdt * gettimeconvsec(unittime), closetol)
00488
00489 !----- CASE
00490 CASE ('BEGSIMTIME')
00491     IF (begdatespecified .OR. begsimspecified) THEN
00492         scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'

```

```

00493         CALL allmessage(error, scratchmessage)
00494
00495         begsimspecified = .false.
00496     ELSE
00497         IF (.NOT. refdatespecified) THEN
00498             scratchmessage = 'Add the "refDateTime" keyword before "begSimTime").'
00499             CALL allmessage(error, scratchmessage)
00500     ELSE
00501         npnts = loadrealvar(nval, realval, 1, rvalout)
00502         begsimtime = rvalout(1)
00503
00504         mdbegsimtime = begsimtime * gettimeconvsec(unittime)
00505
00506         jday = (mdbegsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
00507         refhour, refmin, refsec)
00508             CALL juldaytoreg(jday, begyear, begmonth, begday, beghour, begmin, begsec)
00509             begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)
00510
00511         begdatespecified = .true.
00512         begsimspecified = .true.
00513     END IF
00514 END IF
00515
00516 !----- CASE
00517 CASE ('ENDSIMTIME')
00518     IF (enddatespecified .OR. endsimspecified) THEN
00519         scratchmessage = 'Only one of "endDateTime" and "endSimTime" can be specified'
00520         CALL allmessage(error, scratchmessage)
00521
00522     endsimspecified = .false.
00523 ELSE
00524     IF (.NOT. refdatespecified) THEN
00525         scratchmessage = 'Add the "refDateTime" keyword before "endSimTime").'
00526         CALL allmessage(error, scratchmessage)
00527 ELSE
00528     npnts = loadrealvar(nval, realval, 1, rvalout)
00529     endsimtime = rvalout(1)
00530
00531     mdendsimtime = endsimtime * gettimeconvsec(unittime)
00532
00533     jday = (mdendsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
00534     refhour, refmin, refsec)
00535             CALL juldaytoreg(jday, endyear, endmonth, endday, endhour, endmin, endsec)
00536             begdatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)
00537
00538         enddatespecified = .true.
00539         endsimspecified = .true.
00540     END IF
00541 END IF
00542
00543 !----- CASE
00544 CASE ('OUTFILENAME')
00545     IF (nval == 1) THEN
00546         outfilename = trim(adjustl(charval(nval)))
00547     ELSE
00548         IF (trim(adjustl(outfilename)) == "") THEN
00549             outfilename = trim(adjustl(charval(nval)))
00550         END IF
00551     IF (trim(adjustl(outfilename)) /= "") outfilenamespecified = .true.
00552
00553 !----- CASE
00554 CASE ('NCVARNAM_PRES')
00555     IF (nval == 1) THEN
00556         ncvarnam_pres = trim(adjustl(charval(nval)))
00557     ELSE
00558         IF (trim(adjustl(ncvarnam_pres)) == "") THEN
00559             ncvarnam_pres = trim(adjustl(charval(nval)))
00560         END IF
00561 END IF
00562
00563 !----- CASE
00564 CASE ('NCVARNAM_WNDX')
00565     IF (nval == 1) THEN
00566         ncvarnam_wndx = trim(adjustl(charval(nval)))
00567     ELSE
00568         IF (trim(adjustl(ncvarnam_wndx)) == "") THEN
00569             ncvarnam_wndx = trim(adjustl(charval(nval)))
00570         END IF
00571 END IF

```

```

00572      !----- CASE
00573      CASE ('NCVARNAM_WNDY')
00574          IF (nval == 1) THEN
00575              ncvarnam_wndy = trim(adjustl(charval(nval)))
00576          ELSE
00577              IF (trim(adjustl(ncvnam_wndy)) == "") THEN
00578                  ncvarnam_wndy = trim(adjustl(charval(nval)))
00579              END IF
00580          END IF
00581
00582      !----- CASE
00583      CASE ('NCSHUFFLE')
00584          npnts = loadintvar(nval, realval, 1, ivalout)
00585          ncshuffle = ivalout(1)
00586
00587      !----- CASE
00588      CASE ('NCDEFLATE')
00589          npnts = loadintvar(nval, realval, 1, ivalout)
00590          ncdeflate = ivalout(1)
00591
00592      !----- CASE
00593      CASE ('NCLEVEL')
00594          npnts = loadintvar(nval, realval, 1, ivalout)
00595          ncdlevel = ivalout(1)
00596
00597      !----- CASE
00598      CASE ('MODELTYPE')
00599          npnts = loadintvar(nval, realval, 1, ivalout)
00600          modeltype = ivalout(1)
00601
00602      !----- CASE
00603      CASE DEFAULT
00604          ! Do nothing
00605      END SELECT
00606  END IF
00607 END DO
00608
00609 10 WRITE(lun_screen, '("Error while processing line: ", a, " in file: ", a)') &
00610     trim(adjustl(inpline)), trim(tmpfilename)
00611
00612 CLOSE(iunit)
00613 stop
00614
00615 20 CLOSE(iunit)
00616
00617 WRITE(lun_screen, '(a)') 'Finished processing the input fields from the control file ...'
00618
00619 !-----
00620 !--- CHECK INPUT VARIABLES AND SET DEFAULTS
00621 !-----
00622 CALL initlogging()
00623
00624 IF (checkcontrolfileinputs() /= 0) THEN
00625     WRITE(scratchmessage, '(a)') &
00626         'Errors found while processing the input variables. Check the log file for details.'
00627     CALL screenmessage(error, scratchmessage)
00628     CALL unsetmessagesource()
00629     CALL terminate()
00630 END IF
00631
00632 CALL printmodelparams()
00633
00634 CALL unsetmessagesource()
00635
00636 END SUBROUTINE readcontrolfile
00637
00638 !=====
00639
00640 !-----
00641 !      S U B R O U T I N E   P R I N T   M O D E L   P A R A M S
00642 !-----
00643
00644 !-----
00645 !----- SUBROUTINE printmodelparams()
00646
00647 USE pahm_global
00648
00649 IMPLICIT NONE
00650
00651 CHARACTER(LEN=128) :: tmpStr
00652 INTEGER :: i
00653
00654
00655
00656
00657
00658
00659

```

```

00660    IF (writeparams) THEN
00661        print *, "
00662        WRITE(*, '(a)')      '----- MODEL PARAMETERS -----'
00663
00664        WRITE(*, '(a, a)')      ' title                  = ', trim(adjustl(title))
00665
00666        DO i = 1, nbtrfiles
00667            WRITE(*, '(a, "", il, "", a)')      ' bestTrackFileName', i, " = " //
00668            trim(adjustl(besttrackfilename(i)))
00669        END DO
00670        WRITE(*, '(a, a)')      ' meshFileType          = ', trim(adjustl(meshfiletype))
00671        WRITE(*, '(a, a)')      ' meshFileName           = ', trim(adjustl(meshfilename))
00672        WRITE(*, '(a, a)')      ' meshFileForm           = ', trim(adjustl(meshfileform))
00673
00674        print *, "
00675        WRITE(tmpstr, '(f20.5)') gravity
00676        WRITE(*, '(a, a)')      ' gravity                = ', trim(adjustl(tmpstr)) // " m/s^2"
00677        WRITE(tmpstr, '(f20.5)') rhoWater
00678        WRITE(*, '(a, a)')      ' rhoWater               = ', trim(adjustl(tmpstr)) // " kg/m^3"
00679        WRITE(tmpstr, '(f20.5)') rhoAir
00680        WRITE(*, '(a, a)')      ' rhoAir                 = ', trim(adjustl(tmpstr)) // " kg/m^3"
00681        WRITE(tmpstr, '(f20.5)') backgroundAtmPress
00682        WRITE(*, '(a, a)')      ' backgroundAtmPress   = ', trim(adjustl(tmpstr)) // " mbar"
00683        WRITE(tmpstr, '(f20.2)') windreduction
00684        WRITE(*, '(a, a)')      ' windReduction         = ', trim(adjustl(tmpstr))
00685
00686        print *, "
00687        WRITE(*, '(a, a)')      ' refDateTime           = ', trim(adjustl(refdatetime))
00688        WRITE(*, '(a, i4.4)')    ' refYear                = ', refyear
00689        WRITE(*, '(a, i2.2)')    ' refMonth               = ', refmonth
00690        WRITE(*, '(a, i2.2)')    ' refDay                 = ', refday
00691        WRITE(*, '(a, i2.2)')    ' refHour                = ', refhour
00692        WRITE(*, '(a, i2.2)')    ' refMin                 = ', refmin
00693        WRITE(*, '(a, i2.2)')    ' refSec                 = ', refsec
00694        WRITE(*, '(a, 11)')     ' refDateSpecified      = ', refdatespecified
00695
00696        print *, "
00697        WRITE(*, '(a, a)')      ' begDateTime           = ', trim(adjustl(begdatetime))
00698        WRITE(*, '(a, i4.4)')    ' begYear                = ', begyear
00699        WRITE(*, '(a, i2.2)')    ' begMonth               = ', begmonth
00700        WRITE(*, '(a, i2.2)')    ' begDay                 = ', begday
00701        WRITE(*, '(a, i2.2)')    ' begHour                = ', beghour
00702        WRITE(*, '(a, i2.2)')    ' begMin                 = ', beginmin
00703        WRITE(*, '(a, i2.2)')    ' begSec                 = ', begsec
00704        WRITE(*, '(a, 11)')     ' begDateSpecified      = ', begdatespecified
00705
00706        print *, "
00707        WRITE(*, '(a, a)')      ' endDateTime           = ', trim(adjustl(enddatetime))
00708        WRITE(*, '(a, i4.4)')    ' endYear                = ', endyear
00709        WRITE(*, '(a, i2.2)')    ' endMonth               = ', endmonth
00710        WRITE(*, '(a, i2.2)')    ' endDay                 = ', endday
00711        WRITE(*, '(a, i2.2)')    ' endHour                = ', endhour
00712        WRITE(*, '(a, i2.2)')    ' endMin                 = ', endmin
00713        WRITE(*, '(a, i2.2)')    ' endSec                 = ', endsec
00714        WRITE(*, '(a, 11)')     ' endDateSpecified      = ', enddatespecified
00715
00716        print *, "
00717        WRITE(*, '(a, a1)')    ' unitTime               = ', unittime
00718        WRITE(tmpstr, '(f20.5)') outdt
00719        WRITE(*, '(a, a)')      ' outDT
00720        tolowercase(trim(unittime))
00721        WRITE(tmpstr, '(f20.5)') mdoutdt
00722        WRITE(*, '(a, a)')      ' mdOutDT
00723        WRITE(tmpstr, '(f20.5)') begsimtime
00724        WRITE(*, '(a, a)')      ' begSimTime
00725        WRITE(*, '(a, 11)')     ' begSimSpecified
00726        WRITE(tmpstr, '(f20.5)') endsimtime
00727        WRITE(*, '(a, a)')      ' endSimTime
00728        tolowercase(trim(unittime))
00729        WRITE(tmpstr, '(f20.5)') mdendsimtime
00730        WRITE(*, '(a, a)')      ' mdEndSimTime
00731        WRITE(*, '(a, 11)')     ' endSimSpecified
00732        WRITE(tmpstr, '(i10)')  noutdt
00733        WRITE(*, '(a, a)')      ' nOutDT
00734
00735        print *, "
00736        WRITE(*, '(a, a)')      ' outFileName             = ', trim(adjustl(outfilename))
00737        WRITE(*, '(a, il)')     ' ncShuffle
00738

```

```

00737      WRITE(*, '(a, il)')      , ncDeflate      = ', ncdeflate
00738      WRITE(*, '(a, il)')      , ncDLevel       = ', nclevel
00739      WRITE(*, '(a, a)')       , ncVarNam_Pres   = ', trim(ncvarnam_pres)
00740      WRITE(*, '(a, a)')       , ncVarNam_WndX   = ', trim(ncvarnam_wndx)
00741      WRITE(*, '(a, a)')       , ncVarNam_WndY   = ', trim(ncvarnam_wndy)
00742
00743      print *, "
00744      WRITE(*, '(a, il)')      , modelType      = ', modeltype
00745
00746      WRITE(*, '(a)')          '----- MODEL PARAMETERS -----'
00747      print *, "
00748  END IF
00749
00750 END SUBROUTINE printmodelparams
00751
00752 !=====
00753
00754 !-----
00755 ! F U N C T I O N  G E T  L I N E  R E C O R D
00756 !-----
00779 !
00780 INTEGER FUNCTION getlinerecord(inpLine, outLine, lastCommFlag) RESULT(myLen)
00781
00782 IMPLICIT NONE
00783
00784 ! Imported variable declarations.
00785 CHARACTER(LEN=*) , INTENT(IN)           :: inpline
00786 CHARACTER(LEN=LEN(inpLine)), INTENT(OUT)  :: outline
00787 INTEGER, OPTIONAL                      :: lastcommflag
00788
00789 ! Local variable declarations.
00790 CHARACTER(LEN=LEN(inpLine))             :: line
00791 CHARACTER                           :: tmpinpline(len(inpLine))
00792 INTEGER                               :: lenline, commflag, icomm
00793
00794 ! Table of some ASCII character symbols
00795 !  CHAR     ASCII
00796 !  TAB      9
00797 !  SPC      32
00798 !  !        33
00799 !  #        35
00800 !  *        42
00801 !  +        43
00802 !  -        45
00803 !  /        47
00804 !  :        58
00805 !  =        61
00806 !  \        92
00807 !  |        124
00808
00809 mylen      = 0
00810 outline    = blank
00811 tmpinpline = blank
00812
00813 commflag = 0
00814 IF (PRESENT(lastcommflag)) THEN
00815   IF (lastcommflag <= 0) commflag = 0
00816   IF (lastcommflag > 0) commflag = 1
00817 END IF
00818
00819 tmpinpline = transfer(inpLine, tmpinpline)
00820 tmpinpline = pack(tmpinpline, tmpinpline /= achar(9), spread(' ', 1, len(inpLine)))
00821
00822 line       = trim(adjustl(transfer(tmpinpline, line)))
00823 lenline    = len_trim(line)
00824
00825 IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
00826   IF (commflag > 0) THEN
00827     icomm = index(line, char(33), back = .false.)
00828     IF (icomm == 0) icomm = index(line, char(35), back = .false.)
00829     IF (icomm > 0) lenline = icomm - 1
00830
00831   line = trim(adjustl(line(1:lenline)))
00832   lenline = len_trim(line)
00833 END IF
00834
00835 outline = line
00836 mylen   = lenline
00837 END IF
00838
00839 RETURN

```

```

00840
00841      END FUNCTION getlinerecord
00842
00843 !=====
00844 !-----+
00845 !-----+ F U N C T I O N   P A R S E   L I N E
00846 !-----+
00847 !-----+
00848 INTEGER FUNCTION parseline(inpLine, outLine, keyWord, nVal, cVal, rVal) RESULT(myStatus)
00849
00850     IMPLICIT NONE
00851
00852     ! Imported variable declarations.
00853     CHARACTER(LEN=*) , INTENT(IN)          :: inpline
00854     CHARACTER(LEN=LEN(inpLine)) , INTENT(OUT) :: outline
00855     CHARACTER(LEN=40) , INTENT(INOUT)        :: keyword
00856     INTEGER , INTENT(INOUT)                 :: nval
00857     CHARACTER(LEN=512) , DIMENSION(200) , INTENT(INOUT) :: cval
00858     REAL(sz) , DIMENSION(200) , INTENT(INOUT) :: rval
00859
00860     ! Local variable declarations
00861     LOGICAL                                :: issstring, kextract, decflag, nested
00862     INTEGER                                 :: iblank, icont, ipipe, kstr, kend
00863     INTEGER                                 :: lend, lens, lstr, lval, nmul, schar
00864     INTEGER                                 :: copies, i, ic, ie, ierr, is, j, status
00865     INTEGER, DIMENSION(20)                   :: imul
00866     CHARACTER(LEN=256)                      :: vstring, string
00867     CHARACTER(LEN=LEN(inpLine))             :: line
00868     INTEGER                                 :: lenline
00869
00870     ! Table of some ASCII character symbols
00871     !   CHAR      ASCII
00872     !   TAB       9
00873     !   SPC      32
00874     !   !       33
00875     !   #       35
00876     !   *       42
00877     !   +       43
00878     !   -       45
00879     !   /       47
00880     !   :       58
00881     !   =       61
00882     !   \       92
00883     !   |       124
00884
00885     ! Initialize.
00886     line      = blank
00887     vstring   = blank
00888     string    = blank
00889
00890     lenline = getlinerecord(inpline, line, 1)
00891     outline = line
00892
00893     ! If not a blank or comment line [CHAR(33)!=], decode and extract input
00894     ! values. Find equal sign [CHAR(61)].
00895     status = -1
00896     nested = .false.
00897     IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
00898         status = 1
00899         kstr = 1
01000         kend = index(line, char(61), back = .false.) - 1
01001         lstr = index(line, char(61), back = .true.) + 1
01002
01003         ! Determine if KEYWORD is followed by double equal sign (==) indicating
01004         ! nested parameter.
01005         IF ((lstr - kend) == 3) nested = .true.
01006
01007         ! Extract KEYWORD, trim leading and trailing blanks.
01008         kextract = .false.
01009         IF (kend > 0) THEN
01010             lend = lenline
01011             keyword = line(kstr:kend)
01012             nval = 0
01013             kextract = .true.
01014
01015         ELSE
01016             lstr = 1
01017             lend = lenline
01018             kextract = .true.
01019
01020         END IF
01021

```

```

00948 ! Extract parameter values string. Remove continuation symbol
00949 ! [CHAR(92)=\] or multi-line value [CHAR(124)=|], if any. Trim
00950 ! leading trailing blanks.
00951 IF (kextract) THEN
00952   icont = index(line, char(92 ), back = .false.)
00953   ipipe = index(line, char(124), back = .false.)
00954   IF (icont > 0) lend = icont - 1
00955   IF (ipipe > 0) lend = ipipe - 1
00956   vstring = adjustl(line(lstr:lend))
00957   lval = len_trim(vstring)
00958
00959 ! The PROGRAM, VERSION and TITLE KEYWORDS are special ones because
00960 ! they can include strings, numbers, spaces, and continuation symbol.
00961 isstring = .false.
00962 SELECT CASE (touppercase(trim(keyword)))
00963   CASE ('TITLE')
00964     nval = nval + 1
00965     cval(nval) = vstring(1:lval)
00966     isstring = .true.
00967
00968   CASE ('LOGFILENAME')
00969     nval = nval + 1
00970     cval(nval) = vstring(1:lval)
00971     isstring = .true.
00972
00973   CASE ('BESTTRACKFILENAME')
00974     nval = nval + 1
00975     cval(nval) = vstring(1:lval)
00976     isstring = .true.
00977
00978   CASE ('MESHFILENAME')
00979     nval = nval + 1
00980     cval(nval) = vstring(1:lval)
00981     isstring = .true.
00982
00983   CASE ('MESHFILETYPE')
00984     nval = nval + 1
00985     cval(nval) = vstring(1:lval)
00986     isstring = .true.
00987
00988   CASE ('MESHFILEFORM')
00989     nval = nval + 1
00990     cval(nval) = vstring(1:lval)
00991     isstring = .true.
00992
00993   CASE ('REFDATETIME')
00994     nval = nval + 1
00995     cval(nval) = vstring(1:lval)
00996     isstring = .true.
00997
00998   CASE ('UNITTIME')
00999     nval = nval + 1
01000     cval(nval) = vstring(1:lval)
01001     isstring = .true.
01002
01003   CASE ('BEGDATETIME')
01004     nval = nval + 1
01005     cval(nval) = vstring(1:lval)
01006     isstring = .true.
01007
01008   CASE ('ENDDATETIME')
01009     nval = nval + 1
01010     cval(nval) = vstring(1:lval)
01011     isstring = .true.
01012
01013   CASE ('OUTFILENAME')
01014     nval = nval + 1
01015     cval(nval) = vstring(1:lval)
01016     isstring = .true.
01017
01018   CASE ('NCVARNAM_PRES')
01019     nval = nval + 1
01020     cval(nval) = vstring(1:lval)
01021     isstring = .true.
01022
01023   CASE ('NCVARNAM_WNDX')
01024     nval = nval + 1
01025     cval(nval) = vstring(1:lval)
01026     isstring = .true.
01027
01028   CASE ('NCVARNAM_WNDY')

```

```

01029      nval = nval + 1
01030      cval(nval) = vstring(1:lval)
01031      isstring = .true.
01032
01033      CASE DEFAULT
01034          ! For every other KEYWORD except the above.
01035          ! Check if there is a multiplication symbol [CHAR(42)=*] in the variable
01036          ! string indicating repetition of input values.
01037          nmul = 0
01038          DO i = 1, lval
01039              IF (vstring(i:i) == char(42)) THEN
01040                  nmul = nmul + 1
01041                  imul(nmul) = i
01042              END IF
01043          END DO
01044          ic = 1
01045
01046          ! Check for blank spaces [CHAR(32)=' '] between entries and decode.
01047          is = 1
01048          ie = lval
01049          iblank = 0
01050          decflag = .false.
01051          DO i = 1, lval
01052              IF (vstring(i:i) == char(32)) THEN
01053                  IF (vstring(i + 1:i + 1) /= char(32)) decflag = .true.
01054                  iblank = i
01055              ELSE
01056                  ie = i
01057              END IF
01058              IF (decflag .OR. (i == lval)) THEN
01059                  nval = nval + 1
01060
01061          ! Processing numeric values. Check starting character to determine
01062          ! if numeric or character values. It is possible to have both when
01063          ! processing repetitions via the multiplication symbol.
01064          schar = ichar(vstring(is:is))
01065          IF (((48 <= schar) .AND. (schar <= 57)) .OR. (schar == 43) .OR. (schar == 45)) THEN
01066              IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
01067                  READ(vstring(is:imul(ic) - 1), *) copies
01068                  schar = ichar(vstring(imul(ic) + 1:imul(ic) + 1))
01069                  IF ((43 <= schar) .AND. (schar <= 57)) THEN
01070                      READ(vstring(imul(ic) + 1:ie), *) rval(nval)
01071                      DO j = 1, copies - 1
01072                          rval(nval + j) = rval(nval)
01073                      END DO
01074                  ELSE
01075                      string = vstring(imul(ic) + 1:ie)
01076                      lens = len_trim(string)
01077                      cval(nval) = string(1:lens)
01078                      DO j = 1, copies - 1
01079                          cval(nval + j) = cval(nval)
01080                      END DO
01081                  END IF
01082                  nval = nval + copies - 1
01083                  ic = ic + 1
01084              ELSE
01085                  string = vstring(is:ie)
01086                  lens = len_trim(string)
01087                  !READ(string(1:lens), *) rVal(nVal)
01088                  READ(string(1:lens), *, iostat=ierr) rval(nval)
01089                  IF (ierr /= 0) THEN
01090                      WRITE(*, *) '#### ERROR :: Cannot interpret string ', string(1:lens), &
01091                           ' as a REAL number.'
01092                  END IF
01093              END IF
01094          ELSE
01095
01096          ! Processing character values (logicals and strings).
01097          IF ((nmul > 0) .AND. (is < imul(ic)) .AND. (imul(ic) < ie)) THEN
01098              READ(vstring(is:imul(ic) - 1), *) copies
01099              cval(nval) = vstring(imul(ic) + 1:ie)
01100              DO j = 1, copies - 1
01101                  cval(nval + j) = cval(nval)
01102              END DO
01103              nval = nval + copies - 1
01104              ic = ic + 1
01105          ELSE
01106              string = vstring(is:ie)
01107              cval(nval) = trim(adjustl(string))
01108          END IF
01109          isstring = .true.

```

```

01110          END IF
01111          is = iblank + 1
01112          ie = lval
01113          decflag = .false.
01114          END IF
01115          END DO
01116          END SELECT ! keyWord
01117          END IF ! kExtract
01118          status = nval
01119          END IF
01120
01121          mystatus = status
01122
01123          RETURN
01124
01125      END FUNCTION parseline
01126
01127 !=====
01128
01129 !-----+
01130 !     S U B R O U T I N E   C H E C K   C O N T R O L   F I L E   I N P U T S
01131 !-----+
01132
01133
01134
01135      INTEGER FUNCTION checkcontrolfileinputs() RESULT(errStatus)
01136
01137      USE pahm_global
01138      USE timedateutils, ONLY : firstgregdate, firstgreetime, &
01139                                gregtojulday, joindate, gettimeconvsec
01140
01141      IMPLICIT NONE
01142
01143      ! Local variables
01144      INTEGER             :: errio, errnum
01145      INTEGER             :: icnt
01146      LOGICAL             :: filefound
01147      REAL(sz)           :: gregjd, refjd, jd0, jd1
01148      REAL(sz)           :: timesec
01149      CHARACTER(LEN=64)   :: tmpstr, tmpstr1, tmpstr2
01150
01151
01152
01153
01154
01155      !----- Initialize variables
01156      errio      = 0
01157      errnum     = 0
01158      errstatus   = 0
01159      scratchmessage = blank
01160
01161
01162
01163
01164
01165      CALL setmessagesource("CheckControlFileInputs")
01166
01167      !---- 1) Best track files (mandatory variables) ----
01168      IF (nbtrfiles <= 0) THEN
01169          errnum = 1
01170
01171      WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01172                               '. Invalid value supplied for dimension parameter: nBTrFiles = ', &
01173                               nbtrfiles, ' (should be greater than zero).'
01174      CALL logmessage(error, scratchmessage)
01175      ELSE IF (besttrackfilenamespecified) THEN
01176          IF (numbtfiles /= nbtrfiles) THEN
01177              errnum = 2
01178
01179          WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01180                               '. The number of files for <bestTrackFileName> should be equal to nBTrFiles:
01181
01182          ', &
01183          nbtrfiles, '.'
01184          CALL logmessage(error, scratchmessage)
01185      END IF
01186
01187      DO icnt = 1, numbtfiles
01188          INQUIRE(file=trim(adjustl(besttrackfilename(icnt))), iostat=errio, exist=filefound)
01189          IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01190              errnum = 3
01191
01192          WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01193                               '. Could not access the best track file for read: ' // &
01194                               trim(adjustl(besttrackfilename(icnt)))
01195          CALL logmessage(error, scratchmessage)
01196      END IF
01197      END DO
01198
01199      ELSE
01200          errnum = 4
01201
01202
01203

```

```

01204
01205     WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01206     ' . bestTrackFileName(s) not specified. This is a mandatory variable. '
01207     CALL logmessage(error, scratchmessage)
01208 END IF
01209
01210 !---- 2) Mesh file (mandatory variables) -----
01211 IF (.NOT. meshfilenamespecified) THEN
01212     errnum = 5
01213
01214     WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01215     ' . Parameter <meshFileName> is not specified (mandatory variable) '
01216     CALL logmessage(error, scratchmessage)
01217 ELSE
01218     meshfilename = adjustl(meshfilename)
01219     INQUIRE(file=trim(meshfilename), iostat=errio, exist=filefound)
01220     IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01221         errnum = 6
01222
01223     WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01224     ' . Could not access the mesh file for read: ' // trim(meshfilename)
01225     CALL logmessage(error, scratchmessage)
01226 END IF
01227 END IF
01228
01229 ! Check for meshFileType
01230 meshfiletype = toupper(trim(adjustl(meshfiletype)))
01231 SELECT CASE (trim(meshfiletype))
01232     !CASE ('ADCIRC', 'SCHISM', 'FVCOM', 'ROMS', 'GENERIC')
01233     CASE ('ADCIRC', 'SCHISM')
01234         ! These are the valid values
01235
01236     CASE DEFAULT
01237         WRITE(scratchmessage, '(a)') 'This file type is not supported: meshFileType = ' //
01238         trim(meshfiletype)
01239         CALL logmessage(info, scratchmessage)
01240
01241     meshfiletype = 'ADCIRC'
01242
01243     WRITE(scratchmessage, '(a)') 'This value of meshFileType is adjusted to: meshFileType = ' //
01244     trim(meshfiletype)
01245     CALL logmessage(info, scratchmessage)
01246 END SELECT
01247
01248 ! Check for meshFileForm
01249 meshfileform = toupper(trim(adjustl(meshfileform)))
01250 SELECT CASE (trim(meshfileform))
01251     !CASE ('ASCII', 'NETCDF')
01252     CASE ('ASCII')
01253         ! These are valid values
01254
01255     CASE DEFAULT
01256         WRITE(scratchmessage, '(a)') 'This file format is not supported: meshFileForm = ' //
01257         trim(meshfileform)
01258         CALL logmessage(info, scratchmessage)
01259
01260     meshfileform = 'ASCII'
01261
01262     WRITE(scratchmessage, '(a)') 'This value of meshFileForm is adjusted to: meshFileForm = ' //
01263     trim(meshfileform)
01264     CALL logmessage(info, scratchmessage)
01265 END SELECT
01266
01267 !---- 3) Reference date and time (mandatory variables) -----
01268 gregjd = gregtojulday(firstgregdate, firstgregtime)
01269 refjd = gregtojulday(refdate, reftime)
01270 IF (refjd < gregjd) THEN
01271     errnum = 7
01272     WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01273     ' . Invalid DateTime string was supplied: refDateTime = ' // trim(refdatetime)
01274     CALL logmessage(error, scratchmessage)
01275 END IF
01276
01277 !---- 4) Stepping parameters (mandatory variables) -----
01278 ! check for valid start time
01279 IF (begsimspecified) THEN
01280     IF (refjd + (mdbegsimtime * gettimeconvsec('D', 1)) < gregjd) THEN
01281         errnum = 8
01282         WRITE(tmpstr, '(f20.5)') begsimtime
01283         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01284         ' . Invalid start time in reference to refDateTime was supplied: begSimTime =

```

```

' // &
01281           trim(adjustl(tmpstr))
01282     CALL logmessage(error, scratchmessage)
01283   END IF
01284 ELSE
01285   errnum = 81
01286   WRITE(scratchmessage, '("errNum = ", i0, a') errnum, &
01287         '. Neither "begDateTime" or "begSimTime" are defined properly'
01288   CALL logmessage(error, scratchmessage)
01289 END IF
01290
01291 ! check for valid stop time
01292 IF (endsimspecified) THEN
01293   IF (comparereals(endsimtime, begsimtime, closetol) <= 0) THEN
01294     errnum = 9
01295     WRITE(tmpstr1, '(f20.5)') begsimtime
01296     WRITE(tmpstr2, '(f20.5)') endsimtime
01297     WRITE(scratchmessage, '("errNum = ", i0, a') errnum, &
01298           '. Stop time should be greater than start time: begSimTime = ' // &
01299           trim(adjustl(tmpstr1)) // ', endSimTime = ' // trim(adjustl(tmpstr2))
01300   CALL logmessage(error, scratchmessage)
01301 END IF
01302 ELSE
01303   errnum = 91
01304   WRITE(scratchmessage, '("errNum = ", i0, a') errnum, &
01305         '. Neither "endDateTime" or "endSimTime" are defined properly'
01306   CALL logmessage(error, scratchmessage)
01307 END IF
01308
01309 ! check for valid outDT; (endSimTime - begSimTime) should be an integral integral multiple of outDT
01310 IF (outdt <= 0) THEN
01311   WRITE(tmpstr, '(f20.5)') outdt
01312   WRITE(scratchmessage, '(a)') 'Frequency of output data should be greater than zero: outDT = ' // &
01313           trim(adjustl(tmpstr))
01314   CALL logmessage(info, scratchmessage)
01315
01316 mdoutdt = 3600.0
01317 outdt = fixnearwholereal(mdoutdt * gettimeconvsec(unittime, 1), closetol)
01318
01319 WRITE(tmpstr, '(f20.5)') outdt
01320 WRITE(scratchmessage, '(a)') 'The outDT value is adjusted to: outDT = ' // trim(adjustl(tmpstr))
01321 CALL logmessage(info, scratchmessage)
01322 END IF
01323
01324 jd0 = refjd + (mdbegsimtime * gettimeconvsec('D', 1))
01325 jdl = refjd + (mdendsimtime * gettimeconvsec('D', 1))
01326 timesec = fixnearwholereal((jdl - jd0) * gettimeconvsec('D'), closetol)
01327 IF ((timesec < mdoutdt) .OR. comparereals(modulo(timesec, mdoutdt), 0.0_sz) /= 0) THEN
01328   errnum = 10
01329
01330   WRITE(tmpstr1, '(f20.5)') timesec
01331   WRITE(tmpstr2, '(f20.5)') outdt
01332   WRITE(scratchmessage, '("errNum = ", i0, a') errnum, &
01333         '. The value of (endSimTime - begSimTime) = ' // trim(adjustl(tmpstr1)) // &
01334         ' should be an integral multiple of outDT = ' // trim(adjustl(tmpstr2))
01335   CALL logmessage(error, scratchmessage)
01336 ELSE
01337   noutdt = int((timesec / mdoutdt) + 1)
01338 END IF
01339
01340 !---- 4) outFileName (mandatory variable) ----
01341 outfilename = adjustl(outfilename)
01342 IF (.NOT. outfilenamespecified) THEN
01343   errnum = 11
01344
01345   WRITE(scratchmessage, '("errNum = ", i0, a') errnum, &
01346         '. Output filename is not specified: outFileName = ' // trim(outfilename)
01347   CALL logmessage(error, scratchmessage)
01348 END IF
01349
01350 !---- 5) NetCDF variables ncShuffle, ncDeflate, ncDLevel and others ----
01351 IF (ncshuffle <= 0) THEN
01352   ncshuffle = 0
01353 ELSE
01354   ncshuffle = 1
01355 END IF
01356
01357 IF (ncdeflate <= 0) THEN
01358   ncdeflate = 0
01359 ELSE
01360   ncdeflate = 1

```

```

01361      END IF
01362
01363      IF (ncdlevel <= 0) THEN
01364          ncdlevel = 0
01365      ELSE
01366          IF (ncdlevel > 9) ncdlevel = 9
01367      END IF
01368
01369      ncvarnam_pres = trim(adjustl(ncvarnam_pres))
01370      IF (len_trim(ncvarnam_pres) == 0) ncvarnam_pres = trim(adjustl(def_ncnam_pres))
01371      ncvarnam_wndx = trim(adjustl(ncvarnam_wndx))
01372      IF (len_trim(ncvarnam_wndx) == 0) ncvarnam_wndx = trim(adjustl(def_ncnam_wndx))
01373      ncvarnam_wndy = trim(adjustl(ncvarnam_wndy))
01374      IF (len_trim(ncvarnam_wndy) == 0) ncvarnam_wndy = trim(adjustl(def_ncnam_wndy))
01375
01376      !---- 5) modelType (mandatory variable) ----
01377      SELECT CASE (modeltype)
01378          !CASE (1, 2, 3, 4)
01379          CASE (1, 10)
01380              ! These are all valid values
01381
01382          CASE DEFAULT
01383              errnum = 12
01384
01385          WRITE(scratchmessage, ('(errNum = ', i0, a, i0)') errnum, &
01386                  '. This model type is not supported: modelType = ', modeltype
01387          CALL logmessage(error, scratchmessage)
01388      END SELECT
01389
01390      !---- 6) various physical parameters ----
01391      IF ((gravity < 9.76) .OR. (gravity > 9.83)) THEN
01392          WRITE(tmpstr1, '(f20.5, a)') gravity
01393          tmpstr1 = trim(tmpstr1) // ' m/s^2'
01394          WRITE(tmpstr2, '(f20.5, a)') defv_gravity
01395          tmpstr2 = trim(tmpstr2) // ' m/s^2'
01396          WRITE(scratchmessage, '(a)') 'The value of gravity = ' // trim(adjustl(tmpstr1)) // &
01397              ' is adjusted to: gravity = ' // trim(adjustl(tmpstr2))
01398
01399          CALL logmessage(info, scratchmessage)
01400
01401          gravity = defv_gravity
01402      END IF
01403
01404      IF ((rhowater < 992.0) .OR. (rhowater > 1029.0)) THEN
01405          WRITE(tmpstr1, '(f20.5, a)') rhowater
01406          tmpstr1 = trim(tmpstr1) // ' kg/m^3'
01407          WRITE(tmpstr2, '(f20.5, a)') defv_rhowater
01408          tmpstr2 = trim(tmpstr2) // ' kg/m^3'
01409          WRITE(scratchmessage, '(a)') 'The value of rhoWater = ' // trim(adjustl(tmpstr1)) // &
01410              ' is adjusted to: rhoWater = ' // trim(adjustl(tmpstr2))
01411
01412          CALL logmessage(info, scratchmessage)
01413
01414          rhowater = defv_rhowater
01415      END IF
01416
01417      IF ((rhoair < 1.0) .OR. (rhoair > 1.3)) THEN
01418          WRITE(tmpstr1, '(f20.5, a)') rhoair
01419          tmpstr1 = trim(tmpstr1) // ' kg/m^3'
01420          WRITE(tmpstr2, '(f20.5, a)') defv_rhoair
01421          tmpstr2 = trim(tmpstr2) // ' kg/m^3'
01422          WRITE(scratchmessage, '(a)') 'The value of rhoAir = ' // trim(adjustl(tmpstr1)) // &
01423              ' is adjusted to: rhoAir = ' // trim(adjustl(tmpstr2))
01424
01425          CALL logmessage(info, scratchmessage)
01426
01427          rhoair = defv_rhoair
01428      END IF
01429
01430      IF ((backgroundatmpress < 1000.0) .OR. (backgroundatmpress > 1025.0)) THEN
01431          WRITE(tmpstr1, '(f20.5, a)') backgroundatmpress
01432          tmpstr1 = trim(tmpstr1) // ' mb'
01433          WRITE(tmpstr2, '(f20.5, a)') defv_atmpress
01434          tmpstr2 = trim(tmpstr2) // ' mb'
01435          WRITE(scratchmessage, '(a)') 'The value of backgroundAtmPress = ' // trim(adjustl(tmpstr1)) // &
01436              ' is adjusted to: backgroundAtmPress = ' // trim(adjustl(tmpstr2))
01437
01438          CALL logmessage(info, scratchmessage)
01439
01440          backgroundatmpress = defv_atmpress
01441      END IF

```

```

01442
01443     IF ((windreduction < 0.65) .OR. (windreduction > 1.0)) THEN
01444         WRITE(tmpstr1, '(f20.5)') windreduction
01445         WRITE(tmpstr2, '(f20.5)') defv_windreduction
01446         WRITE(scratchmessage, '(a)') 'The value of windReduction = ' // trim(adjustl(tmpstr1)) // &
01447             ' is adjusted to: windReduction = ' // trim(adjustl(tmpstr2))
01448
01449         CALL logmessage(info, scratchmessage)
01450
01451         windreduction = defv_windreduction
01452     END IF
01453
01454     errstatus = errnum
01455
01456 END FUNCTION checkcontrolfileinputs
01457
01458 !=====
01459 !-----+
01460 ! F U N C T I O N   L O A D   I N T   V A R
01461 !-----+
01484 !
01485 INTEGER FUNCTION loadintvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01486
01487     IMPLICIT NONE
01488
01489     INTEGER, INTENT(IN) :: ninp, nout
01490     REAL(sz), INTENT(IN) :: vinp(ninp)
01491     INTEGER, INTENT(OUT) :: vout(nout)
01492
01493     INTEGER :: i, ic
01494
01495 !-----+
01496 ! Load INTEGER variable with input values.
01497 !-----+
01498
01499 ! If not all values are provided for variable, assume the last value
01500 ! for the rest of the array.
01501     ic = 0
01502     IF (ninp <= nout) THEN
01503         DO i = 1, ninp
01504             ic = ic + 1
01505             vout(i) = int(vinp(i))
01506         END DO
01507         DO i = ninp + 1, nout
01508             ic = ic + 1
01509             vout(i) = int(vinp(ninp))
01510         END DO
01511     ELSE
01512         DO i = 1, nout
01513             ic = ic + 1
01514             vout(i) = int(vinp(i))
01515         END DO
01516     END IF
01517
01518     nvalsout = ic
01519
01520     RETURN
01521
01522 END FUNCTION loadintvar
01523
01524 !=====
01525
01526 !-----+
01527 ! F U N C T I O N   L O A D   L O G   V A R
01528 !-----+
01550 !
01551 INTEGER FUNCTION loadlogvar (nInp, vInp, nOut, vOut) RESULT(nValsOut)
01552
01553     IMPLICIT NONE
01554
01555     INTEGER, INTENT(IN) :: ninp, nout
01556     CHARACTER(LEN=*) , INTENT(IN) :: vinp(ninp)
01557     LOGICAL, INTENT(OUT) :: vout(nout)
01558
01559     INTEGER :: i, ic
01560
01561 !-----+
01562 ! Load INTEGER variable with input values.
01563 !-----+
01564

```

```

01565 ! If not all values are provided for variable, assume the last value
01566 ! for the rest of the array.
01567 ic = 0
01568 IF (ninp <= nout) THEN
01569   DO i = 1, ninp
01570     ic = ic + 1
01571     IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01572       vout(i) = .true.
01573     ELSE
01574       vout(i) = .false.
01575     END IF
01576   END DO
01577   DO i = ninp + 1, nout
01578     ic = ic + 1
01579     IF ((vinp(ninp)(1:1) == 'T') .OR. (vinp(ninp)(1:1) == 't')) THEN
01580       vout(i) = .true.
01581     ELSE
01582       vout(i) = .false.
01583     END IF
01584   END DO
01585 ELSE
01586   DO i = 1, nout
01587     ic = ic + 1
01588     IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01589       vout(i) = .true.
01590     ELSE
01591       vout(i) = .false.
01592     END IF
01593   END DO
01594 END IF
01595
01596 nvalsout = ic
01597
01598 RETURN
01599
01600 END FUNCTION loadlogvar
01601
01602 !=====
01603
01604 !-----+
01605 ! F U N C T I O N   L O A D   R E A L   V A R
01606 !-----+
01628 !
01629 INTEGER FUNCTION loadrealvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01630
01631 IMPLICIT NONE
01632
01633 INTEGER, INTENT(IN) :: ninp, nout
01634 REAL(sz), INTENT(IN) :: vinp(ninp)
01635 REAL(sz), INTENT(OUT) :: vout(nout)
01636
01637 INTEGER :: i, ic
01638
01639 !-----+
01640 ! Load INTEGER variable with input values.
01641 !-----+
01642
01643 ! If not all values are provided for variable, assume the last value
01644 ! for the rest of the array.
01645 ic = 0
01646 IF (ninp <= nout) THEN
01647   DO i = 1, ninp
01648     ic = ic + 1
01649     vout(i) = vinp(i)
01650   END DO
01651   DO i = ninp + 1, nout
01652     ic = ic + 1
01653     vout(i) = vinp(ninp)
01654   END DO
01655 ELSE
01656   DO i = 1, nout
01657     ic = ic + 1
01658     vout(i) = vinp(i)
01659   END DO
01660 END IF
01661
01662 nvalsout = ic
01663
01664 RETURN
01665
01666 END FUNCTION loadrealvar

```

```

01667 !=====
01668 !-----+
01669 !
01670 !-----+
01671 ! F U N C T I O N   T O   L O W E R   C A S E
01672 !-----+
01686 !
01687 PURE FUNCTION tolowercase(inpString) RESULT(outString)
01688
01689     IMPLICIT NONE
01690
01691     CHARACTER(*), INTENT(IN)    :: inpstring
01692
01693     INTEGER, PARAMETER         :: duc = ichar('A') - ichar('a')
01694     CHARACTER(LEN(inpString))  :: outstring
01695     CHARACTER                  :: ch
01696     INTEGER                     :: i
01697
01698     DO i = 1, len(inpstring)
01699         ch = inpstring(i:i)
01700         IF ((ch >= 'A') .AND. (ch <= 'Z')) ch = char(ichar(ch) - duc)
01701         outstring(i:i) = ch
01702     END DO
01703
01704     RETURN
01705
01706 END FUNCTION tolowercase
01707
01708 !=====
01709 !
01710 !-----+
01711 ! F U N C T I O N   T O   U P P E R   C A S E
01712 !-----+
01726 !
01727 PURE FUNCTION touppercase(inpString) RESULT(outString)
01728
01729     IMPLICIT NONE
01730
01731     CHARACTER(*), INTENT(IN)    :: inpstring
01732
01733     INTEGER, PARAMETER         :: duc = ichar('A') - ichar('a')
01734     CHARACTER(LEN(inpString))  :: outstring
01735     CHARACTER                  :: ch
01736     INTEGER                     :: i
01737
01738     DO i = 1, len(inpstring)
01739         ch = inpstring(i:i)
01740         IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
01741         outstring(i:i) = ch
01742     END DO
01743
01744     RETURN
01745
01746 END FUNCTION touppercase
01747
01748 !=====
01749 !
01750 !-----+
01751 ! F U N C T I O N   C O N V _ L O N
01752 !
01766 !
01767 REAL(sz) function convlon(inplon) result (myvalout)
01768
01769     IMPLICIT NONE
01770
01771     REAL(sz), INTENT(IN)    :: inplon
01772
01773     myvalout = mod(inplon + 180.0_sz, 360.0_sz) - 180.0_sz
01774
01775     RETURN
01776
01777 END FUNCTION convlon
01778
01779 !=====
01780 !
01781 !-----+
01782 ! S U B R O U T I N E   G E O   T O   C P P   S C A L A R
01783 !
01806 !
01807 SUBROUTINE geotocpp_scalar(lat, lon, lat0, lon0, x, y)
01808

```

```
01809 USE pahm_global, ONLY : rearth, deg2rad
01810
01811 IMPLICIT NONE
01812
01813 REAL(SZ), INTENT(IN) :: lat
01814 REAL(SZ), INTENT(IN) :: lon
01815 REAL(SZ), INTENT(IN) :: lat0
01816 REAL(SZ), INTENT(IN) :: lon0
01817 REAL(SZ), INTENT(OUT) :: x
01818 REAL(SZ), INTENT(OUT) :: y
01819
01820 x = deg2rad * rearth * (lon - lon0) * cos(lat0)
01821 y = deg2rad * rearth * lat
01822
01823 END SUBROUTINE geotocpp_scalar
01824
01825 !=====
01826
01827 !-----+
01828 ! S U B R O U T I N E   G E O   T O   C P P   1D
01829 !-----+
01830
01831 !
01832
01833 !-----+
01834 SUBROUTINE geotocpp_1d(lat, lon, lat0, lon0, x, y)
01835
01836 USE pahm_global, ONLY : rearth, deg2rad
01837
01838 IMPLICIT NONE
01839
01840 REAL(SZ), INTENT(IN) :: lat(:)
01841 REAL(SZ), INTENT(IN) :: lon(:)
01842 REAL(SZ), INTENT(IN) :: lat0
01843 REAL(SZ), INTENT(IN) :: lon0
01844 REAL(SZ), INTENT(OUT) :: x(:)
01845 REAL(SZ), INTENT(OUT) :: y(:)
01846
01847 x = deg2rad * rearth * (lon - lon0) * cos(lat0)
01848 y = deg2rad * rearth * lat
01849
01850 END SUBROUTINE geotocpp_1d
01851
01852 !=====
01853
01854 !-----+
01855 ! S U B R O U T I N E   C P P   T O   G E O   S C A L A R
01856 !-----+
01857
01858 !
01859
01860 !-----+
01861 SUBROUTINE cpptogeoo_scalar(x, y, lat0, lon0, lat, lon)
01862
01863 USE pahm_global, ONLY : rearth, deg2rad
01864
01865 IMPLICIT NONE
01866
01867 REAL(SZ), INTENT(IN) :: x
01868 REAL(SZ), INTENT(IN) :: y
01869 REAL(SZ), INTENT(IN) :: lat0
01870 REAL(SZ), INTENT(IN) :: lon0
01871 REAL(SZ), INTENT(OUT) :: lat
01872 REAL(SZ), INTENT(OUT) :: lon
01873
01874 lat = y / (deg2rad * rearth)
01875 lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01876
01877 END SUBROUTINE cpptogeoo_scalar
01878
01879 !=====
01880
01881 !-----+
01882 ! S U B R O U T I N E   C P P   T O   G E O   1D
01883 !-----+
01884
01885 !
01886
01887 !-----+
01888 SUBROUTINE cpptogeoo_1d(x, y, lat0, lon0, lat, lon)
01889
01890 USE pahm_global, ONLY : rearth, deg2rad
01891
01892 IMPLICIT NONE
01893
01894 REAL(SZ), INTENT(IN) :: x(:)
01895 REAL(SZ), INTENT(IN) :: y(:)
01896 REAL(SZ), INTENT(IN) :: lat0
01897 REAL(SZ), INTENT(IN) :: lon0
01898 REAL(SZ), INTENT(OUT) :: lat(:)
```

```

01958     REAL(SZ), INTENT(OUT) :: lon(:)
01959
01960     lat = y / (deg2rad * rearth)
01961     lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01962
01963 END SUBROUTINE cpptogeo_1d
01964
01965 !=====
01966 ! -----
01967 ! 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
01968 ! -----
01969 !
01970 !-----
02000 REAL(sz) function sphericaldistance_scalar(lat1, lon1, lat2, lon2) result(myvalout)
02001
02002     USE pahm_global, ONLY : rearth, deg2rad
02003
02004     IMPLICIT NONE
02005
02006     REAL(sz), INTENT(IN) :: lat1      ! latitude of point 1 on the sphere (degrees north)
02007     REAL(sz), INTENT(IN) :: lon1      ! longitude of point 1 on the sphere (degrees east)
02008     REAL(sz), INTENT(IN) :: lat2      ! latitude of point 2 on the sphere (degrees north)
02009     REAL(sz), INTENT(IN) :: lon2      ! longitude of point 2 on the sphere (degrees east)
02010
02011     REAL(sz)          :: phil, phi2, lamdal, lamda2, dphi, dlamda, dsigma
02012
02013     phil   = deg2rad * lat1
02014     phi2   = deg2rad * lat2
02015     dphi   = abs(phi2 - phil)
02016
02017     lamdal = deg2rad * lon1
02018     lamda2 = deg2rad * lon2
02019     dlamda = abs(lamda2 - lamdal)
02020
02021     ! Vincenty formula to calculate a distance along a sphere
02022     dsigma = atan(sqrt((cos(phi2) * sin(dlamda))**2 + &
02023                         (cos(phi1) * sin(phi2) - sin(phi1) * cos(phi2) * cos(dlamda))**2))
02024     dsigma = dsigma / (sin(phi1) * sin(phi2) + cos(phi1) * cos(phi2) * cos(dlamda))
02025
02026     ! This is the great-circle distance; REARTH in meters
02027     myvalout = rearth * dsigma
02028
02029     RETURN
02030
02031 END FUNCTION sphericaldistance_scalar
02032
02033 !=====
02034 ! -----
02035 ! 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
02036 ! -----
02037 !
02038 !
02068 FUNCTION sphericaldistance_1d(lats, lons, lat0, lon0) RESULT(myValOut)
02069
02070     USE pahm_global, ONLY : rearth, deg2rad
02071
02072     IMPLICIT NONE
02073
02074     ! Global variables
02075     REAL(sz), INTENT(IN) :: lats(:) ! latitude of point 1 on the sphere (degrees north)
02076     REAL(sz), INTENT(IN) :: lons(:) ! longitude of point 1 on the sphere (degrees east)
02077     REAL(sz), INTENT(IN) :: lat0    ! latitude of point 2 on the sphere (degrees north)
02078     REAL(sz), INTENT(IN) :: lon0    ! longitude of point 2 on the sphere (degrees east)
02079
02080     REAL(sz), DIMENSION(:), ALLOCATABLE :: myvalout
02081
02082     ! Local variables
02083     REAL(sz), DIMENSION(:), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
02084     REAL(sz)                           :: phi0, lamda0
02085     INTEGER                            :: status, n1
02086
02087
02088     CALL setmessagesource("SphericalDistance_1D")
02089
02090     IF (SIZE(lats) /= SIZE(lons)) THEN
02091         WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
02092         CALL allmessage(error, scratchmessage)
02093
02094         CALL terminate()
02095     END IF
02096

```

```

02097 n1 = SIZE(lats, 1)
02098 ALLOCATE(myvalout(n1), stat = status)
02099 ALLOCATE(phiis(n1), lamdas(n1), dphi(n1), dlamda(n1), stat = status)
02100
02101 IF (status /= 0) THEN
02102   WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02103   CALL allmessage(error, scratchmessage)
02104
02105   CALL terminate()
02106 END IF
02107
02108 phiis = deg2rad * lats
02109 phi0 = deg2rad * lat0
02110 dphi = abs(phi0 - phiis)
02111
02112 lamdas = deg2rad * lons
02113 lamda0 = deg2rad * lon0
02114 dlamda = abs(lamda0 - lamdas)
02115
02116 ! Vincenty formula to calculate a distance along a sphere
02117 dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
02118                   (cos(phiis) * sin(phi0) - sin(phiis) * cos(phi0) * cos(dlamda))**2))
02119 dsigma = dsigma / (sin(phiis) * sin(phi0) + cos(phiis) * cos(phi0) * cos(dlamda))
02120
02121 ! This is the great-circle distance; REARTH in meters
02122 myvalout = rearth * dsigma
02123
02124 DEALLOCATE(phiis, lamdas, dphi, dlamda, dsigma)
02125
02126 CALL unsetmessagesource()
02127
02128 RETURN
02129
02130 END FUNCTION sphericaldistance_1d
02131
02132 !=====
02133 !
02134 ! -----
02135 ! 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
02136 ! -----
02137 !
02138
02139 FUNCTION sphericaldistance_2d(lats, lons, lat0, lon0) RESULT(myValOut)
02140
02141 USE pahm_global, ONLY : rearth, deg2rad
02142
02143 IMPLICIT NONE
02144
02145 ! Global variables
02146 REAL(sz), INTENT(IN) :: lats(:, :) ! latitude of point 1 on the sphere (degrees north)
02147 REAL(sz), INTENT(IN) :: lons(:, :) ! longitude of point 1 on the sphere (degrees east)
02148 REAL(sz), INTENT(IN) :: lat0        ! latitude of point 2 on the sphere (degrees north)
02149 REAL(sz), INTENT(IN) :: lon0        ! longitude of point 2 on the sphere (degrees east)
02150
02151 REAL(sz), DIMENSION(:, :, ), ALLOCATABLE :: myvalout
02152
02153 ! Local variables
02154 REAL(sz), DIMENSION(:, :, ), ALLOCATABLE :: phiis, lamdas, dphi, dlamda, dsigma
02155 REAL(sz) :: phi0, lamda0
02156 INTEGER :: status, n1, n2
02157
02158
02159 CALL setmessagesource("SphericalDistance_2D")
02160
02161 IF (SIZE(lats) /= SIZE(lons)) THEN
02162   WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
02163   CALL allmessage(error, scratchmessage)
02164
02165   CALL unsetmessagesource()
02166
02167   CALL terminate()
02168 END IF
02169
02170 n1 = SIZE(lats, 1)
02171 n2 = SIZE(lats, 2)
02172 ALLOCATE(myvalout(n1, n2), stat = status)
02173 ALLOCATE(phiis(n1, n2), lamdas(n1, n2), dphi(n1, n2), dlamda(n1, n2), dsigma(n1, n2), stat = status)
02174
02175 IF (status /= 0) THEN
02176   WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02177   CALL allmessage(error, scratchmessage)
02178
02179
02180
02181
02182
02183
02184
02185
02186
02187
02188
02189
02190
02191
02192
02193
02194
02195
02196
02197
02198
02199
02200
02201
02202
02203
02204
02205
02206

```

```

02207     CALL unsetmessagesource()
02208
02209     CALL terminate()
02210 END IF
02211
02212 phis    = deg2rad * lats
02213 phi0   = deg2rad * lat0
02214 dphi   = abs(phi0 - phis)
02215
02216 lamdas = deg2rad * lons
02217 lamda0 = deg2rad * lon0
02218 dlamda = abs(lamda0 - lamdas)
02219
02220 ! Vincenty formula to calculate a distance along a sphere
02221 dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + 
02222             (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
02223 dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))
02224
02225 ! This is the great-circle distance; REARTH in meters
02226 myvalout = rearth * dsigma
02227
02228 DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)
02229
02230 CALL unsetmessagesource()
02231
02232 RETURN
02233
02234 END FUNCTION sphericaldistance_2d
02235
02236 !=====
02237
02238 !
02239 !-----+
02240 ! F U N C T I O N S P H E R I C A L D I S T A N C E H A R V
02241 !-----+
02242
02243 !-----
02244
02245 REAL(sz) function sphericaldistancehav(lat1, lon1, lat2, lon2) result(myvalout)
02246
02247 USE pahm_global, ONLY : rearth, deg2rad
02248
02249 IMPLICIT NONE
02250
02251 REAL(sz), INTENT(IN) :: lat1      ! latitude of point 1 on the sphere (degrees north)
02252 REAL(sz), INTENT(IN) :: lon1      ! longitude of point 1 on the sphere (degrees east)
02253 REAL(sz), INTENT(IN) :: lat2      ! latitude of point 2 on the sphere (degrees north)
02254 REAL(sz), INTENT(IN) :: lon2      ! longitude of point 2 on the sphere (degrees east)
02255
02256 REAL(sz)          :: phi1, phi2, lamdal, lamda2, dphi, dlamda, dsigma
02257
02258 phi1   = deg2rad * lat1
02259 phi2   = deg2rad * lat2
02260 dphi   = abs(phi2 - phi1)
02261
02262 lamdal = deg2rad * lon1
02263 lamda2 = deg2rad * lon2
02264 dlamda = abs(lamda2 - lamdal)
02265
02266 ! Haversine formula to calculate a distance along a sphere
02267 dsigma = sqrt(sin(dphi / 2.0_sz)**2 + cos(phi1) * cos(phi2) * sin(dlamda / 2.0_sz)**2)
02268 dsigma = 2.0_sz * asin(dsigma)
02269
02270 ! This is the great-circle distance; REARTH in meters
02271 myvalout = rearth * dsigma
02272
02273 RETURN
02274
02275 END FUNCTION sphericaldistancehav
02276
02277 !=====
02278
02279 !-----+
02280 ! S U B R O U T I N E S P H E R I C A L F R A C P O I N T
02281 !-----+
02282
02283 !
02284
02285 SUBROUTINE sphericalfracpoint(lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)
02286
02287 USE pahm_global, ONLY : rearth, deg2rad, rad2deg
02288
02289 IMPLICIT NONE
02290
02291 ! Global variables
02292 REAL(SZ), INTENT(IN)          :: lat1      ! latitude of point 1 on the sphere (degrees north)

```

```

02347    REAL(SZ), INTENT(IN)          :: lon1      ! longitude of point 1 on the sphere (degrees east)
02348    REAL(SZ), INTENT(IN)          :: lat2      ! latitude of point 2 on the sphere (degrees north)
02349    REAL(SZ), INTENT(IN)          :: lon2      ! longitude of point 2 on the sphere (degrees east)
02350    REAL(SZ), INTENT(IN)          :: fraction   ! distance fraction of the intermediate point (0 <= f
02351    <= 1)
02351    REAL(SZ), INTENT(OUT)         :: latf, lonf ! the calculated latitude and longitude of the
02352                                         ! intermediate point
02353    REAL(SZ), OPTIONAL, INTENT(OUT) :: distf     ! the distance between point 1 and the intermediate
02354                                         ! point
02354    REAL(SZ), OPTIONAL, INTENT(OUT) :: dist12    ! the distance between point 1 and point 2
02355
02356    ! Local variables
02357    REAL(SZ)                      :: myFrac
02358    REAL(SZ)                      :: phil, phi2, lamdal, lamda2, delta
02359    REAL(SZ)                      :: aa, bb, xx, yy, zz
02360    REAL(SZ) :: myDist12, myDistF
02361
02362
02363    myfrac = fraction
02364    IF (myfrac < 0) myfrac = 0.0_sz
02365    IF (myfrac > 1) myfrac = 1.0_sz
02366
02367    ! Calculate the great circle distance between points 1 and 2
02368    mydist12 = sphericaldistance(lat1, lon1, lat2, lon2)
02369
02370    ! Distance is in meters (REARTH in meters). If myDist12 < 0.01_sz
02371    ! the two points are coincident
02372    IF (mydist12 < 0.01_sz) THEN
02373        latf = lat1
02374        lonf = lon1
02375        IF (PRESENT(distf)) distf = 0.0_sz
02376        IF (PRESENT(dist12)) dist12 = 0.0_sz
02377
02378        RETURN
02379    END IF
02380
02381    phil = deg2rad * lat1
02382    phi2 = deg2rad * lat2
02383    lamdal = deg2rad * lon1
02384    lamda2 = deg2rad * lon2
02385
02386    delta = mydist12 / rearth
02387
02388    aa = sin((1.0_sz - myfrac) * delta) / sin(delta)
02389    bb = sin(myfrac * delta) / sin(delta)
02390
02391    xx = aa * cos(phi1) * cos(lamdal) + bb * cos(phi2) * cos(lamda2)
02392    yy = aa * cos(phi1) * sin(lamdal) + bb * cos(phi2) * sin(lamda2)
02393    zz = aa * sin(phi1) + bb * sin(phi2)
02394
02395    ! The (lat, lon) values of the intermediate point
02396    latf = rad2deg * atan2(zz, sqrt(xx * xx + yy * yy))
02397    lonf = rad2deg * atan2(yy, xx)
02398
02399    ! This is the great-circle distance; REARTH in meters
02400    mydistf = sphericaldistance(lat1, lon1, latf, lonf)
02401
02402    IF (PRESENT(distf)) distf = mydistf
02403    IF (PRESENT(dist12)) dist12 = mydist12
02404
02405    RETURN
02406
02407 END SUBROUTINE sphericalfracpoint
02408
02409 !=====
02410
02411 !-----
02412 ! S U B R O U T I N E   G E T   L O C   A N D   R A T I O
02413 !-----
02436 !
02437 SUBROUTINE getlocandratio(val, arrVal, idx1, idx2, wtRatio)
02438
02439 IMPLICIT NONE
02440
02441 ! Global variables
02442    REAL(SZ), INTENT(IN) :: val      ! value to search for
02443    REAL(SZ), INTENT(IN) :: arrVal(:) ! search array (1D)
02444    INTEGER, INTENT(OUT) :: idx1     ! the index of the lowest bound
02445    INTEGER, INTENT(OUT) :: idx2     ! the index of the highest bound
02446    REAL(SZ), INTENT(OUT) :: wtRatio ! the ratio factor that used in the linear interpolation
02447                                         ! calculations: F = F(idx1) + wtRatio * (F(idx2) - F(idx1))

```

```

02448           ! 0 <= wtRatio <= 1.0
02449
02450 ! Local variables
02451 INTEGER          :: nn, jl, jll, jl2
02452 REAL(SZ)         :: diffVal
02453
02454
02455      idx1 = -1
02456      idx2 = -1
02457      wtratio = 0.0_sz
02458
02459      nn = SIZE(arrval, 1)
02460      jl = minloc(abs(val - arrval), 1)
02461
02462 !----- Check if we got an exact bin value
02463 IF (comparereals(val - arrval(jl), 0.0_sz) == 0) THEN
02464     idx1 = jl
02465     idx2 = jl
02466     wtratio = 0.0_sz
02467
02468     RETURN
02469 END IF
02470 !-----
02471
02472 !----- Checking the values at the two edges of the arrVal
02473 IF ((jl == 1) .OR. (jl == nn)) THEN
02474     IF (jl == 1) THEN
02475         jll = jl
02476         jl2 = jl + 1
02477     ELSE
02478         jll = jl - 1
02479         jl2 = jl
02480     END IF
02481
02482     diffval = arrval(jl2) - arrval(jll)
02483
02484     IF (comparereals(diffval, 0.0_sz) == 0) THEN
02485         idx1 = jll
02486         idx2 = jll
02487         wtratio = 0.0_sz
02488
02489     ELSE
02490         IF (comparereals(val - arrval(jll), 0.0_sz) * &
02491             comparereals(val - arrval(jl2), 0.0_sz) < 0) THEN
02492             idx1 = jll
02493             idx2 = jl2
02494             wtratio = (val - arrval(jll)) / diffval
02495
02496     END IF
02497 END IF
02498
02499     RETURN
02500 END IF
02501 !-----
02502
02503 IF (comparereals(val - arrval(jl - 1), 0.0_sz) * &
02504     comparereals(val - arrval(jl), 0.0_sz) < 0) THEN
02505     jll = jl - 1
02506     jl2 = jl
02507
02508     diffval = arrval(jl2) - arrval(jll)
02509
02510     idx1 = jll
02511     idx2 = jl2
02512     wtratio = (val - arrval(jll)) / diffval
02513 ELSE IF (comparereals(val - arrval(jl), 0.0_sz) * &
02514     comparereals(val - arrval(jl + 1), 0.0_sz) < 0) THEN
02515
02516     jll = jl
02517     jl2 = jl + 1
02518
02519     diffval = arrval(jl2) - arrval(jll)
02520
02521     idx1 = jll
02522     idx2 = jl2
02523     wtratio = (val - arrval(jll)) / diffval
02524 END IF
02525
02526     RETURN
02527
02528 END SUBROUTINE getlocandratio

```

```

02529
02530 !=====
02531
02532 !-----
02533 ! F U N C T I O N   C H A R   U N I Q U E
02534 !-----
02535 !-----
02536
02537 INTEGER FUNCTION charunique(inpVec, outVec, idxVec) RESULT (myRec)
02538
02539 IMPLICIT NONE
02540
02541 CHARACTER(LEN=*) , INTENT(IN) :: inpvec(:)
02542 CHARACTER(LEN=*) , INTENT(OUT) :: outvec(:)
02543 INTEGER, ALLOCATABLE, INTENT(OUT) :: idxvec(:)
02544
02545 CHARACTER(LEN=LEN(inpVec(1))), ALLOCATABLE :: chkstr(:)
02546 INTEGER, ALLOCATABLE :: chkint(:)
02547 INTEGER :: neis
02548 INTEGER :: icnt, jcnt ! counters
02549
02550
02551 nels = SIZE(inpvec, 1)
02552
02553 ALLOCATE(chkstr(nels))
02554 ALLOCATE(chkint(nels))
02555
02556
02557 jcnd = 1
02558 DO icnt = 1, nels
02559   IF (trim(inpvec(icnt)) == "") cycle
02560   IF (any(chkstr == inpvec(icnt))) cycle
02561
02562   ! No match found so add it to the output
02563   chkstr(jcnd) = inpvec(icnt)
02564   chkint(jcnd) = icnt
02565   jcnd = jcnd + 1
02566
02567 END DO
02568
02569 myrec = jcnd - 1
02570 outvec = chkstr
02571 idxvec = chkint
02572
02573 DEALLOCATE(chkstr)
02574 DEALLOCATE(chkint)
02575
02576 RETURN
02577
02578 END FUNCTION charunique
02579
02580 !=====
02581
02582 !-----
02583 ! F U N C T I O N   V A L   S T R
02584 !-----
02585 !-----
02586 !-----
02587 !-----
02588 !-----
02589 !-----
02590 !-----
02591 !-----
02592 !-----
02593 !-----
02594 !-----
02595 !=====
02596
02597 !-----
02598 ! F U N C T I O N   V A L   S T R
02599 !-----
02600 !-----
02601 !-----
02602 !-----
02603 !-----
02604 !-----
02605 ! -----
02606 ! -----
02607 ! -----
02608 ! -----
02609 ! -----
02610 ! -----
02611 ! -----
02612 ! -----
02613 ! -----
02614 ! -----
02615 ! -----
02616 ! -----
02617 ! -----
02618 ! -----
02619 REAL(sp) function valstr(string) result(myval)
02620
02621 IMPLICIT NONE
02622
02623 ! Dummy arguments
02624 CHARACTER(LEN=*) , INTENT(IN) :: string
02625
02626 ! Local variables
02627 INTEGER :: i
02628 REAL(sp) :: v
02629
02630 i = realscan(string,1,v)
02631 myval = v
02632
02633 RETURN
02634
02635 END FUNCTION valstr
02636
02637 !=====
02638
02639 !-----
02640 ! F U N C T I O N   D   V A L   S T R
02641 !-----
02642 !-----
02643 !-----
02644 ! -----
02645 ! -----
02646 ! -----
02647 ! -----
02648 ! -----
02649 ! -----
02650 ! -----
02651 ! -----
02652 ! -----
02653 ! -----
02654 ! -----
02655 ! -----
02656 ! -----
02657 ! -----
02658 ! -----
02659 ! -----
02660 ! -----
02661 REAL(hp) function dvalstr(string) result(myval)
02662

```

```

02663      IMPLICIT NONE
02664
02665      ! Dummy arguments
02666      CHARACTER(LEN=*), INTENT(IN) :: string
02667
02668      ! Local variables
02669      INTEGER :: i
02670      REAL(hp) :: v
02671
02672      i = drealscan(string,1,v)
02673      myval = v
02674
02675      RETURN
02676
02677      END FUNCTION dvalstr
02678
02679 !=====
02680
02681 !-----
02682 ! F U N C T I O N   I N T   V A L   S T R
02683 !-----
02702 !-----
02703      INTEGER FUNCTION intvalstr(String) Result(myVal)
02704
02705      IMPLICIT NONE
02706
02707      ! Dummy arguments
02708      CHARACTER(LEN=*), INTENT(IN) :: string
02709
02710      ! Local variables
02711      INTEGER :: i
02712      INTEGER :: v
02713
02714      i = intscan(string,1,.true.,v)
02715      myval = v
02716
02717      RETURN
02718
02719      END FUNCTION intvalstr
02720
02721 !=====
02722
02723 !-----
02724 ! F U N C T I O N   R E A L   S C A N
02725 !-----
02764 !-----
02765      INTEGER FUNCTION realscan(String, Pos, Value) Result(myVal)
02766
02767      IMPLICIT NONE
02768
02769      ! Dummy arguments
02770      INTEGER, INTENT(IN) :: pos
02771      CHARACTER(LEN=*), INTENT(IN) :: string
02772      REAL(sp), INTENT(OUT) :: value
02773
02774      ! Local variables
02775      INTEGER :: fract, intg, kfract, pmsign, power, ptr
02776
02777      ! CHECK POS.
02778      myval = pos
02779      Value = 0.0_sp
02780      IF(pos < 1 .OR. len(string) < pos) RETURN
02781
02782      ! SET UP WORKING VARIABLES.
02783      intg = 0
02784      fract = 0
02785      kfract = 0
02786      power = 0
02787      DO WHILE (.true.)
02788          ! SKIP LEADING BLANKS.
02789          IF(string(myval:myval) == ' ') THEN
02790              myval = myval + 1
02791              IF(myval > len(string)) RETURN
02792              cycle
02793          END IF
02794
02795          ! LOOK FOR SIGN.
02796          ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
02797          pmsign = 0
02798          IF(string(myval:myval) == '+') THEN
02799              pmsign = +1

```

```

02800      ELSE IF(string(myval:myval) == '-') THEN
02801          pmsign = -1
02802      END IF
02803      IF(pmsign.NE.0) myval = myval + 1
02804
02805      ! LOOK FOR INTEGER PART.
02806      myval = intscan(string,myval,.false.,intg)
02807
02808      ! LOOK FOR FRACTION PART.
02809      IF(myval.LE.len(string)) THEN
02810          IF(myval > pos+abs(pmsign)) THEN
02811              ! DETERMINE IF FIRST FORM OR SECOND FORM.
02812              ! HANDLE FIRST FORM:  D+ ['. D*]
02813              IF(string(myval:myval) == '.') THEN
02814                  myval = myval + 1
02815                  IF(myval.LE.len_trim(string)) THEN
02816                      IF(string(myval:myval).NE.' ') THEN
02817                          ptr = intscan(string,myval,.false.,fract)
02818                          kfract = ptr - myval
02819                          myval = ptr
02820                  END IF
02821          END IF
02822      END IF
02823      ! HANDLE SECOND FORM:  '.' D+
02824      ELSE IF(string(myval:myval).NE.')') THEN
02825          ! IF '.' MISSING, THEN WE HAVE NOTHING.
02826          myval = pos
02827          RETURN
02828
02829      ELSE
02830          myval = myval + 1
02831          ptr = intscan(string,myval,.false.,fract)
02832          kfract = ptr - myval
02833          IF(kfract == 0) THEN
02834              ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
02835              myval = pos
02836              RETURN
02837          ELSE
02838              myval = ptr
02839          END IF
02840      END IF
02841
02842      ! LOOK FOR EXPONENT PART.
02843      IF((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e')) THEN
02844          myval = myval + 1
02845          ptr = intscan(string,myval,.true.,power)
02846          IF(ptr == myval) THEN
02847              ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
02848              ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
02849              ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
02850              myval = myval - 1
02851              Value = intg + real(fract/10.0**kfract, sp)
02852              IF(pmsign == -1) Value = -Value
02853              RETURN
02854          ELSE
02855              myval = ptr
02856          END IF
02857      END IF
02858
02859  END IF
02860
02861      ! COMPUTE REAL VALUE FROM ITS PARTS.
02862      IF(kfract.NE.0) THEN
02863          Value = real((intg + fract/10.0**kfract)*10.0**power, sp)
02864      ELSE
02865          Value = real(intg*10.0**power, sp)
02866      END IF
02867      IF(pmsign == -1) Value = -Value
02868      EXIT
02869  END DO
02870
02871  RETURN
02872
02873 END FUNCTION realscan
02874
02875 =====
02876
02877 !-----+
02878 ! F U N C T I O N   D R E A L S C A N
02879 !-----+
02880 !-----+

```

```

02919  INTEGER FUNCTION drealscan(String,Pos,Value) RESULT(myVal)
02920
02921     IMPLICIT NONE
02922
02923     ! Dummy arguments
02924     INTEGER, INTENT(IN)      :: pos
02925     CHARACTER(LEN=*) , INTENT(IN) :: string
02926     REAL(hp), INTENT(OUT)    :: value
02927
02928     ! Local variables
02929     INTEGER :: fract, intg, kfract, pmsign, power, ptr
02930
02931     ! CHECK POS.
02932     myval = pos
02933     Value = 0.0
02934     IF(pos < 1 .OR. len(string) < pos) RETURN
02935
02936     ! SET UP WORKING VARIABLES.
02937     intg = 0
02938     fract = 0
02939     kfract = 0
02940     power = 0
02941     DO WHILE (.true.)
02942         ! SKIP LEADING BLANKS.
02943         IF(string(myval:myval) == ' ') THEN
02944             myval = myval + 1
02945             IF(myval > len(string)) RETURN
02946             cycle
02947         END IF
02948
02949         ! LOOK FOR SIGN.
02950         ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
02951         pmsign = 0
02952         IF(string(myval:myval) == '+') THEN
02953             pmsign = +1
02954         ELSE IF(string(myval:myval) == '-') THEN
02955             pmsign = -1
02956         END IF
02957         IF(pmsign.NE.0)myval = myval + 1
02958
02959         ! LOOK FOR INTEGER PART.
02960         myval = intscan(string,myval,.false.,intg)
02961
02962         ! LOOK FOR FRACTION PART.
02963         IF(myval.LE.len(string)) THEN
02964             IF(myval > pos+abs(pmsign)) THEN
02965                 ! DETERMINE IF FIRST FORM OR SECOND FORM.
02966                 ! HANDLE FIRST FORM: D+ [.' D*]
02967                 IF(string(myval:myval) == '.') THEN
02968                     myval = myval + 1
02969                     IF(myval.LE.len_trim(string)) THEN
02970                         IF(string(myval:myval).NE.' ') THEN
02971                             ptr = intscan(string,myval,.false.,fract)
02972                             kfract = ptr - myval
02973                             myval = ptr
02974                         END IF
02975                     END IF
02976                 END IF
02977                 ! HANDLE SECOND FORM: '.' D+
02978                 ELSE IF(string(myval:myval).NE.'.') THEN
02979                     ! IF '.' MISSING, THEN WE HAVE NOTHING.
02980                     myval = pos
02981                     RETURN
02982                 ELSE
02983                     myval = myval + 1
02984                     ptr = intscan(string,myval,.false.,fract)
02985                     kfract = ptr - myval
02986                     IF(kfract == 0) THEN
02987                         ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
02988                         myval = pos
02989                         RETURN
02990                     ELSE
02991                         myval = ptr
02992                     END IF
02993                 END IF
02994
02995         ! LOOK FOR EXPONENT PART.
02996         IF(myval.LE.len(string)) THEN
02997             IF((string(myval:myval) == 'E') .OR. (string(myval:myval) == 'e') .OR. &
02998                 (string(myval:myval) == 'D') .OR. (string(myval:myval) == 'd')) THEN
02999             myval = myval + 1

```

```

03000      ptr = intscan(string,myval,.true.,power)
03001      IF(ptr == myval) THEN
03002          ! IF WE HAVE THE 'E' BUT NOTHING ELSE THEN WE ASSUME
03003          ! THAT THE 'E' IS A TERMINATOR (E.G., 5.3EV) AND
03004          ! RETURN WHAT WE HAVE SO FAR (E.G., 5.3).
03005          myval = myval - 1
03006          Value = intg + fract/10.0**kfract
03007          IF(pmsign == -1)Value = -Value
03008          RETURN
03009      ELSE
03010          myval = ptr
03011          END IF
03012          END IF
03013          END IF
03014
03015
03016      ! COMPUTE REAL VALUE FROM ITS PARTS.
03017      IF(kfract.NE.0) THEN
03018          Value = (intg+fract/10.0**kfract)*10.0**power
03019      ELSE
03020          Value = intg*10.0**power
03021          END IF
03022          IF(pmsign == -1)Value = -Value
03023          EXIT
03024      END DO
03025
03026      RETURN
03027
03028  END FUNCTION drealscan
03029
03030 !=====
03031
03032 !-----
03033 !----- F U N C T I O N   I N T   S C A N
03034 !-----
03071 !
03072 INTEGER FUNCTION intscan(String, Pos, Signed, Value) Result(myVal)
03073
03074     IMPLICIT NONE
03075
03076     ! Dummy arguments
03077     INTEGER, INTENT(IN) :: pos
03078     LOGICAL, INTENT(IN) :: signed
03079     CHARACTER(LEN=*) , INTENT(IN) :: string
03080     INTEGER, INTENT(OUT) :: value
03081
03082     ! Local variables
03083     INTEGER(KIND=4) :: digit,pmsign
03084
03085     ! CHECK POS.
03086     myval = pos
03087     Value = 0
03088     IF(pos < 1 .OR. len(string) < pos)RETURN
03089     DO WHILE (.true.)
03090
03091         ! SKIP LEADING BLANKS.
03092         IF(string(myval:myval) == ' ') THEN
03093             myval = myval + 1
03094             IF(myval > len(string))RETURN
03095             cycle
03096         END IF
03097
03098         ! IF SIGNED, CHECK FOR SIGN.
03099         pmsign = 0
03100         IF(signed) THEN
03101             IF(string(myval:myval) == '+') THEN
03102                 pmsign = +1
03103             ELSE IF(string(myval:myval) == '-') THEN
03104                 pmsign = -1
03105             END IF
03106             IF(pmsign.NE.0)myval = myval + 1
03107
03108             ! IF sign is the last char in the field (with no integer following it)
03109             ! myVal value is left as POS or at the end of leading blanks.
03110             IF(myval > len_trim(string)) THEN
03111                 myval = myval - 1
03112                 RETURN
03113             END IF
03114         END IF
03115
03116         ! PROCESS DIGIT STRING.

```

```
03117      DO myval = myval,len(string)
03118          digit = ichar(string(myval:myval)) - ichar('0')
03119          IF(digit < 0 .OR. 9 < digit) GO TO 10
03120          Value = Value*10 + digit
03121      END DO
03122      ! Explicitly defined intscn to avoid possible compiler dependences (TWB. 930223)
03123      myval = len(string) + 1
03124      EXIT
03125  END DO
03126
03127      ! ADJUST SIGN.
03128  10 IF(signed.AND.pmsign == -1)Value = -Value
03129
03130      RETURN
03131
03132  END FUNCTION intscan
03133
03134 !=====
03135
03136 END MODULE utilities
03137
```

20.43 /home/takis/CSDL/parwinds-doc/src/vortex.F90 File Reference

Modules

- module pahm_vortex

Functions/Subroutines

- subroutine, public pahm_vortex::calcintensitychange (var, times, calclnt, status, order)
This subroutine calculates the intensity time change of a variable using second order numerical accuracy and uneven spacing.
- subroutine, public 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, public pahm_vortex::uvtranspoint (lat1, lon1, lat2, lon2, time1, time2, u, v)
This subroutine calculates the translational velocity of a moving hurricane.
- subroutine, public pahm_vortex::newvortex (pinf, p0, lat, lon, vm)
Creates a new Vortex object.
- subroutine, public pahm_vortex::newvortexfull (pinf, p0, lat, lon, vm)
Creates a new Vortex object.
- subroutine, public pahm_vortex::setvortex (pinf, p0, lat, lon)
Sets basic parameters for a new Vortex object.
- subroutine, public pahm_vortex::setrmaxes (rMaxW)
• subroutine, public pahm_vortex::getrmaxes (rMaxW)
• subroutine, public pahm_vortex::calcrmaxes ()
Calculates the radius of maximum winds for all storm quadrants.
- subroutine, public pahm_vortex::calcrmaxesfull ()
Calculates the radius of maximum winds for all storm quadrants.
- subroutine, public pahm_vortex::fitrmaxes ()
Calculates the coefficients that fit the given radius of maximum winds for all storm quadrants.
- subroutine, public pahm_vortex::fitrmaxes4 ()
- subroutine, public pahm_vortex::setvmaxesbl (vMaxW)
• subroutine, public pahm_vortex::getvmaxesbl (vMaxW)

- subroutine, public `pahm_vortex::setusevmaxesbl` (`u`)
- subroutine, public `pahm_vortex::setshapeparameter` (`param`)
- real(`sz`) function, public `pahm_vortex::getshapeparameter` ()
- real(`sz`) function, dimension(4), public `pahm_vortex::getshapeparameters` ()
- real(`sz`) function, dimension(4), public `pahm_vortex::getphifactors` ()
- subroutine, public `pahm_vortex::setisotachradii` (`ir`)
- subroutine, public `pahm_vortex::setisotachwindspeeds` (`vrQ`)
- subroutine, public `pahm_vortex::setisotachwindspeed` (`sp`)
- subroutine, public `pahm_vortex::setusequadrantvr` (`u`)
- logical function, public `pahm_vortex::getusequadrantvr` ()
- real(`sz`) function, public `pahm_vortex::splinep` (`angle, dist, opt`)

Spatial Interpolation function based on angle and r.

- real(`sz`) function, public `pahm_vortex::interp` (`quadVal, quadSel, quadDis`)
- real(`sz`) function, public `pahm_vortex::rmw` (`angle`)

Calculate the radius of maximum winds.

- subroutine, public `pahm_vortex::uvp` (`lat, lon, uTrans, vTrans, u, v, p`)

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

- subroutine, public `pahm_vortex::uvpr` (`iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, uTrans, vTrans, geof, u, v, p`)

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

- real(`sz`) function, public `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, public `pahm_vortex::getlatestrmax` ()
- real(`sz`) function, public `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), public `pahm_vortex::rmaxes4`
- real(`sz`) `pahm_vortex::pn`
- real(`sz`) `pahm_vortex::pc`
- real(`sz`) `pahm_vortex::clat`
- real(`sz`) `pahm_vortex::clon`
- real(`sz`) `pahm_vortex::vmax`
- real(`sz`) `pahm_vortex::b`
- real(`sz`) `pahm_vortex::corio`
- real(`sz`) `pahm_vortex::vr`
- real(`sz`) `pahm_vortex::phi`

- real(sz), dimension(npoints) `pahm_vortex::phis`
- real(sz), dimension(npoints, 4) `pahm_vortex::phis4`
- real(sz), dimension(npoints) `pahm_vortex::bs`
- real(sz), dimension(npoints, 4), public `pahm_vortex::bs4`
- real(sz), dimension(npoints) `pahm_vortex::vmb1`
- real(sz), dimension(npoints, 4), public `pahm_vortex::vmb14`
- integer, dimension(npoints, 4), public `pahm_vortex::quadflag4`
- real(sz), dimension(npoints, 4), public `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`

20.43.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Note

Adopted from the ADCIRC source code.

Definition in file `vortex.F90`.

20.44 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 PUBLIC :: newvortex, calcrmaxes, fitrmaxes,  
00025      &  
00026      rmw, uvp, uvtrans, fang,  
00027      &  
00028      getshapeparameter, setshapeparameter,  
00029      &  
00030      getlatestrmax, getlatestangle,  
00031      &  
00032      getusequadrantvr, getrmaxes,  
00033      &  
00034      setusequadrantvr, setrmaxes,  
00035      &  
00036      setisotachwindspeed,  
00037      &  
00038      getshapeparameters, getphifactors,  
00039      &  
00040      setisotachwindspeeds, setisotachradii,  
00041      &  
00042      setvortex, spinterp, interpr,  
00043      &  
00044      setusevmaxesbl,  
00045      &  
00046      getvmaxesbl, setvmaxesbl,  
00047      &  
00048      fitrmaxes4, calcrmaxesfull,  
00049      &  
00050      uvpr, newvortexfull,  
00051      &
```

```
00038      rmaxes4, quadflag4, quadir4, bs4, vmb14, &
00039      calcintensitychange, uvtranspoint
00040
00041 PRIVATE
00042
00043 INTEGER, PARAMETER :: nquads = 4           ! Number of quadrants for which wind radii are
00044 provided
00045 INTEGER, PARAMETER :: npoints = nquads + 2   ! Number of (theta, rMax) points for curve fit
00046 REAL(sz), DIMENSION(NPOINTS) :: rmaxes       ! Radius of maximum winds
00047 REAL(sz), DIMENSION(NPOINTS, 4) :: rmaxes4      ! (nautical miles)
00048 REAL(sz) :: pn                                ! Ambient surface pressure (mb) !PV global
00049 var?
00050 REAL(sz) :: pc                                ! Surface pressure at center of storm (mb) !PV
00051 REAL(sz) :: clat                             ! Latitude of storm center (degrees north)
00052 !PV global var?
00053 REAL(sz) :: clon                             ! Longitude of storm center (degrees east)
00054 REAL(sz) :: vmax                                ! Max sustained wind velocity in storm (knots)
00055 !PV global var?
00056
00057 REAL(sz) :: b                                 ! Exponential shape parameter
00058 REAL(sz) :: corio                            ! Coriolis force (1/s)
00059 REAL(sz) :: vr                                ! Velocity @ wind radii (knots)
00060 REAL(sz) :: phi                               ! phi
00061 REAL(sz), DIMENSION(NPOINTS) :: phis          ! Correction factor to B and vh
00062 REAL(sz), DIMENSION(NPOINTS, 4) :: phis4        ! Correction factor to B and vh
00063
00064 REAL(sz), DIMENSION(NPOINTS) :: bs             ! Wind radii - the distance
00065 REAL(sz), DIMENSION(NPOINTS, 4) :: bs4
00066 REAL(sz), DIMENSION(NPOINTS, 4) :: vmb14
00067 REAL(sz), DIMENSION(NPOINTS, 4) :: quadflag4
00068 REAL(sz), DIMENSION(NPOINTS, 4) :: quadir4
00069 REAL(sz), DIMENSION(NQUADS) :: vrquadrant
00070 REAL(sz), DIMENSION(NQUADS) :: radius
00071 ! Quadrant counter
00072 !-----+
00073 !-----+
00074 !-----+
00075 !-----+
00076 !-----+
00077 !-----+
00078 !-----+
00079 !-----+
00080 !-----+
00081 !-----+
00082 !-----+
00083 !-----+
00084 !-----+
00085 !-----+
00086 !-----+
00087 !-----+
00088 !-----+
00089 !-----+
00090 !-----+
00091 !-----+
00092 !-----+
00093 !-----+
00094 !-----+
00095 !-----+
00096 !-----+
00097 !-----+
00098 !-----+
00099 !-----+
00100 !-----+
00101 !-----+
00102 !-----+
00103 !-----+
00104 !-----+
00105 !-----+
00106 !-----+
00107 !-----+
00108 !-----+
00109 !-----+
00110 !-----+
00111 !-----+
00112 !-----+
00113 !-----+
00114 !-----+
00115 !-----+
00116 !-----+
00117 !-----+
00118 !-----+
00119 !-----+
00120 !-----+
00121 !-----+
00122 !-----+
00123 !-----+
00124 !-----+
00125 !-----+
00126 !-----+
00127 !-----+
00128 !-----+
00129 !-----+
00130 !-----+
00131 !-----+
00132 !-----+
00133 !-----+
```

```

00134     CALL allmessage(error, scratchmessage)
00135
00136     CALL unsetmessagesource()
00137
00138     status = 1
00139
00140     RETURN
00141
00142     maxcnt = SIZE(var)
00143     END IF
00144
00145     ordacur = 2
00146     IF (PRESENT(order)) THEN
00147         IF (order <= 1) ordacur = 1
00148         IF (order > 1) ordacur = 2
00149     END IF
00150     IF (SIZE(var) < 3) ordacur = 1
00151
00152     ! Case 1st order accuracy using backward differences
00153     IF (ordacur == 1 )THEN
00154         DO icnt = 2, maxcnt
00155             dt1 = times(icnt) - times(icnt - 1)
00156             dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00157
00158             vall = 0.0_sz
00159             IF (dt1ok) vall = (var(icnt) - var(icnt - 1)) / dt1
00160
00161             calcint(icnt) = vall
00162         END DO
00163         calcint(1) = calcint(2)
00164
00165         CALL unsetmessagesource()
00166
00167         RETURN
00168     END IF
00169
00170     ! Case 2nd order accuracy using Forward differences for the first point,
00171     ! backward differences for the last point and central differences in
00172     ! between points. Temporal spacing assumed to be uneven (general case).
00173     ! Forward, backward and central differences are all 2nd order accurate
00174     ! approximations.
00175
00176     !----- Forward differences (first point)
00177     icnt = 1
00178     dt1 = times(icnt + 1) - times(icnt)
00179     dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00180     dt2 = times(icnt + 2) - times(icnt + 1)
00181     dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00182
00183     vall = 0.0_sz
00184     IF (dt1ok) vall = (var(icnt + 1) - var(icnt)) / dt1
00185
00186     val2 = 0.0_sz
00187     IF (dt2ok) val2 = (var(icnt + 2) - var(icnt + 1)) / dt2
00188
00189     IF (dt1ok .AND. dt2ok) THEN
00190         calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * vall - (dt1 / (dt1 + dt2)) * val2
00191     ELSE IF (.NOT. dt1ok) THEN
00192         calcint(icnt) = vall
00193     ELSE
00194         calcint(icnt) = 2.0_sz * vall - val2
00195     END IF
00196     !----- Forward differences (first point)
00197
00198     !----- Central differences
00199     DO icnt = 2, maxcnt - 1
00200         ! Forward
00201         dt1 = times(icnt + 1) - times(icnt)
00202         dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00203         ! Backward
00204         dt2 = times(icnt) - times(icnt - 1)
00205         dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00206
00207         vall = 0.0_sz
00208         IF (dt1ok) vall = (var(icnt + 1) - var(icnt)) / dt1
00209
00210         val2 = 0.0_sz
00211         IF (dt2ok) val2 = (var(icnt) - var(icnt - 1)) / dt2
00212
00213         IF (dt1ok .AND. dt2ok) THEN
00214             calcint(icnt) = (dt2 / (dt1 + dt2)) * vall + (dt1 / (dt1 + dt2)) * val2

```

```

00215      ELSE IF (.NOT. dtlok) THEN
00216          calcint(icnt) = val1
00217      ELSE
00218          calcint(icnt) = val2
00219      END IF
00220  END DO
00221 !----- Central differences
00222
00223 !----- Backward differences (last point)
00224  icnt = maxcnt
00225  dt1 = times(icnt) - times(icnt - 1)
00226  dtlok = (comparereals(dt1, 0.0_sz) /=0)
00227  dt2 = times(icnt - 1) - times(icnt - 2)
00228  dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00229
00230  val1 = 0.0_sz
00231  IF (dtlok) val1 = (var(icnt) - var(icnt - 1)) / dt1
00232
00233  val2 = 0.0_sz
00234  IF (dt2ok) val2 = (var(icnt - 1) - var(icnt - 2)) / dt2
00235
00236  IF (dtlok .AND. dt2ok) THEN
00237      calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * val1 - (dt1 / (dt1 + dt2)) * val2
00238  ELSE IF (.NOT. dtlok) THEN
00239      calcint(icnt) = val1
00240  ELSE
00241      calcint(icnt) = 2.0_sz * val1 - val2
00242  END IF
00243 !----- Backward differences (last point)
00244
00245  CALL unsetmessagesource()
00246
00247 END SUBROUTINE calcintensitychange
00248 !=====
00250 !-----
00251 !----- S U B R O U T I N E   U V   T R A N S
00252 !-----
00253 !-----
00254 !-----
00255 SUBROUTINE uvtrans(lat, lon, times, u, v, status, order)
00256
00257 USE pahm_global, ONLY : deg2rad
00258 USE utilities, ONLY : sphericaldistance
00259
00260 IMPLICIT NONE
00261
00262 REAL(sz), DIMENSION(:), INTENT(IN) :: lat, lon, times
00263 INTEGER, OPTIONAL, INTENT(IN) :: order
00264
00265 REAL(sz), DIMENSION(:), INTENT(OUT) :: u, v
00266 INTEGER, INTENT(OUT) :: status
00267
00268 INTEGER :: ordacur
00269 REAL(sz) :: dx1, dy1, dx2, dy2
00270 REAL(sz) :: dt1, dt2
00271 LOGICAL :: dtlok, dt2ok
00272 REAL(sz) :: u1, u2, v1, v2
00273 INTEGER :: icnt, maxcnt
00274
00275 status = 0
00276 maxcnt = 0
00277
00278 CALL setmessagesource("UVTrans")
00279
00280 IF ((SIZE(shape(lat)) /= 1) .OR. (SIZE(shape(lon)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
00281     WRITE(scratchmessage, '(a)') 'The rank of arrays lat, lon and times should be equal to 1 (vectors)'
00282     CALL allmessage(error, scratchmessage)
00283
00284     CALL unsetmessagesource()
00285
00286     status = 1
00287
00288     RETURN
00289 ELSE
00290     maxcnt = SIZE(lat)
00291 END IF
00292
00293 ordacur = 2
00294 IF (PRESENT(order)) THEN
00295     IF (order <= 1) ordacur = 1

```

```

00323      IF (order > 1) ordacur = 2
00324  END IF
00325  IF (SIZE(lat) < 3) ordacur = 1
00326
00327 ! Case 1st order accuracy using backward differences
00328  IF (ordacur == 1 )THEN
00329    DO icnt = 2, maxcnt
00330      dx1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
00331      dy1 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
00332      dt1 = abs(times(icnt) - times(icnt - 1))
00333      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00334
00335      u1 = 0.0_sz
00336      v1 = 0.0_sz
00337      IF (dt1ok) THEN
00338        u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
00339        v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
00340    END IF
00341
00342      u(icnt) = u1
00343      v(icnt) = v1
00344  END DO
00345  u(1) = u(2)
00346  v(1) = v(2)
00347
00348  CALL unsetmessagesource()
00349
00350  RETURN
00351 END IF
00352
00353 ! Case 2nd order accuracy using Forward differences for the first point,
00354 ! backward differences for the last point and central differences in
00355 ! between points. Temporal spacing assumed to be uneven (general case).
00356 ! Forward, backward and central differences are all 2nd order accurate
00357 ! approximations.
00358
00359 !----- Forward differences (first point)
00360  icnt = 1
00361  dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
00362  dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
00363  dt1 = abs(times(icnt + 1) - times(icnt))
00364  dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00365
00366  dx2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 1), lon(icnt + 2))
00367  dy2 = sphericaldistance(lat(icnt + 1), lon(icnt + 1), lat(icnt + 2), lon(icnt + 1))
00368  dt2 = abs(times(icnt + 2) - times(icnt + 1))
00369  dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00370
00371  u1 = 0.0_sz
00372  v1 = 0.0_sz
00373  IF (dt1ok) THEN
00374    u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
00375    v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
00376  END IF
00377
00378  u2 = 0.0_sz
00379  v2 = 0.0_sz
00380  IF (dt2ok) THEN
00381    u2 = sign(dx2 / dt2, (lon(icnt + 2) - lon(icnt + 1)))
00382    v2 = sign(dy2 / dt2, (lat(icnt + 2) - lat(icnt + 1)))
00383  END IF
00384
00385  IF (dt1ok .AND. dt2ok) THEN
00386    u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
00387    v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
00388  ELSE IF (.NOT. dt1ok) THEN
00389    u(icnt) = u1
00390    v(icnt) = v1
00391  ELSE
00392    u(icnt) = 2.0_sz * u1 - u2
00393    v(icnt) = 2.0_sz * v1 - v2
00394  END IF
00395 !----- Forward differences (first point)
00396
00397 !----- Central differences
00398  DO icnt = 2, maxcnt - 1
00399    ! Forward
00400    dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt + 1))
00401    dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt + 1), lon(icnt))
00402    dt1 = abs(times(icnt + 1) - times(icnt))
00403    dt1ok = (comparereals(dt1, 0.0_sz) /=0)

```

```

00404      ! Backward
00405      dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
00406      dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
00407      dt2 = abs(times(icnt) - times(icnt - 1))
00408      dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00409
00410      u1 = 0.0_sz
00411      v1 = 0.0_sz
00412      IF (dt1ok) THEN
00413          u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
00414          v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
00415      END IF
00416
00417      u2 = 0.0_sz
00418      v2 = 0.0_sz
00419      IF (dt2ok) THEN
00420          u2 = sign(dx2 / dt2, (lon(icnt) - lon(icnt - 1)))
00421          v2 = sign(dy2 / dt2, (lat(icnt) - lat(icnt - 1)))
00422      END IF
00423
00424      IF (dt1ok .AND. dt2ok) THEN
00425          u(icnt) = (dt2 / (dt1 + dt2)) * u1 + (dt1 / (dt1 + dt2)) * u2
00426          v(icnt) = (dt2 / (dt1 + dt2)) * v1 + (dt1 / (dt1 + dt2)) * v2
00427      ELSE IF (.NOT. dt1ok) THEN
00428          u(icnt) = u1
00429          v(icnt) = v1
00430      ELSE
00431          u(icnt) = u2
00432          v(icnt) = v2
00433      END IF
00434  END DO
00435  !---- Central differences
00436
00437  !---- Backward differences (last point)
00438  icnt = maxcnt
00439  dx1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt), lon(icnt - 1))
00440  dy1 = sphericaldistance(lat(icnt), lon(icnt), lat(icnt - 1), lon(icnt))
00441  dt1 = abs(times(icnt) - times(icnt - 1))
00442  dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00443
00444  dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt - 2))
00445  dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 2), lon(icnt - 1))
00446  dt2 = abs(times(icnt - 1) - times(icnt - 2))
00447  dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00448
00449  u1 = 0.0_sz
00450  v1 = 0.0_sz
00451  IF (dt1ok) THEN
00452      u1 = sign(dx1 / dt1, (lon(icnt) - lon(icnt - 1)))
00453      v1 = sign(dy1 / dt1, (lat(icnt) - lat(icnt - 1)))
00454  END IF
00455
00456  u2 = 0.0_sz
00457  v2 = 0.0_sz
00458  IF (dt2ok) THEN
00459      u2 = sign(dx2 / dt2, (lon(icnt - 1) - lon(icnt - 2)))
00460      v2 = sign(dy2 / dt2, (lat(icnt - 1) - lat(icnt - 2)))
00461  END IF
00462
00463  IF (dt1ok .AND. dt2ok) THEN
00464      u(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * u1 - (dt1 / (dt1 + dt2)) * u2
00465      v(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * v1 - (dt1 / (dt1 + dt2)) * v2
00466  ELSE IF (.NOT. dt1ok) THEN
00467      u(icnt) = u1
00468      v(icnt) = v1
00469  ELSE
00470      u(icnt) = 2.0_sz * u1 - u2
00471      v(icnt) = 2.0_sz * v1 - v2
00472  END IF
00473  !---- Backward differences (last point)
00474
00475  CALL unsetmessagesource()
00476
00477  END SUBROUTINE uvtrans
00478
00479 !=====
00480
00481 !-----  

00482 ! S U B R O U T I N E   U V T R A N S   P O I N T  

00483 !-----  

00508 !-----

```

```

00509  SUBROUTINE uvtranspoint(lat1, lon1, lat2, lon2, time1, time2, u, v)
00510
00511    USE pahm_global, ONLY : deg2rad
00512    USE utilities, ONLY : sphericaldistance
00513
00514    IMPLICIT NONE
00515
00516    ! Global variables
00517    REAL(sz), INTENT(IN) :: lat1, lon1, lat2, lon2
00518    REAL(sz), INTENT(IN) :: time1, time2
00519    REAL(sz), INTENT(OUT) :: u, v
00520
00521    ! Local variables
00522    REAL(sz) :: dx, dy, dt
00523    LOGICAL :: dtok
00524
00525    dx = sphericaldistance(lat1, lon1, lat1, lon2)
00526    dy = sphericaldistance(lat1, lon1, lat2, lon1)
00527    dt = abs(time2 - time1)
00528    dtok = (comparereals(dt, 0.0_sz) /=0)
00529
00530    u = 0.0_sz
00531    v = 0.0_sz
00532    IF (dtok) THEN
00533      u = sign(dx / dt, (lon2 - lon1))
00534      v = sign(dy / dt, (lat2 - lat1))
00535    END IF
00536
00537  END SUBROUTINE uvtranspoint
00538
00539 !=====
00540
00541 !-----
00542 !----- S U B R O U T I N E   N E W   V O R T E X
00543 !-----
00544 !-----
00562
00563  SUBROUTINE newvortex(pinf, p0, lat, lon, vm)
00564
00565    USE pahm_global, ONLY : rhoair, deg2rad, omega, mb2pa, kt2ms
00566
00567    IMPLICIT NONE
00568
00569    REAL(sz), INTENT(IN) :: pinf
00570    REAL(sz), INTENT(IN) :: p0
00571    REAL(sz), INTENT(IN) :: lat
00572    REAL(sz), INTENT(IN) :: lon
00573    REAL(sz), INTENT(IN) :: vm
00574
00575    ! set instance variables
00576    pn = pinf
00577    pc = p0
00578    clat = lat
00579    clon = lon
00580    vmax = vm
00581 !PV Check conversions
00582 ! evaluate basic physical params
00583 corio = 2.0_sz * omega * sin(deg2rad * clat)
00584 b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
00585 b = max(min(b, 2.0_sz), 1.0_sz) ! limit B to range 1.0->2.5
00586 !PV Data already have been converted
00587 ! added for compatibility of CalcRMaxes to use with simplified nws20
00588 bs(1:6) = b
00589 vmb1(1:6) = vmax
00590
00591  END SUBROUTINE newvortex
00592
00593 !=====
00594
00595 !-----
00596 !----- S U B R O U T I N E   N E W   V O R T E X   F U L L
00597 !-----
00616
00617  SUBROUTINE newvortexfull(pinf, p0, lat, lon, vm)
00618
00619    USE pahm_global, ONLY : rhoair, deg2rad, kt2ms, omega, mb2pa
00620
00621    IMPLICIT NONE
00622
00623    REAL(sz), INTENT(IN) :: pinf
00624    REAL(sz), INTENT(IN) :: p0
00625    REAL(sz), INTENT(IN) :: lat

```

```

00626    REAL(sz), INTENT(IN) :: lon
00627    REAL(sz), INTENT(IN) :: vm
00628
00629    ! set instance variables
00630    pn = pinf
00631    pc = p0
00632    clat = lat
00633    clon = lon
00634    vmax = vm
00635
00636    ! evaluate basic physical params
00637    corio = 2.0_sz * omega * sin(deg2rad * clat)
00638    b = (vmax * kt2ms)**2 * rhoair * exp(1.0_sz) / ((pn - pc) * mb2pa)
00639    phi = 1.0_sz
00640    bs(1:6) = b
00641    phis(1:6) = phi
00642    vmb1(1:6) = vmax
00643
00644    ! B = MAX(MIN(B, 2.0_SZ), 1.0_SZ) ! limit B to range 1.0->2.5
00645
00646 END SUBROUTINE newvortexfull
00647
00648 !=====
00649 !-----
00650 !----- S U B R O U T I N E   S E T   V O R T E X
00651 !----- S U B R O U T I N E   S E T   R M A X E S
00652 !-----
00669 !-----
00670 SUBROUTINE setvortex(pinf, p0, lat, lon)
00671
00672 USE pahm_global, ONLY : deg2rad, omega
00673
00674 IMPLICIT NONE
00675
00676    REAL(sz), INTENT(IN) :: pinf
00677    REAL(sz), INTENT(IN) :: p0
00678    REAL(sz), INTENT(IN) :: lat
00679    REAL(sz), INTENT(IN) :: lon
00680
00681    ! set instance variables
00682    pn = pinf
00683    pc = p0
00684    clat = lat
00685    clon = lon
00686
00687    ! evaluate basic physical params
00688    corio = 2.0_sz * omega * sin(deg2rad * clat)
00689
00690 END SUBROUTINE setvortex
00691
00692 !=====
00693
00694 !-----
00695 !----- S U B R O U T I N E   S E T   R M A X E S
00696 !-----
00697 SUBROUTINE setrmaxes(rMaxW)
00698
00699 IMPLICIT NONE
00700
00701    REAL(sz), DIMENSION(4), INTENT(IN) :: rmaxw
00702    INTEGER :: i
00703
00704    DO i = 1, 4
00705        rmaxes(i + 1) = rmaxw(i)
00706    END DO
00707
00708 END SUBROUTINE setrmaxes
00709
00710 !=====
00711
00712 !-----
00713 !----- S U B R O U T I N E   G E T   R M A X E S
00714 !-----
00715 SUBROUTINE getrmaxes(rMaxW)
00716
00717 IMPLICIT NONE
00718
00719    REAL(sz), DIMENSION(4), INTENT(OUT) :: rmaxw
00720
00721    INTEGER :: i
00722

```

```

00723      DO i = 1, 4
00724         rmaxw(i) = rmaxes(i + 1)
00725     END DO
00726
00727   END SUBROUTINE getrmaxes
00728
00729 !=====
00730
00731 !-----
00732 ! S U B R O U T I N E   C A L C   R M A X E S
00733 !-----
00742 !-----
00743 SUBROUTINE calcrmaxes()
00744
00745   IMPLICIT NONE
00746
00747   REAL(sz)          :: root          ! Radius of maximum winds
00748   REAL(sz), PARAMETER :: innerradius = 1.0_sz
00749   REAL(sz), PARAMETER :: outerradius = 400.0_sz
00750   REAL(sz), PARAMETER :: accuracy    = 0.0001_sz
00751   REAL(sz), PARAMETER :: zoom        = 0.01_sz
00752   INTEGER , PARAMETER :: itermax    = 3
00753   REAL(sz)          :: r1, r2, r3, r4, dr
00754   REAL(sz)          :: vicinity
00755   INTEGER            :: n, iter
00756
00757 !-----
00758 ! Loop over quadrants of storm
00759 !-----
00760 DO n = 1, nquads
00761   ! set B and vMax values for each quadrant
00762   ! for nws19, B and vMax are constant
00763   ! for simplified nws20, B is constant, while vMax is not
00764   b = bs(n + 1)
00765   vmax = vmb1(n + 1)
00766
00767   quad = n
00768   root = -1.0_sz
00769   r1   = innerradius
00770   r2   = outerradius
00771   dr   = 1.0_sz
00772   DO iter = 1, itermax
00773     root = findroot(vhwithcori, r1, r2, dr, r3, r4)
00774     r1 = r3
00775     r2 = r4
00776     dr = dr * zoom
00777   END DO
00778
00779   ! determine if rMax is actually in the vicinity of the
00780   ! isotach radius that we are using to solve for rMax,
00781   ! and if so, take another shot at finding the
00782   ! rMax using the gradient wind balance that neglects
00783   ! coriolis (and is appropriate in the vicinity of rMax)
00784   !
00785   ! CompareReals(r1, r2)
00786   !   -1 (if r1 < r2)
00787   !   0 (if r1 = r2)
00788   !   +1 (if r1 > r2)
00789   vicinity = abs(root - radius(quad)) / root
00790   !PV DEL IF ((root < 0.0_sz) .OR. (vicinity <= 0.1_sz)) THEN
00791   IF (comparereals(root, 0.0_sz) == -1 .OR. comparereals(vicinity, 0.1_sz) /= 1) THEN
00792     r1 = innerradius
00793     r2 = outerradius
00794     dr = 1.0_sz
00795     DO iter = 1, itermax
00796       root = findroot(vhnocori, r1, r2, dr, r3, r4)
00797       r1 = r3
00798       r2 = r4
00799       dr = dr * zoom
00800     END DO
00801   END IF
00802
00803   rmaxes(n + 1) = root
00804 END DO
00805
00806 END SUBROUTINE calcrmaxes
00807
00808 !=====
00809 !-----
00810 ! S U B R O U T I N E   C A L C   R M A X E S   F U L L
00811

```

```

00812 !-----
00822 !-----
00823 SUBROUTINE calcrmaxesfull()
00824
00825 USE pahm_global, ONLY : rhoair, nm2m, kt2ms, mb2pa
00826
00827 IMPLICIT NONE
00828
00829 REAL(sz) :: root          ! Radius of maximum winds
00830 REAL(sz), PARAMETER :: innerradius = 1.0_sz
00831 REAL(sz), PARAMETER :: outerradius = 500.0_sz
00832 REAL(sz), PARAMETER :: accuracy   = 0.0001_sz
00833 REAL(sz), PARAMETER :: zoom       = 0.01_sz
00834 INTEGER, PARAMETER :: itermax    = 3
00835 REAL(sz) :: r1, r2, r3, r4, dr
00836 INTEGER :: n, iter, norootflag
00837 REAL(sz) :: bnew, bnewl
00838 REAL(sz) :: phinew
00839 INTEGER, PARAMETER :: cont = 400      ! Max # of iterations
00840 INTEGER :: icont, ibcont ! iteration counter
00841
00842 !-----
00843 ! Loop over quadrants of storm
00844 !-----
00845 DO n = 1, nquads
00846   norootflag = 0
00847
00848   ! initialize B and phi values for each quadrant
00849   b = bs(n + 1)
00850   phi = phis(n + 1)
00851   vmax = vmb1(n + 1)
00852
00853   ! Loop the root-solving process to converge B, for in the
00854   ! new wind formulation, B is a function of rMax, vMax, f, and phi
00855   DO icont = 1, cont ! logical expre. is at the end to exit the loop
00856     norootflag = 0
00857     quad = n
00858     root = -1.0_sz
00859     r1 = innerradius
00860     r2 = outerradius
00861     dr = 1.0_sz
00862
00863     DO iter = 1, itermax
00864       root = findroot(vhwithcorifull, r1, r2, dr, r3, r4)
00865       r1 = r3
00866       r2 = r4
00867       dr = dr * zoom
00868     END DO
00869
00870     ! Avoid invalid B value when root is not found
00871     IF (root < 0.0_sz) THEN
00872       ! r1 = INNERADIUS
00873       ! r2 = OUTERADIUS
00874       ! dr = 1.0_SZ
00875       ! DO iter = 1, ITERMAX
00876       !   root = FindRoot(VhNoCori, r1, r2, dr, r3, r4)
00877       !   r1 = r3
00878       !   r2 = r4
00879       !   dr = dr * ZOOM
00880     ! END DO
00881     root = 1.0_sz * radius(quad)
00882     norootflag = 1
00883   END IF
00884
00885   rmaxes(n + 1) = root
00886
00887   ! Determine if B converges, if yes, break loop and assign
00888   ! values to rMaxes, if not, continue the loop to re-calculate
00889   ! root and re-evaluate bs
00890   phinew = 1 + vmax * kt2ms * root * nm2m * corio / &
00891   (b * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio))
00892   bnew = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) * &
00893   rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa) &
00894   DO ibcont = 1, cont
00895     bnewl = bnew
00896     phinew = 1 + vmax * kt2ms * root * nm2m * corio / &
00897     (bnew * ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio))
00898     bnew = ((vmax * kt2ms)**2 + vmax * kt2ms * root * nm2m * corio) * &
00899     rhoair * exp(phinew) / (phinew * (pn - pc) * mb2pa) &
00900   IF (abs(bnew - bnewl) <= 0.01_sz) EXIT

```

```

00902      END DO
00903
00904      ! IF (ibCont >= cont) THEN
00905      !   WRITE(1111, '(a7, x ,i2, x, a38)') "iquad=", n, "bNew did not fully converge, procede"
00906      !END IF
00907
00908      ! CompareReals(r1, r2)
00909      !   -1 (if r1 < r2)
00910      !   0 (if r1 = r2)
00911      !   +1 (if r1 > r2)
00912      !PV DEL IF ABS(B - bNew) <= 0.01_SZ EXIT
00913      IF (comparereals(abs(b - bnew), 0.01_sz) /= 1) EXIT
00914
00915      ! update B and phi for next iteration
00916      ! warning: modifications made here also affect other subroutines
00917      b   = bnew
00918      phi = phinew
00919      END DO !iCont = 1, cont
00920
00921      ! update to the latest values for aswip output
00922      bs(n + 1)   = bnew
00923      phis(n + 1 ) = phinew
00924
00925      !IF (iCont >= cont) THEN
00926      !   WRITE(1111, '(a7, x ,i2, x, a38)') "iquad=", n, "B did not fully converge, procede"
00927      !END IF
00928
00929      ! Determine if rMax is actually in the vicinity of the
00930      ! isotach radius that we are using to solve for rMax,
00931      ! and if so, take another shot at finding the
00932      ! rMax using the gradient wind equation that neglects
00933      ! coriolis (and is appropriate in the vicinity of rMax)
00934      !vicinity = ABS(root - radius(quad)) / root
00935      IF (norootflag == 1) THEN
00936          WRITE(*, *) "iquad=", n, "No root found, return dist. to Isotach"
00937      END IF
00938      END DO !n = 1, nQuads
00939
00940  END SUBROUTINE calcrmaxesfull
00941
00942 !=====
00943
00944 !-----
00945 ! S U B R O U T I N E   F I T   R M A X E S
00946 !-----
00947
00948 !-----
00949 SUBROUTINE fitrmaxes()
00950
00951     IMPLICIT NONE
00952
00953     ! Generate 2 additional (theta, rMax) points for curve-fit
00954     rmaxes(1) = rmaxes(5)
00955     rmaxes(6) = rmaxes(2)
00956
00957     END SUBROUTINE fitrmaxes
00958
00959 !=====
00960
00961 !-----
00962 ! S U B R O U T I N E   F I T   R M A X E S   4
00963 !-----
00964
00965 SUBROUTINE fitrmaxes4()
00966
00967     IMPLICIT NONE
00968
00969     ! Generate 2 additional points for curve-fit
00970     quadflag4(1, 1:4) = quadflag4(5, 1:4)
00971     quadflag4(6, 1:4) = quadflag4(2, 1:4)
00972
00973     quadir4(1, 1:4) = quadir4(5, 1:4)
00974     quadir4(6, 1:4) = quadir4(2, 1:4)
00975
00976     rmaxes4(1, 1:4) = rmaxes4(5, 1:4)
00977     rmaxes4(6, 1:4) = rmaxes4(2, 1:4)
00978
00979     bs4(1, 1:4) = bs4(5, 1:4)
00980     bs4(6, 1:4) = bs4(2, 1:4)
00981
00982     phis4(1, 1:4) = phis4(5, 1:4)
00983     phis4(6, 1:4) = phis4(2, 1:4)
00984
00985
00986
00987
00988
00989
00990

```

```

00991      vmb14(1, 1:4) = vmb14(5, 1:4)
00992      vmb14(6, 1:4) = vmb14(2, 1:4)
00993
00994  END SUBROUTINE fitrmaxes4
00995
00996 !=====
00997 !-----
00998 !-----S U B R O U T I N E   S E T   V M A X E S   B L
01000 !-----
01001 SUBROUTINE setvmaxesbl(vMaxW)
01002
01003     IMPLICIT NONE
01004
01005     REAL(sz), DIMENSION(4), INTENT(IN) :: vmaxw
01006
01007     INTEGER :: i
01008
01009     DO i = 1, 4
01010         vmb1(i + 1) = vmaxw(i)
01011     END DO
01012
01013 END SUBROUTINE setvmaxesbl
01014
01015 !=====
01016
01017 !-----
01018 !-----S U B R O U T I N E   G E T   V M A X E S   B L
01019 !-----
01020 SUBROUTINE getvmaxesbl(vMaxW)
01021
01022     IMPLICIT NONE
01023
01024     REAL(sz), DIMENSION(4), INTENT(OUT) :: vmaxw
01025
01026     INTEGER :: i
01027
01028     DO i = 1, 4
01029         vmaxw(i) = vmb1(i + 1)
01030     END DO
01031
01032 END SUBROUTINE getvmaxesbl
01033
01034 !=====
01035
01036 !-----
01037 !-----S U B R O U T I N E   S E T   U S E   V M A X E S   B L
01038 !-----
01039 SUBROUTINE setusevmaxesbl(u)
01040
01041     IMPLICIT NONE
01042
01043     LOGICAL, INTENT(IN) :: u
01044
01045     usevmaxesbl = u
01046
01047 END SUBROUTINE setusevmaxesbl
01048
01049 !-----
01050 !-----S U B R O U T I N E   S E T   S H A P E   P A R A M E T E R
01051 !-----
01052 SUBROUTINE setshapeparameter(param)
01053
01054     IMPLICIT NONE
01055
01056     REAL(sz) :: param
01057
01058     b = param
01059
01060 END SUBROUTINE setshapeparameter
01061
01062 !=====
01063
01064 !-----
01065 !-----F U N C T I O N   G E T   S H A P E   P A R A M E T E R
01066 !-----
01067 REAL(sz) function getshapeparameter() result(myvalout)
01068
01069     IMPLICIT NONE
01070
01071     myvalout = b

```

```
01072      RETURN
01073
01074
01075      END FUNCTION getshapeparameter
01076
01077 !=====
01078
01079 !-----
01080 !  F U N C T I O N   G E T   S H A P E   P A R A M E T E R S
01081 !
01082      FUNCTION getshapeparameters() RESULT(myValOut)
01083
01084      IMPLICIT NONE
01085
01086      REAL(sz), DIMENSION(4) :: myvalout
01087
01088      INTEGER :: i
01089
01090      DO i = 1, 4
01091         myvalout(i) = bs(i + 1)
01092     END DO
01093
01094     RETURN
01095
01096     END FUNCTION getshapeparameters
01097
01098 !=====
01099
01100 !-----
01101 !  F U N C T I O N   G E T   P H I   F A C T O R S
01102 !
01103      FUNCTION getphifactors() RESULT(myValOut)
01104
01105      IMPLICIT NONE
01106
01107      REAL(sz), DIMENSION(4) :: myvalout
01108
01109      INTEGER :: i
01110
01111      DO i = 1, 4
01112         myvalout(i) = phis(i + 1)
01113     END DO
01114
01115     RETURN
01116
01117     END FUNCTION getphifactors
01118
01119 !=====
01120
01121 !-----
01122 !  S U B R O U T I N E   S E T   I S O T A C H   R A D I I
01123 !
01124      SUBROUTINE setisotachradii(ir)
01125
01126      IMPLICIT NONE
01127
01128      REAL(sz), DIMENSION(4), INTENT(IN) :: ir
01129
01130      radius(:) = ir(:)
01131
01132      END SUBROUTINE setisotachradii
01133
01134 !=====
01135
01136 !-----
01137 !  S U B R O U T I N E   S E T   I S O T A C H   W I N D   S P E E D S
01138 !
01139      SUBROUTINE setisotachwindspeeds(vrq)
01140
01141      IMPLICIT NONE
01142
01143      REAL(sz), DIMENSION(4), INTENT(IN) :: vrq
01144
01145      vrquadrant(:) = vrq(:)
01146
01147      END SUBROUTINE setisotachwindspeeds
01148
01149 !=====
01150
01151 !-----
01152 !  S U B R O U T I N E   S E T   I S O T A C H   W I N D   S P E E D
```

```

01153 ! -----
01154 SUBROUTINE setisotachwindspeed(sp)
01155
01156     IMPLICIT NONE
01157
01158     REAL(sz), INTENT(IN) :: sp
01159
01160     vr = sp
01161
01162 END SUBROUTINE setisotachwindspeed
01163
01164 ! =====
01165
01166 ! -----
01167 ! S U B R O U T I N E   S E T   U S E   Q U A D R A N T   V R
01168 ! -----
01169 SUBROUTINE setusequadrantvr(u)
01170
01171     IMPLICIT NONE
01172
01173     LOGICAL, INTENT(IN) :: u
01174
01175     usequadrantvr = u
01176
01177 END SUBROUTINE setusequadrantvr
01178
01179 ! =====
01180
01181 ! -----
01182 ! F U N C T I O N   G E T   L A T E S T   A N G L E
01183 ! -----
01184 LOGICAL FUNCTION getusequadrantvr() RESULT(myValOut)
01185
01186     IMPLICIT NONE
01187
01188     myvalout = usequadrantvr
01189
01190 END FUNCTION getusequadrantvr
01191
01192 ! =====
01193
01194 ! -----
01195 ! F U N C T I O N   S P I N T E R P
01196 ! -----
01197 ! INTEGER validIsot is used as a marker to indicate how many isotachs
01198 ! are available in a certain quadrant
01199 ! SELECT CASE(validIsot)
01200 ! CASE(1): 1 situation
01201 ! CASE(2): 3 situations
01202 ! CASE(3): 4 situations
01203 ! CASE(4): 5 situations
01204 ! -----
01205
01206 REAL(sz) function spinterp(angle, dist, opt) result(myvalout)
01207
01208     IMPLICIT NONE
01209
01210     REAL(sz), INTENT(IN)          :: angle, dist
01211     INTEGER, INTENT(IN)          :: opt
01212     REAL(sz), DIMENSION(NPOINTS, 4) :: param
01213     REAL(sz)                      :: templ, temp2
01214     REAL(sz)                      :: deltaangle
01215     INTEGER                        :: iquad
01216
01217     IF (opt == 1) THEN
01218         param = rmaxes4
01219     ELSE IF (opt == 2) THEN
01220         param = bs4
01221     ELSE IF (opt == 3) THEN
01222         param = vmb14
01223     END IF
01224
01225     deltaangle = 0.0_sz
01226
01227     ! CompareReals(r1, r2)
01228     !   -1 (if r1 < r2)
01229     !   0 (if r1 = r2)
01230     !   +1 (if r1 > r2)
01231     IF (comparereals(angle, 45.0_sz) /= 1) THEN
01232         iquad = 5
01233         deltaangle = 45.0_sz + angle
01234     ELSE IF (comparereals(angle, 135.0_sz) /= 1) THEN
01235

```

```

01251      iquad = 2
01252      deltaangle = angle - 45.0_sz
01253      ELSE IF (comparereals(angle, 225.0_sz) /= 1) THEN
01254          iquad = 3
01255          deltaangle = angle - 135.0_sz
01256      ELSE IF (comparereals(angle, 315.0_sz) /= 1) THEN
01257          iquad = 4
01258          deltaangle = angle - 225.0_sz
01259      ELSE IF (angle > 315.0_sz) THEN
01260          iquad = 5
01261          deltaangle = angle - 315.0_sz
01262  END IF
01263
01264      ! nearest neighbor weighted interpolation
01265      IF (deltaangle < 1.0_sz) THEN
01266          myvalout = interp(param, iquad, dist)
01267      ELSE IF (deltaangle > 89.0_sz) THEN
01268          myvalout = interp(param, iquad + 1, dist)
01269      ELSE
01270          temp1 = interp(param, iquad, dist)
01271          temp2 = interp(param, iquad + 1, dist)
01272          myvalout = (temp1 / deltaangle**2 + temp2 / (90.0 - deltaangle)**2) /   &
01273                  (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
01274  END IF
01275
01276  END FUNCTION spinterp
01277
01278 !=====
01279 !-----
01280 !----- F U N C T I O N   I N T E R P R -----
01281 !-----
```

REAL(sz) function interp(quadval, quadsel, quaddis) result(myvalout)

IMPLICIT NONE

REAL(sz), DIMENSION(NPOINTS, 4), INTENT(IN) :: quadval
 INTEGER, INTENT(IN) :: quadsel
 REAL(sz), INTENT(IN) :: quaddis

REAL(sz) :: fac
 INTEGER :: totalisot

totalisot = sum(quadflag4(quadsel, :))

SELECT CASE(totalisot)

CASE(1)
 myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))

CASE(2)
 IF (quaddis > quadir4(quadsel, 1)) THEN
 myvalout = quadval(quadsel, 1)
 ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
 fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
 myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
 ELSE
 myvalout = quadval(quadsel, 2)
 END IF

CASE(3)
 IF (quaddis > quadir4(quadsel, 1)) THEN
 myvalout = quadval(quadsel, 1)
 ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
 fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
 myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
 ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
 fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
 myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
 ELSE
 myvalout = quadval(quadsel, 3)
 END IF

CASE(4)
 IF (quaddis > quadir4(quadsel, 1)) THEN
 myvalout = quadval(quadsel, 1)
 ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
 fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
 myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
 ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
 fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
 myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
 ELSE IF (quaddis > quadir4(quadsel, 4)) THEN
 fac = (quaddis - quadir4(quadsel, 4)) / (quadir4(quadsel, 3) - quadir4(quadsel, 4))
 myvalout = quadval(quadsel, 3) * fac + quadval(quadsel, 4) * (1 - fac)
 ELSE

```

01332      myvalout = quadval(quadsel, 4)
01333      END IF
01334      CASE default
01335          ! For whatever reason if our algorithm fails, add the following
01336          ! line to avoid run-time errors
01337          myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :, :), 1))
01338          !WRITE(*, *) "ERROR: InterpR failed in nws20get." !PV remove it of modify it?
01339      END SELECT
01340
01341  END FUNCTION interpr
01342
01343 !=====
01344 !-----
01345 !----- F U N C T I O N   R M W -----
01346 !----- !
01347 !----- !
01361 !----- !
01362 REAL(sz) function rmw(angle) result(myvalout)
01363
01364     IMPLICIT NONE
01365
01366     REAL(sz), INTENT(IN) :: angle
01367     INTEGER                 :: basequadrant
01368     REAL(sz)                :: deltaangle
01369
01370     deltaangle = 0.0_sz
01371     basequadrant = 5
01372
01373     ! CompareReals(r1, r2)
01374     !    -1 (if r1 < r2)
01375     !    0 (if r1 = r2)
01376     !    +1 (if r1 > r2)
01377     IF (comparereals(angle, 45.0_sz) /= 1) THEN
01378         basequadrant = 5
01379         deltaangle = 45.0_sz + angle
01380     ELSE IF (comparereals(angle, 135.0_sz) /= 1) THEN
01381         basequadrant = 2
01382         deltaangle = angle - 45.0_sz
01383     ELSE IF (comparereals(angle, 225.0_sz) /= 1) THEN
01384         basequadrant = 3
01385         deltaangle = angle - 135.0_sz
01386     ELSE IF (comparereals(angle, 315.0_sz) /= 1) THEN
01387         basequadrant = 4
01388         deltaangle = angle - 225.0_sz
01389     ELSE IF (angle > 315.0_sz) THEN
01390         basequadrant = 5
01391         deltaangle = angle - 315.0_sz
01392     END IF
01393
01394     ! nearest neighbor weighted interpolation
01395     IF (deltaangle < 1.0_sz) THEN
01396         myvalout = rmaxes(basequadrant) ! avoid div by zero
01397     ELSE IF (deltaangle > 89.0_sz) THEN
01398         myvalout = rmaxes(basequadrant + 1) ! avoid div by zero
01399     ELSE
01400         myvalout = (rmaxes(basequadrant) / deltaangle**2 + &
01401                     rmaxes(basequadrant + 1) / (90.0 - deltaangle)**2) / &
01402                     (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
01403     END IF
01404
01405     ! linearly interpolate
01406     !myValOut = (deltaAngle / 90.0_SZ) * &
01407     !            (rMxes(baseQuadrant + 1) - rMxes(baseQuadrant)) + &
01408     !            rMxes(baseQuadrant)
01409
01410  END FUNCTION rmw
01411
01412 !=====
01413 !-----
01414 !----- S U B R O U T I N E   U V P -----
01415 !----- !
01416 !----- !
01440 !----- !
01441 SUBROUTINE uvp(lat, lon, uTrans, vTrans, u, v, p)
01442
01443     USE pahm_global, ONLY : windreduction, one2ten, deg2rad, rad2deg, mb2pa, kt2ms, nm2m, m2nm, rearth
01444
01445     IMPLICIT NONE
01446
01447     REAL(sz), INTENT(IN) :: lat
01448     REAL(sz), INTENT(IN) :: lon

```

```

01449    REAL(sz), INTENT(IN) :: utrans
01450    REAL(sz), INTENT(IN) :: vtrans
01451
01452    REAL(sz), INTENT(OUT) :: u
01453    REAL(sz), INTENT(OUT) :: v
01454    REAL(sz), INTENT(OUT) :: p
01455
01456    REAL(sz)           :: transspdx !NWS8-style translation speed
01457    REAL(sz)           :: transspdy !NWS8-style translation speed
01458
01459    REAL(sz)           :: dx
01460    REAL(sz)           :: dy
01461    REAL(sz)           :: dist
01462    REAL(sz)           :: rmx
01463    REAL(sz)           :: angle
01464    REAL(sz)           :: speed
01465    REAL(sz)           :: uf
01466    REAL(sz)           :: vf
01467    REAL(sz)           :: percentcoriolis
01468    REAL(sz)           :: speedatrmx
01469    REAL(sz)           :: vmaxfactor
01470
01471 !-----
01472 ! Calculate distance and angle between eye of hurricane
01473 ! and input nodal point
01474 !-----
01475 dx = deg2rad * rearth * (lon - clon) * cos(deg2rad * clat)
01476 dy = deg2rad * rearth * (lat - clat)
01477 dist = sqrt(dx * dx + dy * dy)
01478
01479 !-----
01480 ! Handle special case at eye of hurricane
01481 ! in eye velocity is zero not translational velocity
01482 !-----
01483 IF (dist < 1.0_sz) THEN
01484   u = 0.0_sz
01485   v = 0.0_sz
01486   p = pc * mb2pa
01487
01488   RETURN
01489 END IF
01490
01491 dist = m2nm * dist
01492
01493 angle = 360.0_sz + rad2deg * atan2(dx, dy)
01494 IF (angle > 360.0_sz) angle = angle - 360.0_sz
01495
01496 latestangle = angle
01497 rmx = rmw(angle)
01498 latestrmx = rmx
01499
01500 !-----
01501 ! Compute (u,v) wind velocity components from the
01502 ! asymmetric hurricane vortex.
01503 !
01504 ! Note: the vortex winds are valid at the top of the
01505 ! surface layer, so reduce the winds to the surface.
01506 ! Also convert the winds from max sustained 1-minute
01507 ! averages to 10-minute averages for the storm surge
01508 ! model.
01509 !-----
01510 percentcoriolis = 1.0_sz
01511 speed = sqrt((vmax * kt2ms)**2 * (rmx / dist)**b * exp(1.0_sz - (rmx / dist)**b) + &
01512   (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2) &
01513   - nm2m * dist * percentcoriolis * corio / 2.0_sz
01514
01515 ! Calculate the wind speed (m/s) at rMax, using
01516 ! equation that includes full coriolis
01517 speedatrmx = sqrt((vmax * kt2ms)**2 * exp(0.0_sz) + &
01518   (nm2m * dist * percentcoriolis * corio / 2.0_sz)**2) &
01519   - nm2m * dist * percentcoriolis * corio / 2.0_sz
01520
01521 ! Calculate a factor to place the velocity profile so that
01522 ! it hits vMax
01523 vmaxfactor = vmax * kt2ms / speedatrmx
01524
01525 ! Calculate NWS8-like translation speed
01526 transspdx = (abs(speed / speedatrmx) * utrans * kt2ms
01527 transspdy = (abs(speed / speedatrmx) * vtrans * kt2ms
01528
01529 speed = speed * vmaxfactor

```

```

01530
01531      ! Now reduce the wind speed to the surface
01532      speed = speed * windreduction
01533
01534      u = -speed * cos(deg2rad * angle)
01535      v = speed * sin(deg2rad * angle)
01536
01537      ! Alter wind direction by adding a frictional inflow angle
01538      CALL rotate(u, v, fang(dist, rmx), clat, uf, vf)
01539      u = uf
01540      v = vf
01541      !
01542      ! jgf20111007: Add in the translation velocity
01543      u = u + transspdX
01544      v = v + transspdY
01545      !
01546      ! convert from 1 minute averaged winds to 10 minute averaged
01547      ! winds for use in ADCIRC
01548      u = u * one2ten
01549      v = v * one2ten
01550
01551      ! Compute surface pressure from asymmetric hurricane vortex
01552      p = mb2pa * (pc + (pn - pc) * exp(-(rmx / dist)**b))
01553
01554      ! cut off the vortex field after 401nm !PV Attend to this
01555      ! TODO: 401nm should be replaced with something less
01556      ! arbitrary ... and find a better way to blend this
01557      !IF ( dist > 401.0_SZ ) THEN
01558      !   u = 0.0_SZ
01559      !   v = 0.0_SZ
01560      !   p = MB2PA * pn
01561      !END IF
01562
01563      END SUBROUTINE uvP
01564
01565 !=====
01566
01567
01568 !-----
01569 ! S U B R O U T I N E   U V P R
01570 !-----
01607 !-----
01608 SUBROUTINE uvpr(iDist, iAngle, iRmx, iRmxTrue, iB, iVm, iPhi, &
01609           uTrans, vTrans, geof, u, v, p)
01610
01611 USE pahm_global, ONLY : windreduction, one2ten, deg2rad, mb2pa, kt2ms, nm2m
01612
01613 IMPLICIT NONE
01614
01615 REAL(sz), INTENT(IN) :: idist
01616 REAL(sz), INTENT(IN) :: iangle
01617 REAL(sz), INTENT(IN) :: irmx
01618 REAL(sz), INTENT(IN) :: irmxtrue
01619 REAL(sz), INTENT(IN) :: ib
01620 REAL(sz), INTENT(IN) :: ivm
01621 REAL(sz), INTENT(IN) :: iphi
01622 REAL(sz), INTENT(IN) :: utrans
01623 REAL(sz), INTENT(IN) :: vtrans
01624 INTEGER, INTENT(IN) :: geof
01625
01626 REAL(sz), INTENT(OUT) :: u
01627 REAL(sz), INTENT(OUT) :: v
01628 REAL(sz), INTENT(OUT) :: p
01629
01630 REAL(sz)          :: transspdX !NWS8-style translation speed
01631 REAL(sz)          :: transspdY !NWS8-style translation speed
01632 REAL(sz)          :: rmx
01633 REAL(sz)          :: speed
01634 REAL(sz)          :: uf
01635 REAL(sz)          :: vf
01636 REAL(sz)          :: percentcoriolis
01637
01638 rmx = irmx
01639 b = ib
01640 vmax = ivm
01641 phi = iphi
01642
01643 !-----
01644 ! Handle special case at eye of hurricane
01645 ! in eye velocity is zero not translational velocity
01646 !-----

```

```

01647 IF (idist < 1.0_sz) THEN
01648   u = 0.0_sz
01649   v = 0.0_sz
01650   p = pc * mb2pa
01651
01652   RETURN
01653 END IF
01654
01655 !-----
01656 ! Compute (u, v) wind velocity components from the
01657 ! asymmetric hurricane vortex.
01658 !
01659 ! Note: the vortex winds are valid at the top of the
01660 ! surface layer, so reduce the winds to the surface.
01661 ! Also convert the winds from max sustained 1-minute
01662 ! averages to 10-minute averages for the storm surge
01663 ! model.
01664 !-----
01665 percentcoriolis = 1.0_sz
01666
01667 IF (geof == 1) THEN
01668   speed = sqrt((vmax * kt2ms)**2 + vmax * kt2ms * rmx * nm2m * percentcoriolis * corio) * &
01669     (rmx / idist)**b * exp(phi * (1.0_sz - (rmx / idist)**b)) + &
01670     (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) - &
01671     nm2m * idist * percentcoriolis * corio / 2.0_sz
01672 ELSE
01673   speed = sqrt((vmax * kt2ms)**2 * (rmx / idist)**b * exp(1.0_sz - (rmx / idist)**b) + &
01674     (nm2m * idist * percentcoriolis * corio / 2.0_sz)**2) - &
01675     nm2m * idist * percentcoriolis * corio / 2.0_sz
01676 ENDIF
01677
01678 ! Calculate NWS8-like translation speed
01679 transspd_x = (abs(speed / (vmax * kt2ms))) * utrans * kt2ms
01680 transspd_y = (abs(speed / (vmax * kt2ms))) * vtrans * kt2ms
01681
01682 ! Now reduce the wind speed to the surface
01683 speed = speed * windreduction
01684
01685 u = -speed * cos(deg2rad * iangle)
01686 v = speed * sin(deg2rad * iangle)
01687
01688 ! Alter wind direction by adding a frictional inflow angle
01689 CALL rotate(u, v, fang(idist, irmxtrue), clat, uf, vf)
01690 u = uf
01691 v = vf
01692
01693 ! Add in the translation velocity
01694 u = u + transspd_x
01695 v = v + transspd_y
01696
01697 ! convert from 1 minute averaged winds to 10 minute averaged
01698 ! winds for use in ADCIRC
01699 u = u * one2ten
01700 v = v * one2ten
01701
01702 ! Compute surface pressure from asymmetric hurricane vortex
01703 IF (geof == 1) THEN
01704   p = mb2pa * (pc + (pn - pc) * exp(-phi * (rmx / idist)**b))
01705 ELSE
01706   p = mb2pa * (pc + (pn - pc) * exp(-(rmx / idist)**b))
01707 ENDIF
01708
01709 ! cut off the vortex field after 401nm !PV Attend to this
01710 ! TODO: 401nm should be replaced with something less
01711 ! arbitrary ... and find a better way to blend this
01712 !if ( dist > 401.0_SZ ) then
01713 !u = 0.0_SZ
01714 !v = 0.0_SZ
01715 !p = MB2PA * pn
01716 !endif
01717
01718 END SUBROUTINE uvpr
01719
01720 !-----
01721
01722 !-----
01723 ! F U N C T I O N   F A N G
01724 !
01725 !
01726 !
01727 !
01728 !
01729 !
01730 !
01731 !
01732 !
01733 !
01734 !
01735 !
01736 !
01737 !
01738 !
01739 !
01740 !
01741 !
01742 REAL(sz) function fang(r, rmx) result(myvalout)
01743

```

```

01744      IMPLICIT NONE
01745
01746      REAL(sz), INTENT(IN) :: r
01747      REAL(sz), INTENT(IN) :: rmx
01748
01749      ! CompareReals(r1, r2)
01750      ! -1 (if r1 < r2)
01751      ! 0 (if r1 = r2)
01752      ! +1 (if r1 > r2)
01753      IF (comparereals(0.0_sz, r) /= 1 .AND. comparereals(r, rmx) == -1) THEN
01754         myvalout = 10.0_sz * r / rmx
01755      ELSE IF (comparereals(rmx, r) /= 1 .AND. comparereals(r, 1.2_sz * rmx) == -1) THEN
01756         myvalout = 10.0_sz + 75.0_sz * (r / rmx - 1.0_sz)
01757      ELSE IF (comparereals(r, 1.2_sz * rmx) /= -1) THEN
01758         myvalout = 25.0_sz
01759      ELSE
01760         myvalout = 0.0_sz
01761      END IF
01762
01763  END FUNCTION fang
01764
01765 !=====
01766
01767 !-----
01768 ! S U B R O U T I N E   R O T A T E
01769 !-----
01794 !
01795 SUBROUTINE rotate(x, y, angle, whichWay, xr, yr)
01796
01797     USE pahm_global, ONLY : deg2rad
01798
01799     IMPLICIT NONE
01800
01801     REAL(SZ), INTENT(IN) :: x
01802     REAL(SZ), INTENT(IN) :: y
01803     REAL(SZ), INTENT(IN) :: angle
01804     REAL(SZ), INTENT(IN) :: whichWay
01805
01806     REAL(SZ), INTENT(OUT) :: xr
01807     REAL(SZ), INTENT(OUT) :: yr
01808
01809     REAL(SZ) :: A, cosa, sina
01810
01811     a = sign(1.0_sz, whichway) * deg2rad * angle
01812     cosa = cos(a)
01813     sina = sin(a)
01814
01815     xr = x * cosa - y * sina
01816     yr = x * sina + y * cosa
01817
01818 END SUBROUTINE rotate
01819
01820 !=====
01821
01822 !-----
01823 ! F U N C T I O N   G E T   L A T E S T   R M A X
01824 !-----
01825 REAL(sz) function getlatestrmax() result(myvalout)
01826
01827     IMPLICIT NONE
01828
01829     myvalout = latestrmax
01830
01831 END FUNCTION getlatestrmax
01832
01833 !=====
01834
01835 !-----
01836 ! F U N C T I O N   G E T   L A T E S T   A N G L E
01837 !-----
01838 REAL(sz) function getlatestangle() result(myvalout)
01839
01840     IMPLICIT NONE
01841
01842     myvalout = latestangle
01843
01844 END FUNCTION getlatestangle
01845
01846 !=====
01847
01848 !

```

```

01849 ! F U N C T I O N   V H   W I T H   C O R I   F U L L
01850 !-----
01865 !
01866 REAL(sz) function vhwithcorifull(testrmax) result(myvalout)
01867
01868   USE pahm_global, ONLY : nm2m, kt2ms, ms2kt
01869
01870   IMPLICIT NONE
01871
01872   REAL(sz), INTENT(IN) :: testrmax
01873
01874   REAL(sz)           :: thisvr ! the radial wind speed we've been given
01875   REAL(sz)           :: vh
01876
01877 !-----
01878 ! func(x = rMax) = vh - vr
01879 !-----
01880 IF (getusequadrantvr() .EQV. .true.) THEN
01881   thisvr = vrquadrant(quad)
01882 ELSE
01883   thisvr = vr
01884 END IF
01885
01886 vh = ms2kt * (sqrt((vmax * kt2ms)**2 + vmax * kt2ms * testrmax * nm2m * corio) * &
01887           (testrmax / radius(quad))**b * &
01888           exp(phi * (1.0_sz - (testrmax / radius(quad))**b)) + &
01889           (nm2m * radius(quad) * corio / 2.0_sz)**2) - &
01890           nm2m * radius(quad) * corio / 2.0_sz)
01891
01892 myvalout = vh - thisvr
01893
01894 RETURN
01895
01896 END FUNCTION vhwithcorifull
01897
01898 !=====
01899 !
01900 !----- F U N C T I O N   V H   W I T H   C O R I -----
01901 !
01917 !
01918 REAL(sz) function vhwithcori(testrmax) result(myvalout)
01919
01920   USE pahm_global, ONLY : nm2m, kt2ms, ms2kt
01921
01922   IMPLICIT NONE
01923
01924   REAL(sz), INTENT(IN) :: testrmax
01925
01926   REAL(sz)           :: thisvr ! the radial wind speed we've been given
01927   REAL(sz)           :: vh
01928
01929 !-----
01930 ! func(x = rMax) = vh - vr
01931 !-----
01932 IF (getusequadrantvr() .EQV. .true.) THEN
01933   thisvr = vrquadrant(quad)
01934 ELSE
01935   thisvr = vr
01936 END IF
01937
01938 vh = ms2kt * (sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b * &
01939           exp(1.0_sz - (testrmax / radius(quad))**b) + &
01940           (nm2m * radius(quad) * corio / 2.0_sz)**2) - &
01941           nm2m * radius(quad) * corio / 2.0_sz)
01942
01943 myvalout = vh - thisvr
01944
01945 RETURN
01946
01947 END FUNCTION vhwithcori
01948
01949 !=====
01950 !
01951 !----- F U N C T I O N   V H   N O   C O R I -----
01952 !
01953 !
01954 REAL(sz) function vhnocori(testrmax) result(myvalout)
01955
01956   USE pahm_global, ONLY : kt2ms, ms2kt
01957

```

```

01958      IMPLICIT NONE
01959
01960      REAL(sz), INTENT(IN) :: testrmax
01961
01962      REAL(sz) :: thisvr ! the radial wind speed we've been given
01963
01964      IF (getusequadrantvr() .EQV. .true.) THEN
01965          thisvr = vrquadrant(quad)
01966      ELSE
01967          thisvr = vr
01968      END IF
01969
01970      myvalout = abs(ms2kt * sqrt((vmax * kt2ms)**2 * (testrmax / radius(quad))**b * &
01971                               exp(1 - (testrmax / radius(quad))**b))) - thisvr
01972
01973      RETURN
01974
01975  END FUNCTION vhocori
01976
01977 !=====
01978 !-----
01979 !----- F U N C T I O N   F I N D   R O O T
01980 !-----
02005 !-----
02006      REAL(sz) function findroot(func, x1, x2, dx, a, b) result(myroot)
02007 !PV Need to check for the x2 variable is not used anywhere next
02008      IMPLICIT NONE
02009
02010      REAL(sz), EXTERNAL :: func
02011      REAL(sz), INTENT(IN) :: x1, x2           ! Search interval [x1,x2]
02012      REAL(sz), INTENT(IN) :: dx               ! Marching increment
02013      REAL(sz), INTENT(OUT) :: a, b            ! x values that bracket root
02014
02015      INTEGER , PARAMETER :: itermax = 400    ! Max # of iterations
02016      INTEGER :: iter                      ! iteration counter
02017      REAL(sz) :: fa, fb                  ! function values f(x)
02018
02019      ! For the time being keep the x2 parameter, set it to
02020      ! dummy to eliminate unused variable messages from the compiler
02021      REAL(sz) :: xdummy
02022      xdummy = x2
02023
02024      ! Initialize left side of interval
02025      a = x1
02026      fa = func(a)
02027
02028      ! March along interval until root is found
02029      ! or solution diverges.
02030      myroot = a
02031      DO iter = 1, itermax
02032          b = x1 + iter * dx
02033          fb = func(b)
02034
02035          ! Check progress
02036          IF ((fa * fb < 0.0_sz) .OR. (abs(fb) > abs(fa))) THEN
02037              ! Assign root
02038              IF (abs(fb) > abs(fa)) THEN
02039                  myroot = a
02040              ELSE
02041                  myroot = b
02042              END IF
02043
02044              EXIT
02045          END IF
02046
02047          ! Move right search interval values to left side
02048          ! for next iteration.
02049          a = b
02050          fa = fb
02051      END DO
02052
02053      IF (iter >= itermax) THEN
02054          print *, "FUNCTION FindRoot: exceeded max # of iterations"
02055          myroot = -99999.0
02056      END IF
02057
02058      RETURN
02059
02060  END FUNCTION findroot
02061

```

```
02062 !=====
02063
02064 END MODULE pahm_vortex
```

Index

/home/takis/CSDL/parwinds-doc/src/csv_module.F90,
 295, 298
/home/takis/CSDL/parwinds-doc/src/csv_parameters.F90,
 314, 315
/home/takis/CSDL/parwinds-doc/src/csv_utilities.F90,
 316, 318
/home/takis/CSDL/parwinds-doc/src/driver_mod.F90, 320,
 321
/home/takis/CSDL/parwinds-doc/src/global.F90, 323, 326
/home/takis/CSDL/parwinds-doc/src/mesh.F90, 330, 331
/home/takis/CSDL/parwinds-doc/src/messages.F90, 335,
 336
/home/takis/CSDL/parwinds-doc/src/netcdfio.F90, 341,
 343
/home/takis/CSDL/parwinds-doc/src/pahm.F90, 353, 355
/home/takis/CSDL/parwinds-doc/src/parwind.F90, 356,
 357
/home/takis/CSDL/parwinds-doc/src/sizes.F90, 400, 401
/home/takis/CSDL/parwinds-doc/src/sortutils.F90, 404,
 407
/home/takis/CSDL/parwinds-doc/src/timedateutils.F90,
 426, 428
/home/takis/CSDL/parwinds-doc/src/utilities.F90, 441, 443
/home/takis/CSDL/parwinds-doc/src/vortex.F90, 474, 476

add
 csv_module::csv_file, 249
add_cell
 csv_module, 23
 csv_module::csv_file, 250
add_matrix
 csv_module, 24
 csv_module::csv_file, 250
add_vector
 csv_module, 25
 csv_module::csv_file, 250
agrid
 pahm_mesh, 79
airdensity
 pahm_global, 60
allmessage_1
 pahm_messages, 83
 pahm_messages::allmessage, 219
allmessage_2
 pahm_messages, 83
 pahm_messages::allmessage, 220
allocasymvortstruct
 parwind, 136
allocatenodalandelementalarrays
 pahm_mesh, 78
allocbtrstruct
 parwind, 136
allochollstruct
 parwind, 137
approach
 parwind, 148
arraycopydouble
 sortutils, 150
 sortutils::arraycopy, 221
arraycopyint
 sortutils, 151
 sortutils::arraycopy, 221
arraycopysingle
 sortutils, 151
 sortutils::arraycopy, 222
arrayequaldouble
 sortutils, 152
 sortutils::arrayequal, 223
arrayequalint
 sortutils, 152
 sortutils::arrayequal, 223
arrayequalsingle
 sortutils, 153
 sortutils::arrayequal, 224
arthdouble
 sortutils, 153
 sortutils::arth, 225
arthint
 sortutils, 154
 sortutils::arth, 226
arthsingle
 sortutils, 154
 sortutils::arth, 226
asyvortstru
 parwind, 148

b
 pahm_vortex, 130
backgroundatmpress
 pahm_global, 60
base_netcdfcheckerr
 pahm_netcdfio, 94
basee
 pahm_global, 60
basin
 parwind::asymmetricvortexdata_t, 229
 parwind::besttrackdata_t, 238
 parwind::hollanddata_t, 268
begdate
 pahm_global, 60
begdatespecified
 pahm_global, 61

begdatetime
 pahm_global, 61
begday
 pahm_global, 61
beghour
 pahm_global, 61
begmin
 pahm_global, 61
begmonth
 pahm_global, 61
begsec
 pahm_global, 62
begsimspecified
 pahm_global, 62
begsimtime
 pahm_global, 62
begtime
 pahm_global, 62
begyear
 pahm_global, 62
besttrackdata
 parwind, 148
besttrackfilename
 pahm_global, 62
besttrackfilenamespecified
 pahm_global, 63
blank
 pahm_sizes, 105
bs
 pahm_vortex, 130
bs4
 pahm_vortex, 130

calcintensitychange
 pahm_vortex, 110
calcrmaxes
 pahm_vortex, 111
calcrmaxesfull
 pahm_vortex, 111
casttime
 parwind::asymmetricvortexdata_t, 229
 parwind::hollanddata_t, 268
casttype
 parwind::asymmetricvortexdata_t, 229
 parwind::hollanddata_t, 268
casttypenum
 parwind::asymmetricvortexdata_t, 229

charunique
 utilities, 187
checkcontrolfileinputs
 utilities, 188
chunk_size
 csv_module::csv_file, 254
clat
 pahm_vortex, 131
clon
 pahm_vortex, 131
close
 csv_module::csv_file, 250
close_csv_file
 csv_module, 26
closelogfile
 pahm_messages, 84
closetol
 utilities, 212
cnttimebegin
 pahm_drivermod, 57
cnttimeend
 pahm_drivermod, 57
comparedoublereals
 pahm_sizes, 103
 pahm_sizes::comparereals, 243
comparesinglereals
 pahm_sizes, 104
 pahm_sizes::comparereals, 244
controlfilename
 pahm_global, 63
convlon
 utilities, 189
corio
 pahm_vortex, 131
count
 pahm_ncdfio::adcircodata_t, 213
 pahm_ncdfio::adcircvardata3d_t, 215
 pahm_ncdfio::adcircvardata_t, 218
cpptogeo_1d
 utilities, 189
 utilities::cpptogeo, 246
cpptogeo_scalar
 utilities, 191
 utilities::cpptogeo, 246
cprdt
 parwind::hollanddata_t, 268
cpress
 parwind::asymmetricvortexdata_t, 229
 parwind::hollanddata_t, 268
crdlats
 pahm_ncdfio, 99
crdlons
 pahm_ncdfio, 99
crdtime
 pahm_ncdfio, 99
crdxcs
 pahm_ncdfio, 99
crdycs
 pahm_ncdfio, 99
csv_data
 csv_module::csv_file, 254

csv_get_value
 csv_module, 26
 csv_module::csv_file, 250
csv_module, 22
 add_cell, 23
 add_matrix, 24
 add_vector, 25
 close_csv_file, 26
 csv_get_value, 26
 csv_type_double, 46
 csv_type_integer, 46
 csv_type_logical, 47
 csv_type_string, 47
 destroy_csv_file, 27
 get_character_column, 27
 get_column, 28
 get_csv_data_as_str, 29
 get_csv_string_column, 30
 get_header_csv_str, 31
 get_header_str, 32
 get_integer_column, 33
 get_logical_column, 34
 get_real_column, 35
 infer_variable_type, 36
 initialize_csv_file, 37
 next_row, 38
 number_of_lines_in_file, 38
 open_csv_file, 39
 read_csv_file, 40
 read_line_from_file, 40
 split, 41
 to_integer, 42
 to_logical, 43
 to_real, 44
 tokenize_csv_line, 45
 variable_types, 45
 zero, 47
csv_module::csv_file, 248
 add, 249
 add_cell, 250
 add_matrix, 250
 add_vector, 250
 chunk_size, 254
 close, 250
 csv_data, 254
 csv_get_value, 250
 delimiter, 254
 destroy, 250
 enclose_all_in_quotes, 254
 enclose_strings_in_quotes, 254
 get, 251
 get_character_column, 251
 get_column, 251
 get_csv_data_as_str, 251
 get_csv_string_column, 252
 get_header, 252
 get_header_csv_str, 252
 get_header_str, 252
 get_integer_column, 252
 get_logical_column, 252
 get_real_column, 253
 header, 254
 icol, 254
 initialize, 253
 iunit, 255
 logical_false_string, 255
 logical_true_string, 255
 n_cols, 255
 n_rows, 255
 next_row, 253
 open, 253
 quote, 255
 read, 253
 read_line_from_file, 253
 tokenize, 253
 variable_types, 253
 csv_module::csv_string, 256
 str, 256
csv_parameters, 47
 default_int_fmt, 47
 default_real_fmt, 48
 max_integer_str_len, 48
 max_real_str_len, 48
csv_type_double
 csv_module, 46
csv_type_integer
 csv_module, 46
csv_type_logical
 csv_module, 47
csv_type_string
 csv_module, 47
csv_utilities, 48
 expand_vector, 49
 max_size_for_insertion_sort, 52
 sortAscending, 49
 swap, 50
 unique, 51
csv_utilities.F90
 partition, 316
 quicksort, 317
cyclenum
 parwind::besttrackdata_t, 238
cynum
 parwind::besttrackdata_t, 238
datatmpres
 pahm_netcdfio, 100
datelements

pahm_netcdfio, 100
 datestimes
 pahm_global, 63
 datetime2string
 timedateutils, 165
 datwindx
 pahm_netcdfio, 100
 datwindy
 pahm_netcdfio, 100
 day
 parwind::asymmetricvortexdata_t, 229
 parwind::besttrackdata_t, 238
 parwind::hollanddata_t, 269
 dayofyear
 timedateutils, 166
 dayofyeartogreg
 timedateutils, 167
 deallocasymvortstruct
 parwind, 137
 deallocbtrstruct
 parwind, 138
 deallochollstruct
 parwind, 138
 debug
 pahm_messages, 90
 def_ncnam_pres
 pahm_global, 63
 def_ncnam_wndx
 pahm_global, 63
 def_ncnam_wndy
 pahm_global, 63
 default_int_fmt
 csv_parameters, 47
 default_real_fmt
 csv_parameters, 48
 defv_atmpress
 pahm_global, 64
 defv_gravity
 pahm_global, 64
 defv_rhoair
 pahm_global, 64
 defv_rhowater
 pahm_global, 64
 defv_windreduction
 pahm_global, 64
 deg2rad
 pahm_global, 64
 delimiter
 csv_module::csv_file, 254
 destroy
 csv_module::csv_file, 250
 destroy_csv_file
 csv_module, 27
 dimid
 dir
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 238
 dp
 pahm_mesh, 80
 drealscan
 utilities, 191
 dtg
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 238
 parwind::hollanddata_t, 269
 dvalstr
 utilities, 193
 echo
 pahm_messages, 90
 elapsedsecs
 timedateutils, 167
 elemdimid
 pahm_netcdfio, 100
 enclose_all_in_quotes
 csv_module::csv_file, 254
 enclose_strings_in_quotes
 csv_module::csv_file, 254
 enddate
 pahm_global, 65
 enddatespecified
 pahm_global, 65
 enddatetime
 pahm_global, 65
 endday
 pahm_global, 65
 endhour
 pahm_global, 65
 endmin
 pahm_global, 65
 endmonth
 pahm_global, 66
 endsec
 pahm_global, 66
 endsimspecified
 pahm_global, 66
 endsimtime
 pahm_global, 66
 endtime
 pahm_global, 66
 endyear
 pahm_global, 66
 error
 pahm_messages, 91
 ew
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 238

expand_vector
 csv_utilities, 49

eye
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 238

fang
 pahm_vortex, 112

fcstinc
 parwind::asymmetricvortexdata_t, 230
 parwind::hollanddata_t, 269

filefound
 pahm_netcdfio::filedata_t, 257

filename
 pahm_netcdfio::filedata_t, 257
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 239
 parwind::hollanddata_t, 269

filereccounter
 pahm_netcdfio::filedata_t, 257

findroot
 pahm_vortex, 113

firstgregdate
 timedateutils, 183

firstgregtime
 timedateutils, 183

firtrmaxes
 pahm_vortex, 113

firtrmaxes4
 pahm_vortex, 114

fixnearwholedoublereal
 pahm_sizes, 104
 pahm_sizes::fixnearwholereal, 258

fixnearwholesinglereal
 pahm_sizes, 105
 pahm_sizes::fixnearwholereal, 259

fnamelen
 pahm_sizes, 105

geofactor
 parwind, 148

geostrophicswitch
 parwind, 148

geotocpp_1d
 utilities, 193
 utilities::geotocpp, 260

geotocpp_scalar
 utilities, 194
 utilities::geotocpp, 261

get
 csv_module::csv_file, 251

get_character_column
 csv_module, 27
 csv_module::csv_file, 251

get_column
 csv_module, 28
 csv_module::csv_file, 251

get_csv_data_as_str
 csv_module, 29
 csv_module::csv_file, 251

get_csv_string_column
 csv_module, 30
 csv_module::csv_file, 252

get_header
 csv_module::csv_file, 252

get_header_csv_str
 csv_module, 31
 csv_module::csv_file, 252

get_header_str
 csv_module, 32
 csv_module::csv_file, 252

get_integer_column
 csv_module, 33
 csv_module::csv_file, 252

get_logical_column
 csv_module, 34
 csv_module::csv_file, 252

get_real_column
 csv_module, 35
 csv_module::csv_file, 253

getgahmfields
 parwind, 139

gethollandfields
 parwind, 140

getlatestangle
 pahm_vortex, 114

getlatestrmax
 pahm_vortex, 114

getline record
 utilities, 195

getlocandratio
 utilities, 195

getphifactors
 pahm_vortex, 114

getprogramcmdargs
 pahm_drivermod, 52

getrmaxes
 pahm_vortex, 115

getshapeparameter
 pahm_vortex, 115

getshapeparameters
 pahm_vortex, 115

gettmeconvsec
 timedateutils, 169

getusequadrantvr
 pahm_vortex, 116

getvmaxesbl
 pahm_vortex, 116

gravity

pahm_global, 67
 gregtojulday2
 timedateutils, 169
 timedateutils::gregtojulday, 262
 gregtojuldayisec
 timedateutils, 170
 timedateutils::gregtojulday, 263
 gregtojuldayrsec
 timedateutils, 171
 timedateutils::gregtojulday, 265
 gusts
 parwind::asymmetricvortexdata_t, 230
 parwind::besttrackdata_t, 239
 header
 csv_module::csv_file, 254
 hollb
 parwind::asymmetricvortexdata_t, 230
 hollbs
 parwind::asymmetricvortexdata_t, 231
 holstru
 parwind, 148
 hour
 parwind::asymmetricvortexdata_t, 231
 parwind::besttrackdata_t, 239
 parwind::hollanddata_t, 269
 hp
 pahm_sizes, 106
 icol
 csv_module::csv_file, 254
 icompxchg
 sortutils.F90, 406
 icpress
 parwind::asymmetricvortexdata_t, 231
 parwind::hollanddata_t, 269
 ics
 pahm_mesh, 80
 idir
 parwind::asymmetricvortexdata_t, 231
 ilat
 parwind::asymmetricvortexdata_t, 231
 parwind::hollanddata_t, 269
 ilon
 parwind::asymmetricvortexdata_t, 231
 parwind::hollanddata_t, 269
 imissv
 pahm_sizes, 106
 indexxdouble
 sortutils, 155
 sortutils::indexx, 273
 indexxit
 sortutils, 155
 sortutils::indexx, 273
 indexxit8
 sortutils, 156
 sortutils::indexx, 274
 indexxsingle
 sortutils, 156
 sortutils::indexx, 275
 indexxstring
 sortutils, 157
 sortutils::indexx, 276
 infer_variable_type
 csv_module, 36
 info
 pahm_messages, 91
 initadcircnetcdfoutfile
 pahm_ncdfio, 95
 initialize
 csv_module::csv_file, 253
 initialize_csv_file
 csv_module, 37
 initialized
 pahm_ncdfio::adcirccoorddata_t, 213
 pahm_ncdfio::adcircvardata3d_t, 216
 pahm_ncdfio::adcircvardata_t, 218
 pahm_ncdfio::filedata_t, 258
 pahm_ncdfio::timedata_t, 294
 initials
 parwind::asymmetricvortexdata_t, 231
 parwind::besttrackdata_t, 239
 initlogging
 pahm_messages, 84
 initval
 pahm_ncdfio::adcirccoorddata_t, 213
 pahm_ncdfio::adcircvardata3d_t, 216
 pahm_ncdfio::adcircvardata_t, 218
 int1
 pahm_sizes, 106
 int16
 pahm_sizes, 106
 int2
 pahm_sizes, 106
 int4
 pahm_sizes, 106
 int8
 pahm_sizes, 106
 interpr
 pahm_vortex, 116
 intlat
 parwind::besttrackdata_t, 239
 intlon
 parwind::besttrackdata_t, 239
 intmslp
 parwind::besttrackdata_t, 239
 intpouter
 parwind::besttrackdata_t, 239
 intrad1

parwind::besttrackdata_t, 240
intrad2
 parwind::besttrackdata_t, 240
intrad3
 parwind::besttrackdata_t, 240
intrad4
 parwind::besttrackdata_t, 240
intrmw
 parwind::besttrackdata_t, 240
introuter
 parwind::besttrackdata_t, 240
intscan
 utilities, 196
intspeed
 parwind::besttrackdata_t, 240
intvalstr
 utilities, 197
intvmax
 parwind::besttrackdata_t, 240
ip
 pahm_sizes, 107
iprp
 parwind::asymmetricvortexdata_t, 231
ir
 parwind::asymmetricvortexdata_t, 232
irmw
 parwind::asymmetricvortexdata_t, 232
 parwind::hollanddata_t, 270
irrp
 parwind::asymmetricvortexdata_t, 232
 parwind::hollanddata_t, 270
ismeshok
 pahm_mesh, 80
isotachspercycle
 parwind::asymmetricvortexdata_t, 232
ispeed
 parwind::asymmetricvortexdata_t, 232
 parwind::hollanddata_t, 270
istormspeed
 parwind::asymmetricvortexdata_t, 232
iunit
 csv_module::csv_file, 255
ivr
 parwind::asymmetricvortexdata_t, 232
joindate
 timedateutils, 172
juldaytogreg
 timedateutils, 173
kt2ms
 pahm_global, 67
lat
 parwind::asymmetricvortexdata_t, 232
parwind::besttrackdata_t, 241
parwind::hollanddata_t, 270
latestangle
 pahm_vortex, 131
latestrmax
 pahm_vortex, 131
leapyear
 timedateutils, 175
llong
 pahm_sizes, 107
loaded
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
 parwind::hollanddata_t, 270
loadintvar
 utilities, 198
loadlogvar
 utilities, 199
loadrealvar
 utilities, 199
logfileame
 pahm_global, 67
logfileopened
 pahm_messages, 91
logical_false_string
 csv_module::csv_file, 255
logical_true_string
 csv_module::csv_file, 255
logitcalled
 pahm_messages, 91
loglevelnames
 pahm_messages, 91
logmessage_1
 pahm_messages, 85
 pahm_messages::logmessage, 278
logmessage_2
 pahm_messages, 85
 pahm_messages::logmessage, 278
lon
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
 parwind::hollanddata_t, 270
long
 pahm_sizes, 107
lun_btrk
 pahm_global, 67
lun_btrk1
 pahm_global, 67
lun_ctrl
 pahm_global, 67
lun_inp
 pahm_global, 68
lun_inp1
 pahm_global, 68

lun_log
 pahm_global, 68
 lun_out
 pahm_global, 68
 lun_out1
 pahm_global, 68
 lun_screen
 pahm_global, 68

 m2nm
 pahm_global, 69
 max_integer_str_len
 csv_parameters, 48
 max_real_str_len
 csv_parameters, 48
 max_size_for_insertion_sort
 csv_utilities, 52
 maxfacenodes
 pahm_mesh, 80
 maxseas
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
 mb2kpa
 pahm_global, 69
 mb2pa
 pahm_global, 69
 mdbegsimtime
 pahm_global, 69
 mdendsimtime
 pahm_global, 69
 mdjdate
 timedateutils, 183
 mdjoffset
 timedateutils, 184
 mdjtime
 timedateutils, 184
 mdoutdt
 pahm_global, 69
 meshdimid
 pahm_netcdfio, 100
 meshfileform
 pahm_global, 70
 meshfilename
 pahm_global, 70
 meshfilenamespecified
 pahm_global, 70
 mesh filetype
 pahm_global, 70
 meshvarid
 pahm_netcdfio, 101
 messagesources
 pahm_messages, 92
 method
 parwind, 149

 modeldate
 timedateutils, 184
 modeltime
 timedateutils, 184
 modeltype
 pahm_global, 70
 modjuldate
 timedateutils, 184
 modjultime
 timedateutils, 184
 month
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
 parwind::hollanddata_t, 270
 monthdays
 timedateutils, 176
 ms2kt
 pahm_global, 70
 myfile
 pahm_netcdfio, 101
 mytime
 pahm_netcdfio, 101

 n_cols
 csv_module::csv_file, 255
 n_rows
 csv_module::csv_file, 255
 nbtrfiles
 pahm_global, 71
 nbytes
 pahm_sizes, 107
 nc3form
 pahm_netcdfio, 101
 nc4form
 pahm_netcdfio, 101
 ncdeflate
 pahm_global, 71
 ncdlevel
 pahm_global, 71
 ncformat
 pahm_netcdfio, 101
 ncshuffle
 pahm_global, 71
 ncvarnam_pres
 pahm_global, 71
 ncvarnam_wndx
 pahm_global, 71
 ncvarnam_wndy
 pahm_global, 72
 ncycles
 parwind::asymmetricvortexdata_t, 233
 ne
 pahm_mesh, 80
 NetCDFCheckErr

netcdfio.F90, 342
netcdfio.F90
 NetCDFCheckErr, 342
netcdfterminate
 pahm_netcdfio, 96
newadcircnetcdfoutfile
 pahm_netcdfio, 96
newvortex
 pahm_vortex, 117
newvortexfull
 pahm_vortex, 117
next_row
 csv_module, 38
 csv_module::csv_file, 253
nfn
 pahm_mesh, 80
nm
 pahm_mesh, 81
nm2m
 pahm_global, 72
nodedimid
 pahm_netcdfio, 102
noutdt
 pahm_global, 72
np
 pahm_mesh, 81
npoints
 pahm_vortex, 131
nquads
 pahm_vortex, 132
ns
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
nscreen
 pahm_messages, 92
number_of_lines_in_file
 csv_module, 38
numbtfiles
 utilities, 212
numcycle
 parwind::asymmetricvortexdata_t, 233
numrec
 parwind::asymmetricvortexdata_t, 233
 parwind::besttrackdata_t, 241
 parwind::hollanddata_t, 270

offfirstgregday
 timedateutils, 184
offmodeljulday
 timedateutils, 185
offmodjulday
 timedateutils, 185
offunixjulday
 timedateutils, 185

omega
 pahm_global, 72
one2ten
 pahm_global, 72
open
 csv_module::csv_file, 253
open_csv_file
 csv_module, 39
openfileforread
 utilities, 200
openlogfile
 pahm_messages, 85
outdt
 pahm_global, 72
outfilefilename
 pahm_global, 73
outfilenamespecified
 pahm_global, 73

pahm
 pahm.F90, 354
pahm.F90
 pahm, 354
pahm_drivermod, 52
 cnttimebegin, 57
 cnttimeend, 57
 getprogramcmdargs, 52
 pahm_finalize, 53
 pahm_init, 54
 pahm_run, 55
pahm_finalize
 pahm_drivermod, 53
pahm_global, 57
 airdensity, 60
 backgroundatmpress, 60
 basee, 60
 begdate, 60
 begdatespecified, 61
 begdatetime, 61
 begday, 61
 beghour, 61
 begmin, 61
 begmonth, 61
 begsec, 62
 begsimspecified, 62
 begsimtime, 62
 begtime, 62
 begyear, 62
 besttrackfilename, 62
 besttrackfilenamespecified, 63
 controlfilename, 63
 datestimes, 63
 def_ncnam_pres, 63
 def_ncnam_wndx, 63

def_ncnam_wndy, 63
 defv_atmpress, 64
 defv_gravity, 64
 defv_rhoair, 64
 defv_rhowater, 64
 defv_windreduction, 64
 deg2rad, 64
 enddate, 65
 enddatespecified, 65
 enddatetime, 65
 endday, 65
 endhour, 65
 endmin, 65
 endmonth, 66
 endsec, 66
 endsimspecified, 66
 endsimtime, 66
 endtime, 66
 endyear, 66
 gravity, 67
 kt2ms, 67
 logfilename, 67
 lun_btrk, 67
 lun_btrk1, 67
 lun_ctrl, 67
 lun_inp, 68
 lun_inp1, 68
 lun_log, 68
 lun_out, 68
 lun_out1, 68
 lun_screen, 68
 m2nm, 69
 mb2kpa, 69
 mb2pa, 69
 mdbegsimtime, 69
 mdendsimtime, 69
 mdoutdt, 69
 meshfileform, 70
 meshfilename, 70
 meshfilenamespecified, 70
 mesh filetype, 70
 modeltype, 70
 ms2kt, 70
 nbtrfiles, 71
 ncdeflate, 71
 ncdlevel, 71
 ncshuffle, 71
 ncvarnam_pres, 71
 ncvarnam_wndx, 71
 ncvarnam_wndy, 72
 nm2m, 72
 noutdt, 72
 omega, 72
 one2ten, 72
 outdt, 72
 outfilename, 73
 outfilenamespecified, 73
 pi, 73
 rad2deg, 73
 rearth, 73
 refdate, 73
 refdatespecified, 74
 refdatetime, 74
 refday, 74
 refhour, 74
 refmin, 74
 refmonth, 74
 refsec, 75
 reftime, 75
 refyear, 75
 rhoair, 75
 rhowater, 75
 ten2one, 75
 times, 76
 title, 76
 unittime, 76
 windreduction, 76
 wpress, 76
 writeparams, 76
 wvelx, 77
 wvely, 77
 pahm_init
 pahm_drivermod, 54
 pahm_mesh, 77
 agrid, 79
 allocatenodalandelementalarrays, 78
 dp, 80
 ics, 80
 ismeshok, 80
 maxfacenodes, 80
 ne, 80
 nfn, 80
 nm, 81
 np, 81
 readmesh, 78
 readmeshasciifort14, 79
 fea, 81
 fea0, 81
 slam, 81
 slam0, 81
 xclam, 82
 ycsfea, 82
 pahm_messages, 82
 allmessage_1, 83
 allmessage_2, 83
 closelogfile, 84
 debug, 90
 echo, 90

error, 91
info, 91
initlogging, 84
logfileopened, 91
loginitcalled, 91
loglevelname, 91
logmessage_1, 85
logmessage_2, 85
messagesources, 92
nscreen, 92
openlogfile, 85
programhelp, 85
programversion, 86
scratchformat, 92
scratchmessage, 92
screenmessage_1, 86
screenmessage_2, 87
setmessagesource, 87
sourcenumber, 92
terminate, 88
unsetmessagesource, 89
warning, 93
pahm_messages::allmessage, 219
 allmessage_1, 219
 allmessage_2, 220
pahm_messages::logmessage, 277
 logmessage_1, 278
 logmessage_2, 278
pahm_messages::screenmessage, 279
 screenmessage_1, 279
 screenmessage_2, 280
pahm_netcdlio, 93
 base_netcdfcheckerr, 94
 crdlats, 99
 crdlons, 99
 crdtime, 99
 crdxcs, 99
 crdycs, 99
 datatmpres, 100
 datelements, 100
 datwindx, 100
 datwindy, 100
 elemdimid, 100
 initadcircnetcdfoutfile, 95
 meshdimid, 100
 meshvarid, 101
 myfile, 101
 mytime, 101
 nc3form, 101
 nc4form, 101
 ncformat, 101
 netcdfterminate, 96
 newadcircnetcdfoutfile, 96
 nodedimid, 102
 projvarid, 102
 setrecordcounterandstoretime, 97
 vertdimid, 102
 writenetcdffrecord, 98
pahm_netcdlio::adcirccoorddata_t, 212
 count, 213
 dimid, 213
 initialized, 213
 initval, 213
 start, 213
 var, 214
 vardimids, 214
 vardims, 214
 varid, 214
 varname, 214
pahm_netcdlio::adcircvardata3d_t, 215
 count, 215
 initialized, 216
 initval, 216
 start, 216
 var, 216
 vardimids, 216
 vardims, 216
 varid, 216
 varname, 216
pahm_netcdlio::adcircvardata_t, 217
 count, 218
 initialized, 218
 initval, 218
 start, 218
 var, 218
 vardimids, 218
 vardims, 218
 varid, 218
 varname, 219
pahm_netcdlio::filedata_t, 257
 filefound, 257
 filename, 257
 filereccounter, 257
 initialized, 258
pahm_netcdlio::timedata_t, 293
 initialized, 294
 time, 294
 timedimid, 294
 timedims, 294
 timeid, 294
 timelen, 294
pahm_run
 pahm_drivermod, 55
pahm_sizes, 102
 blank, 105
 comparedoublereals, 103
 comparesinglereals, 104
 fixnearwholedoublereal, 104

fixnearwholesinglereal, 105
 fnamelen, 105
 hp, 106
 imissv, 106
 int1, 106
 int16, 106
 int2, 106
 int4, 106
 int8, 106
 ip, 107
 llong, 107
 long, 107
 nbyte, 107
 rmissv, 107
 sp, 107
 sz, 107
 wp, 108
 pahm_sizes::comparereals, 243
 comparedoublereals, 243
 comparesinglereals, 244
 pahm_sizes::fixnearwholereal, 258
 fixnearwholedoublereal, 258
 fixnearwholesinglereal, 259
 pahm_vortex, 108
 b, 130
 bs, 130
 bs4, 130
 calcintensitychange, 110
 calcrmaxes, 111
 calcrmaxesfull, 111
 clat, 131
 clon, 131
 corio, 131
 fang, 112
 findroot, 113
 firmaxes, 113
 firmaxes4, 114
 getlatestangle, 114
 getlatestrmax, 114
 getphifactors, 114
 getrmaxes, 115
 getshapeparameter, 115
 getshapeparameters, 115
 getusequadrantvr, 116
 getvmaxesbl, 116
 interpr, 116
 latestangle, 131
 latestrmax, 131
 newvortex, 117
 newvortexfull, 117
 npoints, 131
 nquads, 132
 pc, 132
 phi, 132
 phis, 132
 phis4, 132
 pn, 132
 quad, 133
 quadflag4, 133
 quadir4, 133
 radius, 133
 rmaxes, 133
 rmaxes4, 133
 rmw, 118
 rotate, 119
 setisotachradii, 120
 setisotachwindspeed, 120
 setisotachwindspeeds, 120
 setrmaxes, 120
 setshapeparameter, 121
 setusequadrantvr, 121
 setusevmaxesbl, 121
 setvmaxesbl, 121
 setvortex, 122
 spline, 122
 usequadrantvr, 134
 usevmaxesbl, 134
 uvp, 123
 uvpr, 124
 uvtrans, 126
 uvtranspoint, 127
 vhncori, 128
 vhwithcori, 128
 vhwithcorifull, 129
 vmax, 134
 vmbl, 134
 vmbl4, 134
 vr, 134
 vrquadrant, 135
 parseline
 utilities, 201
 partition
 csv_utilities.F90, 316
 parwind, 135
 allocasymvortstruct, 136
 allocbtrstruct, 136
 allochollstruct, 137
 approach, 148
 asyvortstru, 148
 besttrackdata, 148
 deallocasymvortstruct, 137
 deallocbtrstruct, 138
 deallocchollstruct, 138
 geofactor, 148
 geostrophicswitch, 148
 getgahmfields, 139
 gethollandfields, 140
 holstru, 148

method, 149
processasymmetricvortexdata, 141
processhollanddata, 143
readbesttrackfile, 144
readcsvbesttrackfile, 145
stormnamelen, 149
writeasymmetricvortexdata, 146
writebesttrackdata, 147
parwind::asymmetricvortexdata_t, 227
basin, 229
castime, 229
castype, 229
casttypenum, 229
cpres, 229
day, 229
dir, 230
dtg, 230
ew, 230
eye, 230
fcstinc, 230
filename, 230
gusts, 230
hollb, 230
hollbs, 231
hour, 231
icpress, 231
idir, 231
ilat, 231
ilon, 231
initials, 231
iprp, 231
ir, 232
irmw, 232
irrp, 232
isotachspercycle, 232
ispeed, 232
istormspeed, 232
ivr, 232
lat, 232
loaded, 233
lon, 233
maxseas, 233
month, 233
ncycles, 233
ns, 233
numcycle, 233
numrec, 233
prp, 234
quadflag, 234
rmaxw, 234
rmw, 234
rrp, 234
speed, 234
stormname, 234
stormnumber, 234
stormspeed, 235
subregion, 235
thisstorm, 235
trvx, 235
trvy, 235
ty, 235
vmaxesbl, 235
windcode, 235
year, 236
parwind::besttrackdata_t, 236
basin, 238
cyclenum, 238
cynum, 238
day, 238
dir, 238
dtg, 238
ew, 238
eye, 238
filename, 239
gusts, 239
hour, 239
initials, 239
intlat, 239
intlon, 239
intmslp, 239
intpouter, 239
intrad1, 240
intrad2, 240
intrad3, 240
intrad4, 240
intrmw, 240
introuter, 240
intspeed, 240
intvmax, 240
lat, 241
loaded, 241
lon, 241
maxseas, 241
month, 241
ns, 241
numrec, 241
rad, 241
stormname, 242
subregion, 242
tau, 242
tech, 242
technum, 242
thisstorm, 242
ty, 242
windcode, 242
year, 243
parwind::hollanddata_t, 267
basin, 268

casttime, 268
 casttype, 268
 cprdt, 268
 cpres, 268
 day, 269
 dtg, 269
 fcstinc, 269
 filename, 269
 hour, 269
 icpres, 269
 ilat, 269
 ilon, 269
 irmw, 270
 irrp, 270
 ispeed, 270
 lat, 270
 loaded, 270
 lon, 270
 month, 270
 numrec, 270
 rmw, 271
 rrp, 271
 speed, 271
 stormnumber, 271
 thisstorm, 271
 trvx, 271
 trvy, 271
 year, 271

pc
 pahm_vortex, 132

phi
 pahm_vortex, 132

phis
 pahm_vortex, 132

phis4
 pahm_vortex, 132

pi
 pahm_global, 73

pn
 pahm_vortex, 132

preprocessdatetimestring
 timedateutils, 177

printmodelparams
 utilities, 202

processasymmetricvortexdata
 parwind, 141

processhollanddata
 parwind, 143

programhelp
 pahm_messages, 85

programversion
 pahm_messages, 86

projvarid
 pahm_netcdfio, 102

prp
 parwind::asymmetricvortexdata_t, 234

quad
 pahm_vortex, 133

quadflag
 parwind::asymmetricvortexdata_t, 234

quadflag4
 pahm_vortex, 133

quadir4
 pahm_vortex, 133

quicksort
 csv_utilities.F90, 317
 sortutils, 157

quote
 csv_module::csv_file, 255

rad
 parwind::besttrackdata_t, 241

rad2deg
 pahm_global, 73

radius
 pahm_vortex, 133

read
 csv_module::csv_file, 253

read_csv_file
 csv_module, 40

read_line_from_file
 csv_module, 40
 csv_module::csv_file, 253

readbesttrackfile
 parwind, 144

readcontrolfile
 utilities, 203

readcsvbesttrackfile
 parwind, 145

readmesh
 pahm_mesh, 78

readmeshasciifort14
 pahm_mesh, 79

realscan
 utilities, 204

rearth
 pahm_global, 73

refdate
 pahm_global, 73

refdatespecified
 pahm_global, 74

refdatetime
 pahm_global, 74

refday
 pahm_global, 74

refhour
 pahm_global, 74

refmin

pahm_global, 74
refmonth
 pahm_global, 74
refsec
 pahm_global, 75
reftime
 pahm_global, 75
refyear
 pahm_global, 75
rhoair
 pahm_global, 75
rhowater
 pahm_global, 75
rmaxes
 pahm_vortex, 133
rmaxes4
 pahm_vortex, 133
rmaxw
 parwind::asymmetricvortexdata_t, 234
rmissv
 pahm_sizes, 107
rmw
 pahm_vortex, 118
 parwind::asymmetricvortexdata_t, 234
 parwind::hollanddata_t, 271
rotate
 pahm_vortex, 119
rrp
 parwind::asymmetricvortexdata_t, 234
 parwind::hollanddata_t, 271

scratchformat
 pahm_messages, 92
scratchmessage
 pahm_messages, 92
screenmessage_1
 pahm_messages, 86
 pahm_messages::screenmessage, 279
screenmessage_2
 pahm_messages, 87
 pahm_messages::screenmessage, 280
setisotachradii
 pahm_vortex, 120
setisotachwindspeed
 pahm_vortex, 120
setisotachwindspeeds
 pahm_vortex, 120
setmessagesource
 pahm_messages, 87
setrecordcounterandstoretime
 pahm_netcdfio, 97
setrmaxes
 pahm_vortex, 120
setshapeparameter
 pahm_vortex, 121
setusequadrantvr
 pahm_vortex, 121
setusevmaxesbl
 pahm_vortex, 121
setvmaxesbl
 pahm_vortex, 121
setvortex
 pahm_vortex, 122
sfea
 pahm_mesh, 81
sfea0
 pahm_mesh, 81
slam
 pahm_mesh, 81
slam0
 pahm_mesh, 81
sort2
 sortutils, 158
sortAscending
 csv_utilities, 49
sortutils, 149
 arraycopydouble, 150
 arraycopyint, 151
 arraycopysingle, 151
 arrayequaldouble, 152
 arrayequalint, 152
 arrayequalsingle, 153
 arthdouble, 153
 arthint, 154
 arthsingle, 154
 indexxdouble, 155
 indexxit, 155
 indexxit8, 156
 indexxsingle, 156
 indexxstring, 157
 quicksort, 157
 sort2, 158
 stringlexcomp, 159
 swapdouble, 160
 swapdoublevec, 161
 swapint, 161
 swapintvec, 162
 swapsingle, 162
 swapsinglevec, 163
sortutils.F90
 icompxchg, 406
sortutils::arraycopy, 220
 arraycopydouble, 221
 arraycopyint, 221
 arraycopysingle, 222
sortutils::arrayequal, 222
 arrayequaldouble, 223
 arrayequalint, 223

arrayequalsingle, 224
 sortutils::arth, 225
 arthdouble, 225
 arthint, 226
 arthsingle, 226
 sortutils::indexx, 272
 indexxdouble, 273
 indexxit, 273
 indexxit8, 274
 indexxsingle, 275
 indexxstring, 276
 sortutils::swap, 286
 swapdouble, 287
 swapdoublevec, 287
 swapint, 288
 swapintvec, 289
 swapsingle, 289
 swapsinglevec, 290
 sourcenumber
 pahm_messages, 92
 sp
 pahm_sizes, 107
 speed
 parwind::asymmetricvortexdata_t, 234
 parwind::hollanddata_t, 271
 sphericaldistance_1d
 utilities, 205
 utilities::sphericaldistance, 281
 sphericaldistance_2d
 utilities, 206
 utilities::sphericaldistance, 282
 sphericaldistance_scalar
 utilities, 207
 utilities::sphericaldistance, 282
 sphericaldistanceharv
 utilities, 208
 sphericalfracpoint
 utilities, 208
 sinterp
 pahm_vortex, 122
 split
 csv_module, 41
 splitdate
 timedateutils, 178
 splitdatetimestring
 timedateutils, 179
 timedateutils::splitdatetimestring, 284
 splitdatetimestring2
 timedateutils, 179
 timedateutils::splitdatetimestring, 285
 start
 pahm_netcdfio::adcirccoorddata_t, 213
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 218
 stormname
 parwind::asymmetricvortexdata_t, 234
 parwind::besttrackdata_t, 242
 stormnamelen
 parwind, 149
 stormnumber
 parwind::asymmetricvortexdata_t, 234
 parwind::hollanddata_t, 271
 stormspeed
 parwind::asymmetricvortexdata_t, 235
 str
 csv_module::csv_string, 256
 stringlexcomp
 sortutils, 159
 subregion
 parwind::asymmetricvortexdata_t, 235
 parwind::besttrackdata_t, 242
 swap
 csv_utilities, 50
 swapdouble
 sortutils, 160
 sortutils::swap, 287
 swapdoublevec
 sortutils, 161
 sortutils::swap, 287
 swapint
 sortutils, 161
 sortutils::swap, 288
 swapintvec
 sortutils, 162
 sortutils::swap, 289
 swapsingle
 sortutils, 162
 sortutils::swap, 289
 swapsinglevec
 sortutils, 163
 sortutils::swap, 290
 sz
 pahm_sizes, 107
 tau
 parwind::besttrackdata_t, 242
 tech
 parwind::besttrackdata_t, 242
 technum
 parwind::besttrackdata_t, 242
 tenZone
 pahm_global, 75
 terminate
 pahm_messages, 88
 thisstorm
 parwind::asymmetricvortexdata_t, 235
 parwind::besttrackdata_t, 242
 parwind::hollanddata_t, 271

time
 pahm_ncdfio::timedata_t, 294

timeconviseC
 timedateutils, 181
 timedateutils::timeconv, 291

timeconvrsec
 timedateutils, 181
 timedateutils::timeconv, 292

timedateutils, 163
 datetime2string, 165
 dayofyear, 166
 dayofyeartogreg, 167
 elapsedsecs, 167
 firstgregdate, 183
 firstgregtime, 183
 gettimeconvsec, 169
 gregtojulday2, 169
 gregtojuldayisec, 170
 gregtojuldaysec, 171
 joindate, 172
 juldaytogreg, 173
 leapyear, 175
 mdjdate, 183
 mdjoffset, 184
 mdjtime, 184
 modeldate, 184
 modeltime, 184
 modjuldate, 184
 modjultime, 184
 monthdays, 176
 offfirstgregday, 184
 offmodeljulday, 185
 offmodjulday, 185
 offunixjulday, 185
 preprocessdatetimestring, 177
 splitdate, 178
 splitdatetimestring, 179
 splitdatetimestring2, 179
 timeconviseC, 181
 timeconvrsec, 181
 unixdate, 185
 unixtime, 185
 upp, 182
 usemodjulday, 185
 yeardays, 182

timedateutils::gregtojulday, 262
 gregtojulday2, 262
 gregtojuldayisec, 263
 gregtojuldaysec, 265

timedateutils::splitdatetimestring, 284
 splitdatetimestring, 284
 splitdatetimestring2, 285

timedateutils::timeconv, 291
 timeconviseC, 291

timeconvrsec, 292

timedimid
 pahm_ncdfio::timedata_t, 294

timedims
 pahm_ncdfio::timedata_t, 294

timeid
 pahm_ncdfio::timedata_t, 294

timelen
 pahm_ncdfio::timedata_t, 294

times
 pahm_global, 76

title
 pahm_global, 76

to_integer
 csv_module, 42

to_logical
 csv_module, 43

to_real
 csv_module, 44

tokenize
 csv_module::csv_file, 253

tokenize_csv_line
 csv_module, 45

tolowercase
 utilities, 209

touppercase
 utilities, 210

trvx
 parwind::asymmetricvortexdata_t, 235
 parwind::hollanddata_t, 271

trvy
 parwind::asymmetricvortexdata_t, 235
 parwind::hollanddata_t, 271

ty
 parwind::asymmetricvortexdata_t, 235
 parwind::besttrackdata_t, 242

unique
 csv_utilities, 51

unittime
 pahm_global, 76

unixdate
 timedateutils, 185

unixtime
 timedateutils, 185

unsetmessagesource
 pahm_messages, 89

upp
 timedateutils, 182

usemodjulday
 timedateutils, 185

usequadrantvr
 pahm_vortex, 134

user-guide/abstract.md, 295

user-guide/application.md, 295
 user-guide/code.md, 295
 user-guide/credits.md, 295
 user-guide/deliverables.md, 295
 user-guide/evaluation.md, 295
 user-guide/features.md, 295
 user-guide/figures.md, 295
 user-guide/glossary.md, 295
 user-guide/intro.md, 295
 user-guide/models.md, 295
 user-guide/pahm_manual.md, 295
 user-guide/references-dox.md, 295
 user-guide/tables.md, 295
 usevmaxesbl
 pahm_vortex, 134
 utilities, 186
 charunique, 187
 checkcontrolfileinputs, 188
 closetol, 212
 convlon, 189
 cpptogeo_1d, 189
 cpptogeo_scalar, 191
 drealscan, 191
 dvalstr, 193
 geotocpp_1d, 193
 geotocpp_scalar, 194
 getlinerecord, 195
 getlocandratio, 195
 intscan, 196
 intvalstr, 197
 loadintvar, 198
 loadlogvar, 199
 loadrealvar, 199
 numbtfiles, 212
 openfileforread, 200
 parseline, 201
 printmodelparams, 202
 readcontrolfile, 203
 realscan, 204
 sphericaldistance_1d, 205
 sphericaldistance_2d, 206
 sphericaldistance_scalar, 207
 sphericaldistanceharv, 208
 sphericalfracpoint, 208
 tolowercase, 209
 touppercase, 210
 valstr, 211
 utilities::cpptogeo, 245
 cpptogeo_1d, 246
 cpptogeo_scalar, 246
 utilities::geotocpp, 260
 geotocpp_1d, 260
 geotocpp_scalar, 261
 utilities::sphericaldistance, 280
 sphericaldistance_1d, 281
 sphericaldistance_2d, 282
 sphericaldistance_scalar, 282
 uvp
 pahm_vortex, 123
 uvpr
 pahm_vortex, 124
 uvtrans
 pahm_vortex, 126
 uvtranspoint
 pahm_vortex, 127
 valstr
 utilities, 211
 var
 pahm_netcdfio::adcirccoorddata_t, 214
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 218
 vardimids
 pahm_netcdfio::adcirccoorddata_t, 214
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 218
 vardims
 pahm_netcdfio::adcirccoorddata_t, 214
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 218
 variable_types
 csv_module, 45
 csv_module::csv_file, 253
 varid
 pahm_netcdfio::adcirccoorddata_t, 214
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 218
 varname
 pahm_netcdfio::adcirccoorddata_t, 214
 pahm_netcdfio::adcircvardata3d_t, 216
 pahm_netcdfio::adcircvardata_t, 219
 vertdimid
 pahm_netcdfio, 102
 vhnocori
 pahm_vortex, 128
 vhwithcori
 pahm_vortex, 128
 vhwithcorifull
 pahm_vortex, 129
 vmax
 pahm_vortex, 134
 vmaxesbl
 parwind::asymmetricvortexdata_t, 235
 vmb1
 pahm_vortex, 134
 vmb14
 pahm_vortex, 134
 vr

pahm_vortex, 134
vrquadrant
 pahm_vortex, 135

warning
 pahm_messages, 93

windcode
 parwind::asymmetricvortexdata_t, 235
 parwind::besttrackdata_t, 242

windreduction
 pahm_global, 76

wp
 pahm_sizes, 108

wpress
 pahm_global, 76

writesymmetricvortexdata
 parwind, 146

writebesttrackdata
 parwind, 147

writenetcdfrecord
 pahm_netcdffio, 98

writeparams
 pahm_global, 76

wvelx
 pahm_global, 77

wvely
 pahm_global, 77

xcslam
 pahm_mesh, 82

ycsfea
 pahm_mesh, 82

year
 parwind::asymmetricvortexdata_t, 236
 parwind::besttrackdata_t, 243
 parwind::hollanddata_t, 271

yeardays
 timedateutils, 182

zero
 csv_module, 47