

PaHM

Generated by Doxygen 1.9.3

1 PaHM Manual	1
2 Abstract	2
3 List of Tables	3
4 List of Figures	4
5 Introduction	5
5.1 The Parametric Hurricane Modeling System (<i>PaHM</i>)	5
5.2 Downloading <i>PaHM</i>	5
5.2.1 Directory Structure	6
5.3 Building <i>PaHM</i>	8
5.3.1 System Requirements	8
5.3.2 The Build System	8
5.3.3 CMake Configuration Files and Modules	9
5.3.4 Installation	9
5.4 Using <i>PaHM</i>	9
5.4.1 Standalone Configuration	12
5.4.2 Coupling Configuration	13
6 Parametric Models in <i>PaHM</i>	14
6.1 Rankine Vortex Model	14
6.2 Holland Symmetric Vortex Models	15
6.3 Willoughby Symmetric Vortex Model	15
6.4 Generalized Asymmetric Vortex Holland model (<i>GAHM</i>)	15
7 <i>PaHM</i> Features and Capabilities	16
7.1 Data Input Interfaces	16
7.2 Model Grids	16
7.3 Modeling Multiple Interacting Storms	16
7.4 Coupling Environment	16
8 Model Application and Implementation Technology	17
8.1 Standalone Model Application	17
8.2 Coupled Model Application	17
9 Model Evaluation - Hurricane Florence (2018) Study	18
9.1 Statistical Performance Measures	19
9.2 Standalone Model Evaluation	21
9.2.1 Model Results and Discussion	21

9.3 Coupled Model Evaluation	22
9.3.1 Coupled Model Results and Discussion	24
9.4 Conclusions	25
10 List of Deliverables	26
11 Glossary	27
12 Credits	28
13 References	29
14 PaHM Code	30
14.1 Third-Party Libraries	30
14.2 Functional Parts	30
14.3 Modules	30
14.4 Data Types	30
14.4.1 Data Types List	30
14.4.2 Data Fields	30
14.5 Files	30
14.5.1 File List	30
14.5.2 File Members	30
15 Module Documentation	31
15.1 csv_module Module Reference	31
15.1.1 Function/Subroutine Documentation	33
15.1.1.1 add_cell()	33
15.1.1.2 add_matrix()	34
15.1.1.3 add_vector()	35
15.1.1.4 close_csv_file()	35
15.1.1.5 csv_get_value()	36
15.1.1.6 destroy_csv_file()	37
15.1.1.7 get_character_column()	37
15.1.1.8 get_column()	38
15.1.1.9 get_csv_data_as_str()	39
15.1.1.10 get_csv_string_column()	40
15.1.1.11 get_header_csv_str()	41
15.1.1.12 get_header_str()	42
15.1.1.13 get_integer_column()	43
15.1.1.14 get_logical_column()	44

15.1.1.15 get_real_column()	45
15.1.1.16 infer_variable_type()	46
15.1.1.17 initialize_csv_file()	47
15.1.1.18 next_row()	48
15.1.1.19 number_of_lines_in_file()	48
15.1.1.20 open_csv_file()	49
15.1.1.21 read_csv_file()	50
15.1.1.22 read_line_from_file()	51
15.1.1.23 split()	52
15.1.1.24 to_integer()	53
15.1.1.25 to_logical()	54
15.1.1.26 to_real()	55
15.1.1.27 tokenize_csv_line()	55
15.1.1.28 variable_types()	56
15.1.2 Variable Documentation	57
15.1.2.1 csv_type_double	57
15.1.2.2 csv_type_integer	57
15.1.2.3 csv_type_logical	58
15.1.2.4 csv_type_string	58
15.1.2.5 zero	58
15.2 csv_parameters Module Reference	58
15.2.1 Variable Documentation	58
15.2.1.1 default_int_fmt	59
15.2.1.2 default_real_fmt	59
15.2.1.3 max_integer_str_len	59
15.2.1.4 max_real_str_len	59
15.3 csv_utilities Module Reference	59
15.3.1 Function/Subroutine Documentation	60
15.3.1.1 expand_vector()	60
15.3.1.2 sortAscending()	61
15.3.1.3 swap()	61
15.3.1.4 unique()	62
15.3.2 Variable Documentation	63
15.3.2.1 max_size_for_insertion_sort	63
15.4 pahm_drivermod Module Reference	63
15.4.1 Function/Subroutine Documentation	64
15.4.1.1 getprogramcmdlargs()	64
15.4.1.2 pahm_finalize()	65

15.4.1.3 pahm_init()	65
15.4.1.4 pahm_run()	67
15.4.2 Variable Documentation	68
15.4.2.1 cnttimebegin	68
15.4.2.2 cnttimeend	68
15.5 pahm_global Module Reference	68
15.5.1 Function/Subroutine Documentation	71
15.5.1.1 airdensity()	71
15.5.2 Variable Documentation	71
15.5.2.1 backgroundatmpress	71
15.5.2.2 basee	72
15.5.2.3 begdate	72
15.5.2.4 begdatespecified	72
15.5.2.5 begdatetime	72
15.5.2.6 begday	72
15.5.2.7 beghour	73
15.5.2.8 begmin	73
15.5.2.9 begmonth	73
15.5.2.10 begsec	73
15.5.2.11 begsimspecified	73
15.5.2.12 begsimtime	74
15.5.2.13 begtime	74
15.5.2.14 begyear	74
15.5.2.15 besttrackfilename	74
15.5.2.16 besttrackfilenamespecified	74
15.5.2.17 controlfilename	75
15.5.2.18 datestimes	75
15.5.2.19 def_ncnam_pres	75
15.5.2.20 def_ncnam_wndx	75
15.5.2.21 def_ncnam_wndy	75
15.5.2.22 defv_atmpress	76
15.5.2.23 defv_gravity	76
15.5.2.24 defv_rhoair	76
15.5.2.25 defv_rhowater	76
15.5.2.26 defv_windreduction	76
15.5.2.27 deg2rad	77
15.5.2.28 enddate	77
15.5.2.29 enddatespecified	77

15.5.2.30 enddatetime	77
15.5.2.31 endday	78
15.5.2.32 endhour	78
15.5.2.33 endmin	78
15.5.2.34 endmonth	78
15.5.2.35 endsec	78
15.5.2.36 endsimsspecified	79
15.5.2.37 endsimtime	79
15.5.2.38 endtime	79
15.5.2.39 endyear	79
15.5.2.40 gravity	79
15.5.2.41 kt2ms	80
15.5.2.42 logfilename	80
15.5.2.43 lun_btrk	80
15.5.2.44 lun_btrk1	80
15.5.2.45 lun_ctrl	81
15.5.2.46 lun_inp	81
15.5.2.47 lun_inp1	81
15.5.2.48 lun_log	81
15.5.2.49 lun_out	81
15.5.2.50 lun_out1	82
15.5.2.51 lun_screen	82
15.5.2.52 m2nm	82
15.5.2.53 mb2kpa	82
15.5.2.54 mb2pa	82
15.5.2.55 mdbegsimtime	83
15.5.2.56 mdendsimtime	83
15.5.2.57 mdoutdt	83
15.5.2.58 meshfileform	83
15.5.2.59 meshfilename	84
15.5.2.60 meshfilenamespecified	84
15.5.2.61 meshfiletype	84
15.5.2.62 modeltype	84
15.5.2.63 ms2kt	84
15.5.2.64 nbtrfiles	85
15.5.2.65 ncdeflate	85
15.5.2.66 ncdlevel	85
15.5.2.67 ncshuffle	85

15.5.2.68 ncvarnam_pres	86
15.5.2.69 ncvarnam_wndx	86
15.5.2.70 ncvarnam_wndy	86
15.5.2.71 nm2m	86
15.5.2.72 noutdt	87
15.5.2.73 omega	87
15.5.2.74 one2ten	87
15.5.2.75 outdt	87
15.5.2.76 outfilename	88
15.5.2.77 outfilenamespecified	88
15.5.2.78 pi	88
15.5.2.79 rad2deg	88
15.5.2.80 rearth	88
15.5.2.81 refdate	89
15.5.2.82 refdatespecified	89
15.5.2.83 refdatetime	89
15.5.2.84 refday	89
15.5.2.85 refhour	89
15.5.2.86 refmin	90
15.5.2.87 refmonth	90
15.5.2.88 refsec	90
15.5.2.89 reftime	90
15.5.2.90 refyear	91
15.5.2.91 rhoair	91
15.5.2.92 rhowater	91
15.5.2.93 ten2one	91
15.5.2.94 times	91
15.5.2.95 title	92
15.5.2.96 unittime	92
15.5.2.97 windreduction	92
15.5.2.98 wpress	92
15.5.2.99 writeparams	93
15.5.2.100 wvelx	93
15.5.2.101 wvely	93
15.6 pahm_mesh Module Reference	93
15.6.1 Function/Subroutine Documentation	94
15.6.1.1 allocatenodalandelementalarrays()	94
15.6.1.2 readmesh()	95

15.6.1.3 readmeshasciifort14()	96
15.6.2 Variable Documentation	96
15.6.2.1 agrid	96
15.6.2.2 dp	97
15.6.2.3 ics	97
15.6.2.4 ismeshok	97
15.6.2.5 maxfacenodes	97
15.6.2.6 ne	97
15.6.2.7 nfn	98
15.6.2.8 nm	98
15.6.2.9 np	98
15.6.2.10 sfea	98
15.6.2.11 sfea0	99
15.6.2.12 slam	99
15.6.2.13 slam0	99
15.6.2.14 xcslam	99
15.6.2.15 ycsfea	99
15.7 pahm_messages Module Reference	100
15.7.1 Function/Subroutine Documentation	101
15.7.1.1 allmessage_1()	101
15.7.1.2 allmessage_2()	101
15.7.1.3 closelogfile()	101
15.7.1.4 initlogging()	102
15.7.1.5 logmessage_1()	102
15.7.1.6 logmessage_2()	103
15.7.1.7 openlogfile()	103
15.7.1.8 programhelp()	104
15.7.1.9 programversion()	104
15.7.1.10 screenmessage_1()	105
15.7.1.11 screenmessage_2()	105
15.7.1.12 setmessagesource()	105
15.7.1.13 terminate()	107
15.7.1.14 unsetmessagesource()	108
15.7.2 Variable Documentation	109
15.7.2.1 debug	110
15.7.2.2 echo	110
15.7.2.3 error	110
15.7.2.4 info	110

15.7.2.5 logfileopened	111
15.7.2.6 loginitcalled	111
15.7.2.7 loglevelname	111
15.7.2.8 messagesources	111
15.7.2.9 nscreen	112
15.7.2.10 scratchformat	112
15.7.2.11 scratchmessage	112
15.7.2.12 sourcenumber	112
15.7.2.13 warning	113
15.8 pahm_ncdfio Module Reference	113
15.8.1 Function/Subroutine Documentation	114
15.8.1.1 base_ncdfcheckerr()	114
15.8.1.2 initadcircnetcdfoutfile()	115
15.8.1.3 netcdfterminate()	116
15.8.1.4 newadcircnetcdfoutfile()	116
15.8.1.5 setrecordcounterandstoretime()	117
15.8.1.6 writenetcdfrecord()	118
15.8.2 Variable Documentation	119
15.8.2.1 crdlats	119
15.8.2.2 crdlons	119
15.8.2.3 crdftime	119
15.8.2.4 crdxcs	120
15.8.2.5 crdycs	120
15.8.2.6 datatmpres	120
15.8.2.7 datelements	120
15.8.2.8 datwindx	120
15.8.2.9 datwindy	121
15.8.2.10 elemdimid	121
15.8.2.11 meshdimid	121
15.8.2.12 meshvarid	121
15.8.2.13 myfile	121
15.8.2.14 mytime	122
15.8.2.15 nc3form	122
15.8.2.16 nc4form	122
15.8.2.17 ncformat	122
15.8.2.18 nodedimid	122
15.8.2.19 projvarid	123
15.8.2.20 vertdimid	123

15.9 pahm_sizes Module Reference	123
15.9.1 Function/Subroutine Documentation	124
15.9.1.1 comparedoublereals()	124
15.9.1.2 comparesinglereals()	125
15.9.1.3 fixnearwholedoublereal()	125
15.9.1.4 fixnearwholesinglereal()	126
15.9.2 Variable Documentation	126
15.9.2.1 blank	127
15.9.2.2 fnamelen	127
15.9.2.3 hp	127
15.9.2.4 imissv	127
15.9.2.5 int1	127
15.9.2.6 int16	128
15.9.2.7 int2	128
15.9.2.8 int4	128
15.9.2.9 int8	128
15.9.2.10 ip	128
15.9.2.11 llong	128
15.9.2.12 long	129
15.9.2.13 nbyte	129
15.9.2.14 rmissv	129
15.9.2.15 sp	129
15.9.2.16 sz	129
15.9.2.17 wp	130
15.10 pahm_vortex Module Reference	130
15.10.1 Function/Subroutine Documentation	132
15.10.1.1 calcintensitychange()	132
15.10.1.2 calcrmaxes()	133
15.10.1.3 calcrmaxesfull()	134
15.10.1.4 fang()	134
15.10.1.5 findroot()	135
15.10.1.6 fitrmaxes()	136
15.10.1.7 fitrmaxes4()	136
15.10.1.8 getlatestangle()	137
15.10.1.9 getlatestrmax()	137
15.10.1.10 getphifactors()	137
15.10.1.11 getrmaxes()	138
15.10.1.12 getshapeparameter()	138

15.10.1.13 getshapeparameters()	138
15.10.1.14 getusequadrantvr()	139
15.10.1.15 getvmaxesbl()	139
15.10.1.16 interp()	140
15.10.1.17 newvortex()	140
15.10.1.18 newvortexfull()	141
15.10.1.19 rmw()	141
15.10.1.20 rotate()	142
15.10.1.21 setisotachradii()	143
15.10.1.22 setisotachwindspeed()	143
15.10.1.23 setisotachwindspeeds()	144
15.10.1.24 setrmaxes()	144
15.10.1.25 setshapeparameter()	144
15.10.1.26 setusequadrantvr()	145
15.10.1.27 setusevmaxesbl()	145
15.10.1.28 setvmaxesbl()	146
15.10.1.29 setvortex()	146
15.10.1.30 spinterp()	147
15.10.1.31 uv()	148
15.10.1.32 uvpr()	149
15.10.1.33 uvtrans()	151
15.10.1.34 uvtranspoint()	152
15.10.1.35 vhnocori()	153
15.10.1.36 vhwithcori()	153
15.10.1.37 vhwithcorifull()	154
15.10.2 Variable Documentation	155
15.10.2.1 b	155
15.10.2.2 bs	155
15.10.2.3 bs4	156
15.10.2.4 clat	156
15.10.2.5 clon	156
15.10.2.6 corio	156
15.10.2.7 latestangle	156
15.10.2.8 latestrmax	157
15.10.2.9 npoints	157
15.10.2.10 nquads	157
15.10.2.11 pc	157
15.10.2.12 phi	157

15.10.2.13 phis	158
15.10.2.14 phis4	158
15.10.2.15 pn	158
15.10.2.16 quad	158
15.10.2.17 quadflag4	158
15.10.2.18 quadir4	159
15.10.2.19 radius	159
15.10.2.20 rmaxes	159
15.10.2.21 rmaxes4	159
15.10.2.22 usequadrantvr	159
15.10.2.23 usevmaxesbl	160
15.10.2.24 vmax	160
15.10.2.25 vmb1	160
15.10.2.26 vmb14	160
15.10.2.27 vr	161
15.10.2.28 vrquadrant	161
15.11 parwind Module Reference	161
15.11.1 Function/Subroutine Documentation	162
15.11.1.1 allocasymvortstruct()	162
15.11.1.2 allocbtrstruct()	163
15.11.1.3 allochollstruct()	163
15.11.1.4 deallocasymvortstruct()	164
15.11.1.5 deallocbtrstruct()	165
15.11.1.6 deallochollstruct()	166
15.11.1.7 getgahmfields()	166
15.11.1.8 gethollandfields()	168
15.11.1.9 processasymmetricvortexdata()	169
15.11.1.10 processhollanddata()	171
15.11.1.11 readbesttrackfile()	172
15.11.1.12 readcsvbesttrackfile()	173
15.11.1.13 writeasymmetricvortexdata()	174
15.11.1.14 writebesttrackdata()	175
15.11.2 Variable Documentation	175
15.11.2.1 approach	175
15.11.2.2 asyvortstru	176
15.11.2.3 besttrackdata	176
15.11.2.4 geofactor	176
15.11.2.5 geostrophicswitch	176

15.11.2.6 holstru	176
15.11.2.7 method	177
15.11.2.8 stormnamelen	177
15.12 sortutils Module Reference	177
15.12.1 Function/Subroutine Documentation	178
15.12.1.1 arraycopydouble()	178
15.12.1.2 arraycopyint()	179
15.12.1.3 arraycopsingle()	179
15.12.1.4 arrayequaldouble()	180
15.12.1.5 arrayequalint()	181
15.12.1.6 arrayequalsingle()	181
15.12.1.7 arthdouble()	182
15.12.1.8 arthint()	182
15.12.1.9 arthsingle()	183
15.12.1.10 indexxdouble()	183
15.12.1.11 indexxit()	184
15.12.1.12 indexxit8()	184
15.12.1.13 indexxsingle()	185
15.12.1.14 indexxstring()	186
15.12.1.15 quicksort()	186
15.12.1.16 sort2()	187
15.12.1.17 stringlexcomp()	188
15.12.1.18 swapdouble()	189
15.12.1.19 swapdoublevec()	190
15.12.1.20 swapint()	190
15.12.1.21 swapintvec()	191
15.12.1.22 swapsingle()	192
15.12.1.23 swapsinglevec()	192
15.13 timedateutils Module Reference	193
15.13.1 Function/Subroutine Documentation	194
15.13.1.1 datetime2string()	195
15.13.1.2 dayofyear()	196
15.13.1.3 dayofyeartogreg()	197
15.13.1.4 elapsedsecs()	197
15.13.1.5 gettimeconvsec()	199
15.13.1.6 gregtojulday2()	200
15.13.1.7 gregtojuldayisec()	201
15.13.1.8 gregtojuldayrsec()	202

15.13.1.9 joindate()	203
15.13.1.10 juldaytogreg()	203
15.13.1.11 leapyear()	205
15.13.1.12 monthdays()	206
15.13.1.13 preprocessdatetimestring()	207
15.13.1.14 splitdate()	207
15.13.1.15 splitdatetimestring()	208
15.13.1.16 splitdatetimestring2()	209
15.13.1.17 timeconvise()	209
15.13.1.18 timeconvrsec()	210
15.13.1.19 upp()	211
15.13.1.20 yeardays()	211
15.13.2 Variable Documentation	212
15.13.2.1 firstgregdate	212
15.13.2.2 firstgregtime	212
15.13.2.3 mdjdate	213
15.13.2.4 mdjoffset	213
15.13.2.5 mdjtime	213
15.13.2.6 modeldate	213
15.13.2.7 modeltime	213
15.13.2.8 modjuldate	214
15.13.2.9 modjultime	214
15.13.2.10 offfirstgregday	214
15.13.2.11 offmodeljulday	214
15.13.2.12 offmodjulday	214
15.13.2.13 offunixjulday	215
15.13.2.14 unixdate	215
15.13.2.15 unixtime	215
15.13.2.16 usemodjulday	215
15.14 utilities Module Reference	215
15.14.1 Function/Subroutine Documentation	217
15.14.1.1 charunique()	217
15.14.1.2 checkcontrolfileinputs()	218
15.14.1.3 convlon()	219
15.14.1.4 cpptogeot_1d()	219
15.14.1.5 cpptogeoscalar()	220
15.14.1.6 drealscan()	221
15.14.1.7 dvalstr()	222

15.14.1.8 geotocpp_1d()	223
15.14.1.9 geotocpp_scalar()	224
15.14.1.10 getlinerecord()	224
15.14.1.11 getlocandratio()	225
15.14.1.12 intscan()	226
15.14.1.13 intvalstr()	227
15.14.1.14 loadintvar()	228
15.14.1.15 loadlogvar()	229
15.14.1.16 loadrealvar()	230
15.14.1.17 openfileforread()	230
15.14.1.18 parseline()	231
15.14.1.19 printmodelparams()	233
15.14.1.20 readcontrolfile()	234
15.14.1.21 realscan()	236
15.14.1.22 sphericaldistance_1d()	237
15.14.1.23 sphericaldistance_2d()	238
15.14.1.24 sphericaldistance_scalar()	239
15.14.1.25 sphericaldistanceharv()	240
15.14.1.26 sphericalfracpoint()	240
15.14.1.27 tolowercase()	242
15.14.1.28 touppercase()	242
15.14.1.29 valstr()	243
15.14.2 Variable Documentation	244
15.14.2.1 closetol	244
15.14.2.2 numbitfiles	244
16 Data Type Documentation	245
16.1 pahm_netcdfio::adcirccoorddata_t Type Reference	245
16.1.1 Detailed Description	246
16.1.2 Member Data Documentation	246
16.1.2.1 count	246
16.1.2.2 dimid	246
16.1.2.3 initialized	246
16.1.2.4 initval	247
16.1.2.5 start	247
16.1.2.6 var	247
16.1.2.7 vardimids	247
16.1.2.8 vardims	247

16.1.2.9 varid	247
16.1.2.10 varname	248
16.2 pahm_netcdio::adcircvardata3d_t Type Reference	248
16.2.1 Detailed Description	249
16.2.2 Member Data Documentation	249
16.2.2.1 count	249
16.2.2.2 initialized	249
16.2.2.3 initval	249
16.2.2.4 start	249
16.2.2.5 var	250
16.2.2.6 vardimids	250
16.2.2.7 vardims	250
16.2.2.8 varid	250
16.2.2.9 varname	250
16.3 pahm_netcdio::adcircvardata_t Type Reference	251
16.3.1 Detailed Description	251
16.3.2 Member Data Documentation	251
16.3.2.1 count	252
16.3.2.2 initialized	252
16.3.2.3 initval	252
16.3.2.4 start	252
16.3.2.5 var	252
16.3.2.6 vardimids	252
16.3.2.7 vardims	253
16.3.2.8 varid	253
16.3.2.9 varname	253
16.4 pahm_messages::allmessage Interface Reference	253
16.4.1 Detailed Description	254
16.4.2 Member Function/Subroutine Documentation	254
16.4.2.1 allmessage_1()	254
16.4.2.2 allmessage_2()	254
16.5 sortutils::arraycopy Interface Reference	255
16.5.1 Detailed Description	255
16.5.2 Member Function/Subroutine Documentation	255
16.5.2.1 arraycopydouble()	255
16.5.2.2 arraycopyint()	256
16.5.2.3 arraycopsingle()	256
16.6 sortutils::arrayequal Interface Reference	257

16.6.1 Detailed Description	258
16.6.2 Member Function/Subroutine Documentation	258
16.6.2.1 arrayequaldouble()	258
16.6.2.2 arrayequalint()	259
16.6.2.3 arrayequalsingle()	259
16.7 sortutils::arth Interface Reference	260
16.7.1 Detailed Description	260
16.7.2 Member Function/Subroutine Documentation	260
16.7.2.1 arthdouble()	260
16.7.2.2 arthint()	261
16.7.2.3 arthsingle()	262
16.8 parwind::asymmetricvortexdata_t Type Reference	263
16.8.1 Detailed Description	264
16.8.2 Member Data Documentation	264
16.8.2.1 basin	265
16.8.2.2 casttime	265
16.8.2.3 casttype	265
16.8.2.4 casttypenum	265
16.8.2.5 cypress	265
16.8.2.6 day	265
16.8.2.7 dir	266
16.8.2.8 dtg	266
16.8.2.9 ew	266
16.8.2.10 eye	266
16.8.2.11 fcstinc	266
16.8.2.12 filename	266
16.8.2.13 gusts	267
16.8.2.14 hollb	267
16.8.2.15 hollbs	267
16.8.2.16 hour	267
16.8.2.17 icypress	267
16.8.2.18 idir	267
16.8.2.19 ilat	268
16.8.2.20 ilon	268
16.8.2.21 initials	268
16.8.2.22 iprp	268
16.8.2.23 ir	268
16.8.2.24 irmw	268

16.8.2.25 irrp	269
16.8.2.26 isotachspercycle	269
16.8.2.27 ispeed	269
16.8.2.28 istormspeed	269
16.8.2.29 ivr	269
16.8.2.30 lat	269
16.8.2.31 loaded	270
16.8.2.32 lon	270
16.8.2.33 maxseas	270
16.8.2.34 month	270
16.8.2.35 ncycles	270
16.8.2.36 ns	270
16.8.2.37 numcycle	271
16.8.2.38 numrec	271
16.8.2.39 prp	271
16.8.2.40 quadflag	271
16.8.2.41 rmaxw	271
16.8.2.42 rmw	271
16.8.2.43 rrp	272
16.8.2.44 speed	272
16.8.2.45 stormname	272
16.8.2.46 stormnumber	272
16.8.2.47 stormspeed	272
16.8.2.48 subregion	272
16.8.2.49 thisstorm	273
16.8.2.50 trvx	273
16.8.2.51 trvy	273
16.8.2.52 ty	273
16.8.2.53 vmaxesbl	273
16.8.2.54 windcode	273
16.8.2.55 year	274
16.9 parwind::besttrackdata_t Type Reference	274
16.9.1 Detailed Description	275
16.9.2 Member Data Documentation	275
16.9.2.1 basin	276
16.9.2.2 cyclenum	276
16.9.2.3 cnum	276
16.9.2.4 day	276

16.9.2.5 dir	276
16.9.2.6 dtg	276
16.9.2.7 ew	277
16.9.2.8 eye	277
16.9.2.9 filename	277
16.9.2.10 gusts	277
16.9.2.11 hour	277
16.9.2.12 initials	277
16.9.2.13 intlat	278
16.9.2.14 intlon	278
16.9.2.15 intmslp	278
16.9.2.16 intpouter	278
16.9.2.17 intrad1	278
16.9.2.18 intrad2	278
16.9.2.19 intrad3	279
16.9.2.20 intrad4	279
16.9.2.21 intrmw	279
16.9.2.22 introuter	279
16.9.2.23 intspeed	279
16.9.2.24 intvmax	279
16.9.2.25 lat	280
16.9.2.26 loaded	280
16.9.2.27 lon	280
16.9.2.28 maxseas	280
16.9.2.29 month	280
16.9.2.30 ns	280
16.9.2.31 numrec	281
16.9.2.32 rad	281
16.9.2.33 stormname	281
16.9.2.34 subregion	281
16.9.2.35 tau	281
16.9.2.36 tech	281
16.9.2.37 technum	282
16.9.2.38 thisstorm	282
16.9.2.39 ty	282
16.9.2.40 windcode	282
16.9.2.41 year	282
16.10 pahm_sizes::comparereals Interface Reference	283

16.10.1 Detailed Description	283
16.10.2 Member Function/Subroutine Documentation	283
16.10.2.1 comparedoublereals()	283
16.10.2.2 comparesinglereals()	284
16.11 utilities::cpptogeo Interface Reference	285
16.11.1 Detailed Description	285
16.11.2 Member Function/Subroutine Documentation	285
16.11.2.1 cpptogeo_1d()	286
16.11.2.2 cpptogeo_scalar()	286
16.12 csv_module::csv_file Type Reference	288
16.12.1 Detailed Description	289
16.12.2 Member Function/Subroutine Documentation	289
16.12.2.1 add()	290
16.12.2.2 add_cell()	290
16.12.2.3 add_matrix()	290
16.12.2.4 add_vector()	290
16.12.2.5 close()	291
16.12.2.6 csv_get_value()	291
16.12.2.7 destroy()	291
16.12.2.8 get()	291
16.12.2.9 get_character_column()	292
16.12.2.10 get_column()	292
16.12.2.11 get_csv_data_as_str()	292
16.12.2.12 get_csv_string_column()	292
16.12.2.13 get_header()	292
16.12.2.14 get_header_csv_str()	293
16.12.2.15 get_header_str()	293
16.12.2.16 get_integer_column()	293
16.12.2.17 get_logical_column()	293
16.12.2.18 get_real_column()	293
16.12.2.19 initialize()	293
16.12.2.20 next_row()	294
16.12.2.21 open()	294
16.12.2.22 read()	294
16.12.2.23 read_line_from_file()	294
16.12.2.24 tokenize()	294
16.12.2.25 variable_types()	294
16.12.3 Member Data Documentation	295

16.12.3.1 chunk_size	295
16.12.3.2 csv_data	295
16.12.3.3 delimiter	295
16.12.3.4 enclose_all_in_quotes	295
16.12.3.5 enclose_strings_in_quotes	295
16.12.3.6 header	296
16.12.3.7 icol	296
16.12.3.8 iunit	296
16.12.3.9 logical_false_string	296
16.12.3.10 logical_true_string	296
16.12.3.11 n_cols	296
16.12.3.12 n_rows	297
16.12.3.13 quote	297
16.13 csv_module::csv_string Type Reference	297
16.13.1 Detailed Description	297
16.13.2 Member Data Documentation	298
16.13.2.1 str	298
16.14 pahm_netcdff::filedata_t Type Reference	298
16.14.1 Detailed Description	298
16.14.2 Member Data Documentation	299
16.14.2.1 filefound	299
16.14.2.2 filename	299
16.14.2.3 filereccounter	299
16.14.2.4 initialized	299
16.15 pahm_sizes::fixnearwholereal Interface Reference	300
16.15.1 Detailed Description	300
16.15.2 Member Function/Subroutine Documentation	300
16.15.2.1 fixnearwholedoublereal()	300
16.15.2.2 fixnearwholesinglereal()	301
16.16 utilities::geotocpp Interface Reference	302
16.16.1 Detailed Description	302
16.16.2 Member Function/Subroutine Documentation	302
16.16.2.1 geotocpp_1d()	302
16.16.2.2 geotocpp_scalar()	303
16.17 timedateutils::gregtojulday Interface Reference	304
16.17.1 Detailed Description	304
16.17.2 Member Function/Subroutine Documentation	304
16.17.2.1 gregtojulday2()	304

16.17.2.2 gregtojuldayisec()	306
16.17.2.3 gregtojuldayrsec()	307
16.18 parwind::hollanddata_t Type Reference	309
16.18.1 Detailed Description	310
16.18.2 Member Data Documentation	310
16.18.2.1 basin	310
16.18.2.2 casttime	310
16.18.2.3 casttype	310
16.18.2.4 cprdt	311
16.18.2.5 cpress	311
16.18.2.6 day	311
16.18.2.7 dtg	311
16.18.2.8 fcstinc	311
16.18.2.9 filename	311
16.18.2.10 hour	312
16.18.2.11 icpress	312
16.18.2.12 ilat	312
16.18.2.13 ilon	312
16.18.2.14 irmw	312
16.18.2.15 irrp	312
16.18.2.16 ispeed	313
16.18.2.17 lat	313
16.18.2.18 loaded	313
16.18.2.19 lon	313
16.18.2.20 month	313
16.18.2.21 numrec	313
16.18.2.22 rmw	314
16.18.2.23 rrp	314
16.18.2.24 speed	314
16.18.2.25 stormnumber	314
16.18.2.26 thisstorm	314
16.18.2.27 trvx	314
16.18.2.28 trvy	315
16.18.2.29 year	315
16.19 sortutils::indexx Interface Reference	315
16.19.1 Detailed Description	316
16.19.2 Member Function/Subroutine Documentation	316
16.19.2.1 indexxdouble()	316

16.19.2.2 indexxint()	317
16.19.2.3 indexxint8()	318
16.19.2.4 indexxsingle()	319
16.19.2.5 indexxstring()	320
16.20 pahm_messages::logmessage Interface Reference	321
16.20.1 Detailed Description	321
16.20.2 Member Function/Subroutine Documentation	321
16.20.2.1 logmessage_1()	321
16.20.2.2 logmessage_2()	322
16.21 pahm_messages::screenmessage Interface Reference	322
16.21.1 Detailed Description	323
16.21.2 Member Function/Subroutine Documentation	323
16.21.2.1 screenmessage_1()	323
16.21.2.2 screenmessage_2()	323
16.22 utilities::sphericaldistance Interface Reference	324
16.22.1 Detailed Description	324
16.22.2 Member Function/Subroutine Documentation	324
16.22.2.1 sphericaldistance_1d()	325
16.22.2.2 sphericaldistance_2d()	325
16.22.2.3 sphericaldistance_scalar()	326
16.23 timedateutils::splittimetimestring Interface Reference	327
16.23.1 Detailed Description	328
16.23.2 Constructor & Destructor Documentation	328
16.23.2.1 splittimetimestring()	328
16.23.3 Member Function/Subroutine Documentation	329
16.23.3.1 splittimetimestring2()	329
16.24 sortutils::swap Interface Reference	330
16.24.1 Detailed Description	330
16.24.2 Member Function/Subroutine Documentation	331
16.24.2.1 swapdouble()	331
16.24.2.2 swapdoublevec()	331
16.24.2.3 swapint()	332
16.24.2.4 swapintvec()	333
16.24.2.5 swapsingle()	333
16.24.2.6 swapsinglevec()	334
16.25 timedateutils::timeconv Interface Reference	335
16.25.1 Detailed Description	335
16.25.2 Member Function/Subroutine Documentation	335

16.25.2.1 timeconvise()	335
16.25.2.2 timeconvrsec()	336
16.26 pahm_ncdfio::timedata_t Type Reference	338
16.26.1 Detailed Description	338
16.26.2 Member Data Documentation	338
16.26.2.1 initialized	338
16.26.2.2 time	339
16.26.2.3 timedimid	339
16.26.2.4 timedims	339
16.26.2.5 timeid	339
16.26.2.6 timelen	339
17 File Documentation	340
17.1 user-guide/abstract.md File Reference	341
17.2 user-guide/application.md File Reference	341
17.3 user-guide/code.md File Reference	341
17.4 user-guide/credits.md File Reference	341
17.5 user-guide/deliverables.md File Reference	341
17.6 user-guide/evaluation.md File Reference	341
17.7 user-guide/features.md File Reference	341
17.8 user-guide/figures.md File Reference	341
17.9 user-guide/glossary.md File Reference	341
17.10 user-guide/intro.md File Reference	341
17.11 user-guide/models.md File Reference	341
17.12 user-guide/pahm_manual.md File Reference	341
17.13 user-guide/references-dox.md File Reference	341
17.14 user-guide/tables.md File Reference	341
17.15 /home/takis/CSDL/parwinds-doc/src/csv_module.F90 File Reference	341
17.15.1 Detailed Description	343
17.16 csv_module.F90	344
17.17 /home/takis/CSDL/parwinds-doc/src/csv_parameters.F90 File Reference	361
17.17.1 Detailed Description	361
17.18 csv_parameters.F90	361
17.19 /home/takis/CSDL/parwinds-doc/src/csv_utilities.F90 File Reference	362
17.19.1 Detailed Description	362
17.19.2 Function/Subroutine Documentation	363
17.19.2.1 partition()	363
17.19.2.2 quicksort()	364

17.20 csv_utilities.F90	364
17.21 /home/takis/CSDL/parwinds-doc/src/driver_mod.F90 File Reference	367
17.21.1 Detailed Description	368
17.22 driver_mod.F90	368
17.23 /home/takis/CSDL/parwinds-doc/src/global.F90 File Reference	370
17.23.1 Detailed Description	373
17.24 global.F90	373
17.25 /home/takis/CSDL/parwinds-doc/src/mesh.F90 File Reference	376
17.25.1 Detailed Description	377
17.26 mesh.F90	378
17.27 /home/takis/CSDL/parwinds-doc/src/messages.F90 File Reference	381
17.27.1 Detailed Description	383
17.28 messages.F90	383
17.29 /home/takis/CSDL/parwinds-doc/src/netcdfio.F90 File Reference	388
17.29.1 Detailed Description	389
17.29.2 Macro Definition Documentation	389
17.29.2.1 NetCDFCheckErr	389
17.30 netcdfio.F90	390
17.31 /home/takis/CSDL/parwinds-doc/src/pahm.F90 File Reference	400
17.31.1 Detailed Description	400
17.31.2 Function/Subroutine Documentation	401
17.31.2.1 pahm()	401
17.32 pahm.F90	402
17.33 /home/takis/CSDL/parwinds-doc/src/parwind.F90 File Reference	403
17.33.1 Detailed Description	404
17.34 parwind.F90	404
17.35 /home/takis/CSDL/parwinds-doc/src/sizes.F90 File Reference	447
17.35.1 Detailed Description	448
17.36 sizes.F90	448
17.37 /home/takis/CSDL/parwinds-doc/src/sortutils.F90 File Reference	451
17.37.1 Detailed Description	453
17.37.2 Function/Subroutine Documentation	453
17.37.2.1 icompxchg()	453
17.38 sortutils.F90	454
17.39 /home/takis/CSDL/parwinds-doc/src/timedateutils.F90 File Reference	473
17.39.1 Detailed Description	475
17.40 timedateutils.F90	475
17.41 /home/takis/CSDL/parwinds-doc/src/utilities.F90 File Reference	489

17.41.1 Detailed Description	490
17.42 utilities.F90	491
17.43 /home/takis/CSDL/parwinds-doc/src/vortex.F90 File Reference	522
17.43.1 Detailed Description	524
17.44 vortex.F90	524
Index	546

Chapter 1

PaHM Manual

Parametric Hurricane Modeling System

User's Guide

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

February 2022

Chapter 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 is an [ESMF/NUOPC](#) compatible modeling system that can be used either as a standalone atmospheric model, or as an atmospheric modeling component 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 Models ([1980], [2010]) and the Generalized Holland Parametric Tropical Cyclone Model (Gao et al. [2015], [2018]).

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>.

Chapter 3

List of Tables

Chapter 4

List of Figures

Chapter 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.

5.1 The Parametric Hurricane Modeling System (*PaHM*)

The Parametric Hurricane Modeling System is not an atmospheric model but rather an atmospheric 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 core parametric models in *PaHM* are the [Holland models \(1980, 2010\)](#) and the [Generalized Asymmetric Vortex Model \(GAHM\)](#). Two additional parametric modeling components are in development namely, the [Rankine Vortex Model](#) and the [Willoughby Model](#).

PaHM 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. Furthermore, *PaHM*'s built-in GRIB1/2 and NetCDF interface can also be used to read other external atmospheric data products.

5.2 Downloading *PaHM*

The latest source code of *PaHM* can be downloaded from its Git repository at: <https://github.com/noaa-ocs-modeling/PaHM>. Binary distributions of *PaHM* are not currently available. The online documentation of the modeling system is hosted by GitHub Pages: <https://noaa-ocs-modeling.github.io/PaHM/html/index.html>. The PDF version of the online documentation can be downloaded from: https://noaa-ocs-modeling.github.io/PaHM/pahm_manual.pdf.

New Git users are invited to read some online guides to get familiar with vanilla Git concepts and commands:

- Basic and advanced guide with the [Git Book](#).
- Reference guides with the [Git Reference](#).
- GitHub reference sheets with the [GitHub Reference](#).
- Manage your GitHub repositories with Git [Using Git](#).

From *PaHM*'s Git repository you can clone or download the the source code as follows:

- Clone the source using the command:

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

- The source can be downloaded directly from:

```
https://github.com/noaa-ocs-modeling/PaHM/archive/refs/heads/main.zip
```

5.2.1 Directory Structure

Now after downloading the *PaHM*, let us look at the physical directories and the configuration files that come with the system. In the *PaHM* ROOT directory contains the source and all configuration files required to build and run *PaHM* (see [Figure 1](#)). The directories of special interest to us are `src`, `scripts`, `cmake` and `inputs`.

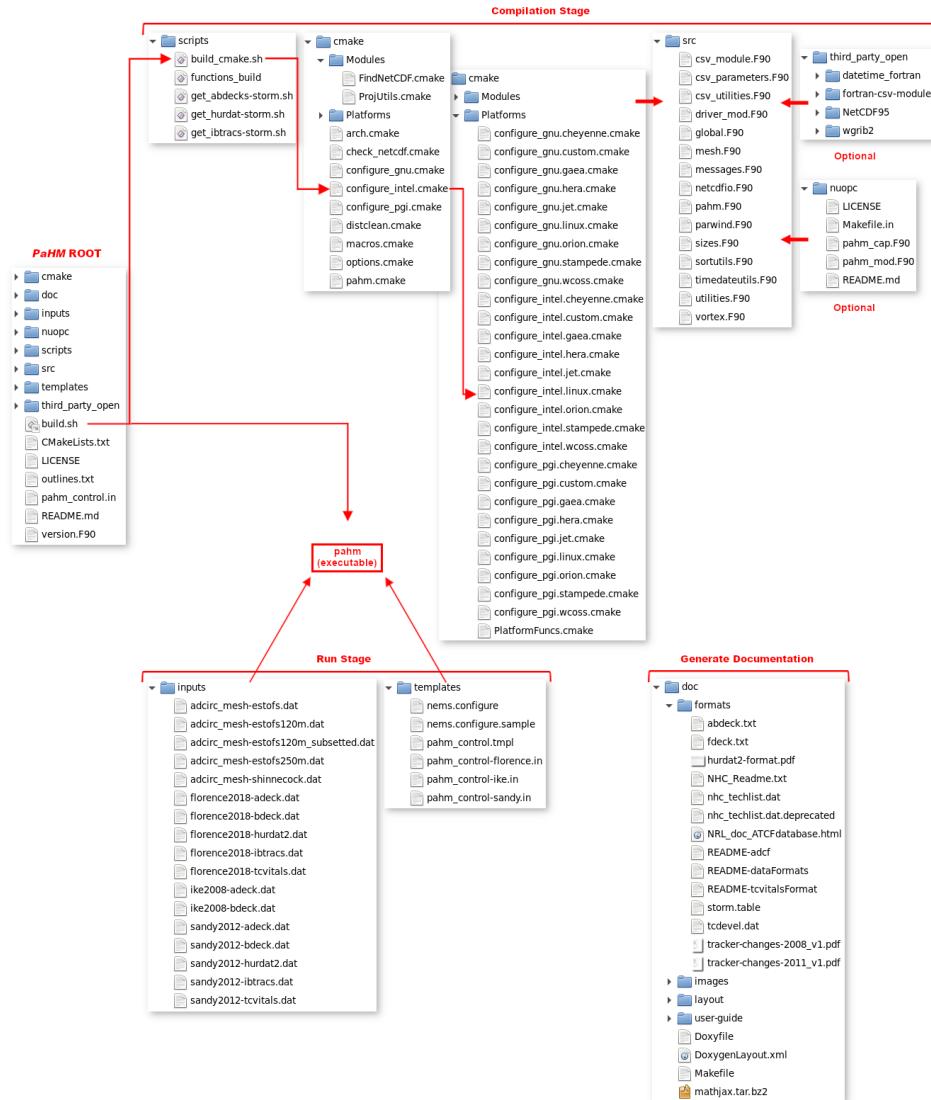


Figure 1: Directory tree of the *PaHM* modeling system.

- **src**
Contains all the Fortran code of *PaHM*
- **build**
- **cmake**
- **third_party_open**
- **nuopc**

- inputs
- templates
- doc

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|--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, wcoss.
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

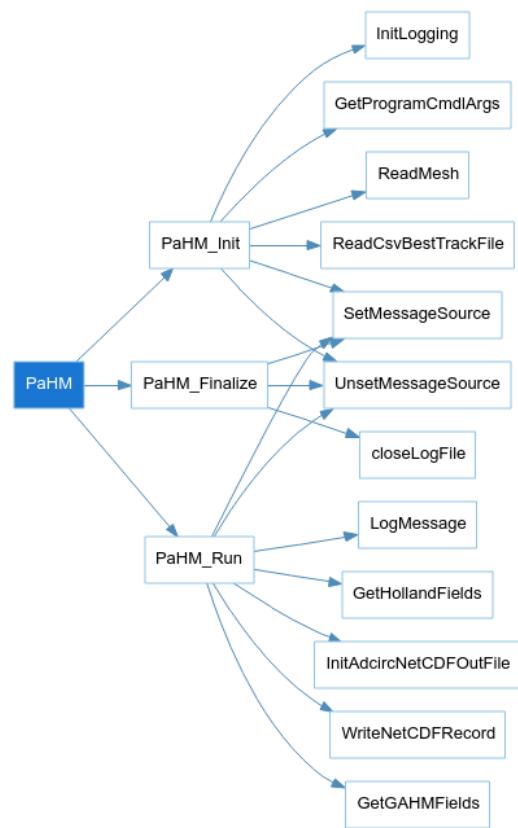
Installation and development of *PaHM* is done through the distributed version control system Git. Even if a tarball could be sufficient, we advise to use Git system to follow *PaHM* development and merge easily to new versions. Building *PaHM* from sources requires to compile third party libraries and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

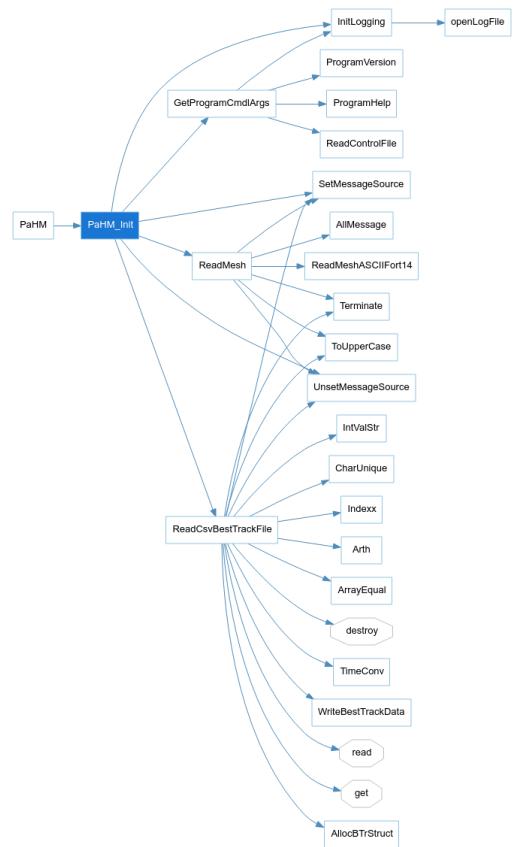
5.4 Using *PaHM*

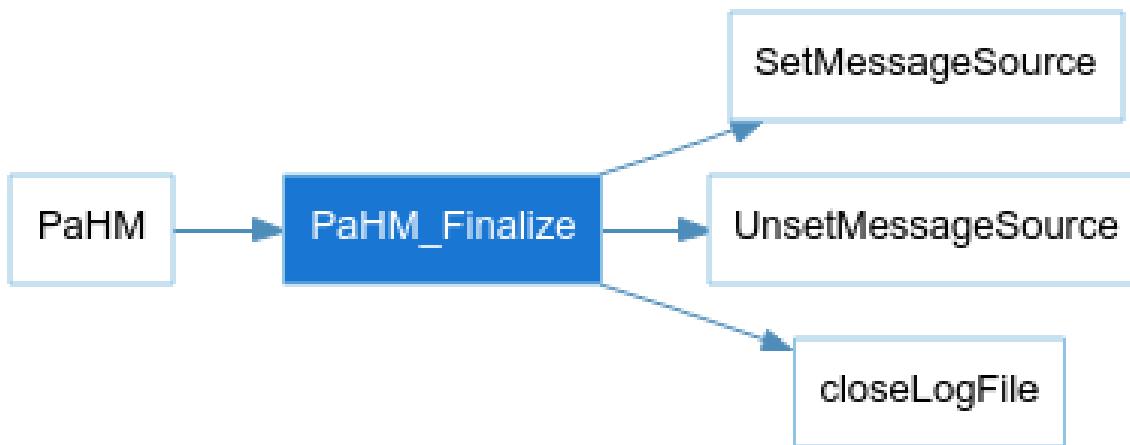
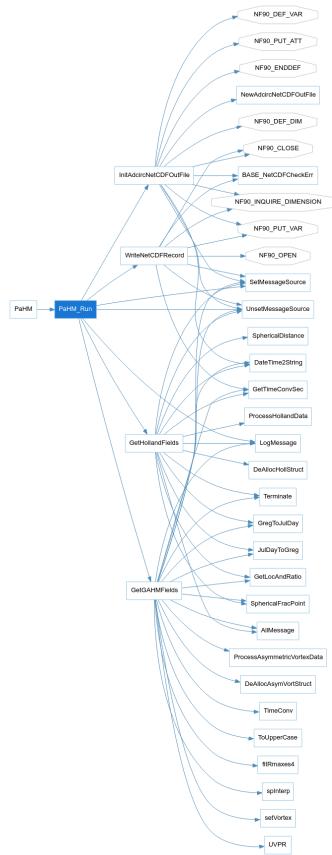
PaHM can accommodate the presence of multiple storms in the basin (defined by the input of multiple \u201cbest track\u201d files) \u2022

Inputs: a \u201cstorm track\u201d and a \u201cgrid/mesh\u201d \u2022 Procedure: \u2022 Converts the \u201cbest track\u201d 10-m wind values to gradient wind values. \u2022 Applies the parametric model (Holland) to generate the wind fields at the gradient level. \u2022 Converts the Pham generated wind fields to 10-m winds. \u2022 Writes the data to a NetCDF-4 file.

Wind speeds outside the last closed isobar are set to zero, while the atmospheric pressure is set equal to 1013.25 mb \u2022 In the presence of multiple storms *PaHM* considers the possible interaction between the storms when generating its gridded wind fields \u2022 Atmospheric fields are shared with the ocean and wave models using *PaHM*\u2019s NUOPC/ESMF Output data: \u2022 \u201cGridded wind fields in NetCDF, CF compliant format \u2022 Corrected\u201d best track data files Required inputs: \u2022 *PaHM*\u2019scontrol file \u2022 List of \u201cbest track\u201d files for the basin \u2022 The grid/mesh file to generate wind fields for







5.4.1 Standalone Configuration

Standalone modeling system that reads “best track” type files (e.g., National Hurricane Center) to generate gridded atmospheric fields (10-m wind velocities, MSL atm. pressure) Formats recognized: a/b-decks,

HurDat2, IBTrACS, TCVitals (PaHM has a csv I/O interface therefore, it can read and write any “csv” formatted “track” file) Uses symmetric and asymmetric vortex parametric TC models to generate its wind fields Currently: Symmetric models: Holland 1998 (functional), Holland 2010 (functional), Willoughby 2004 (in progress); Asymmetric models: Generalized Holland 2015 (GAHM, in progress) PaHM can accommodate the presence of multiple storms in the basin (defined by the input of multiple “best track” files) The system has a GRIB1/2 and NetCDF interface to read external atmospheric data products

5.4.2 Coupling Configuration

The coupled modeling approach described here allows to address the impacts of extreme storm events such as hurricanes on coastal areas. The HSOFS modeling system is applied and evaluated for Hurricane Florence on the Eastern coast of the US that includes the important basins of Delaware Bay, Chesapeake Bay and the Carolinas. While individual modeling components (ADCIRC, WAVEWATCH III) have already been evaluated successfully using standard statistical measures, PAHM is currently evaluated as part of the overall HSOFS evaluation. Initial HSOFS simulations show promising results on predicting total water level and flood inundation.

Initial work has been done to couple the ADCIRC hydrodynamic surge model with the WAVEWATCH III wave model (Moghimi et al. [2019, 2020]) within the NOAA Environmental Modeling System (NEMS), a common modeling framework for coupling and sharing information between NOAA’s models. This coupled wave-surge modeling system is tested by enhancing it for use in an operational setting with the HSOFS grid and modeling system. A key component of this implementation is the inclusion of parametric hurricane wind forcings. To accomplish this in NEMS, a National Unified Operational Prediction Capability (NUOPC) cap has been developed to exchange the wind forcing information with both ADCIRC and WAVEWATCH III within the coupled framework.

Chapter 6

Parametric Models in PaHM

PaHM generates its wind fields at the gradient level. The gradient level is roughly 1 km above the surface of the earth, and is the level most representative of the air flow in the lower atmosphere immediately above the layer affected by surface friction. This level is free of local wind and topographic effects (such as sea breezes, downslope winds etc). The 10-m wind may be estimated by decreasing the gradient level wind speed by approximately 20% over the ocean, 40% over land.

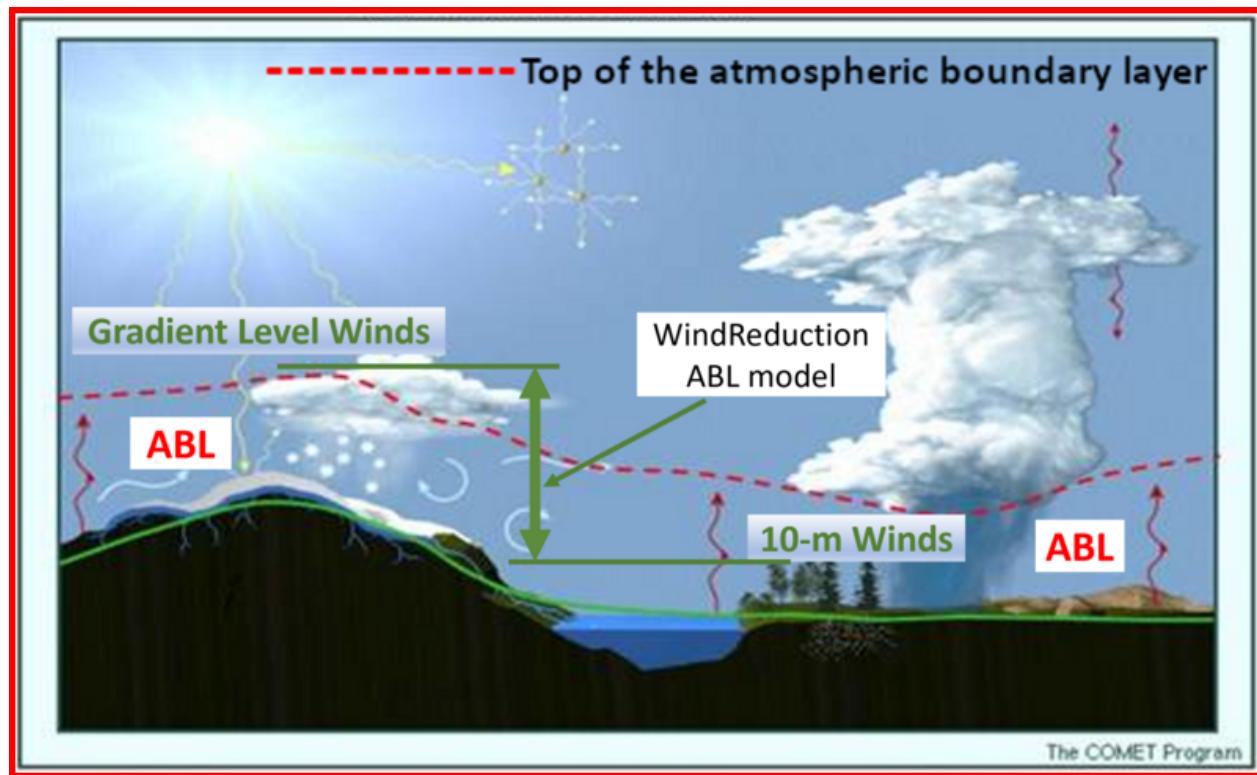


Figure 6.1 Figure1:

6.1 Rankine Vortex Model

In Development ...

6.2 Holland Symmetric Vortex Models

6.3 Willoughby Symmetric Vortex Model

In Development ...

6.4 Generalized Asymmetric Vortex Holland model (*GAHM*)

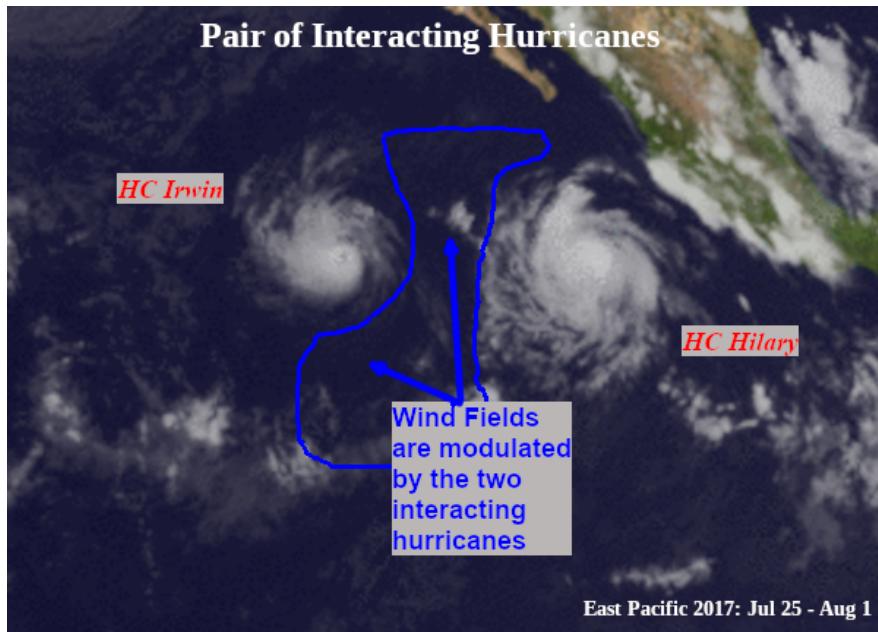
Chapter 7

PaHM Features and Capabilities

7.1 Data Input Interfaces

7.2 Model Grids

7.3 Modeling Multiple Interacting Storms



7.4 Coupling Environment

Chapter 8

Model Application and Implementation Technology

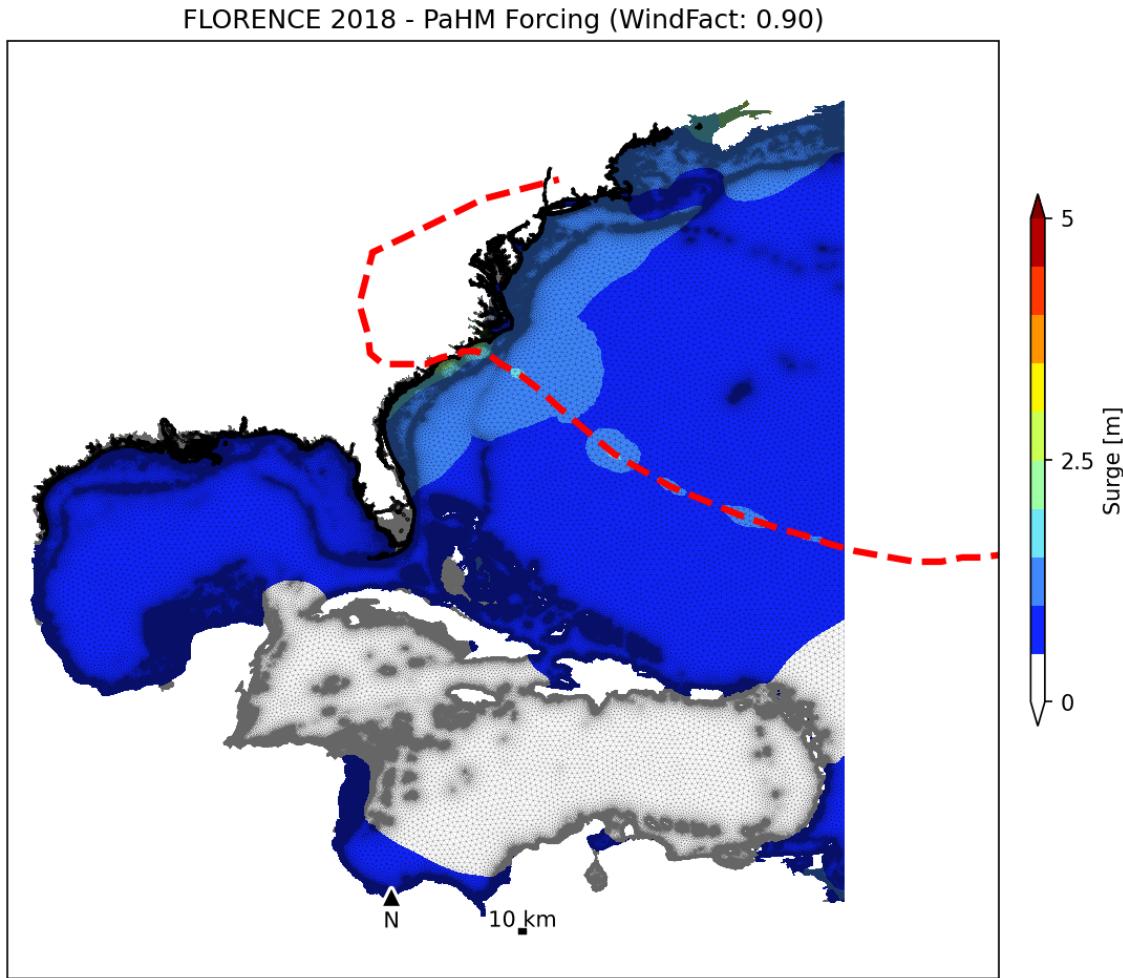
8.1 Standalone Model Application

8.2 Coupled Model Application

Chapter 9

Model Evaluation - Hurricane Florence (2018) Study

The coupled modeling approach described here allows to address the impacts of extreme storm events such as hurricanes on coastal areas. The HSOFS modeling system is applied and evaluated for Hurricane Florence on the Eastern coast of the US that includes the important basins of Delaware Bay, Chesapeake Bay and the Carolinas. While individual modeling components (ADCIRC, WAVEWATCH III) have already been evaluated successfully using standard statistical measures, PAHM is currently evaluated as part of the overall HSOFS evaluation. Initial HSOFS simulations show promising results on predicting total water level and flood inundation.



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 (R^2), (e) the bias ($bias$), (f) the Willmott (2012) skill ($WS12$ skill) and (g) the Nash and Sutcliff (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:

$$m = \frac{\sum_{i=1}^n (M_i - O_i)}{n} \quad (9.1)$$

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:

$$SD = \sqrt{\frac{\sum_{i=1}^n [(M_i - O_i) - m]^2}{n}} \quad (9.2)$$

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:

$$RMSE = \sqrt{\frac{\sum_{i=1}^n (M_i - O_i)^2}{n}} \quad (9.3)$$

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 R^2 , 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 R^2 value indicates that the model is a good fit for the data. An $R^2 = 0.62$, 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.

$$R = \frac{\sum_{i=1}^n (O_i - \bar{O})(M_i - \bar{M})}{\sqrt{\sum_{i=1}^n (O_i - \bar{O})^2(M_i - \bar{M})^2}} \quad (9.4)$$

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.

$$bias = \bar{O} - \bar{M} \quad (9.5)$$

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:

$$SM1 = \sum_{i=1}^n O_i - M_i; \quad SM2 = \sum_{i=1}^n M_i - \bar{O} \quad O_i - \bar{O} \quad WS12skill = 1.0 - SM1/(2.0 \cdot SM2) \quad for \quad SM1 \leq 2.0 \cdot SM2 \quad (9.6)$$

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

$$0.8WS12 \leq 1.0 \quad Excellent \quad 0.6WS12 \leq 0.8 \quad Good \quad 0.3WS12 \leq 0.6 \quad Reasonable \quad 0.0WS12 \leq 0.3 \quad Poor \quad WS12 \leq 0.0 \quad Bad \quad (9.7)$$

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:

$$NS_{skill} = 1.0 - \frac{\sum_{i=1}^n (M_i - O_i)^2}{\sum_{i=1}^n (O_i - \bar{M})^2} \quad (9.8)$$

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 9.1 Caption Text

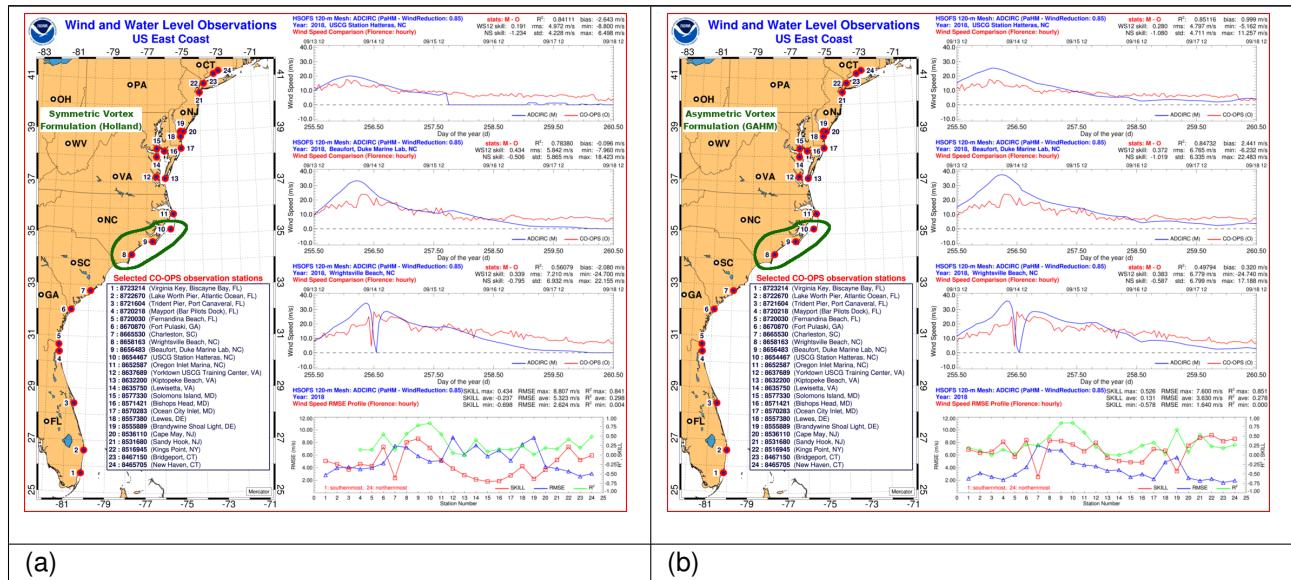


Table 9.2 Caption Text

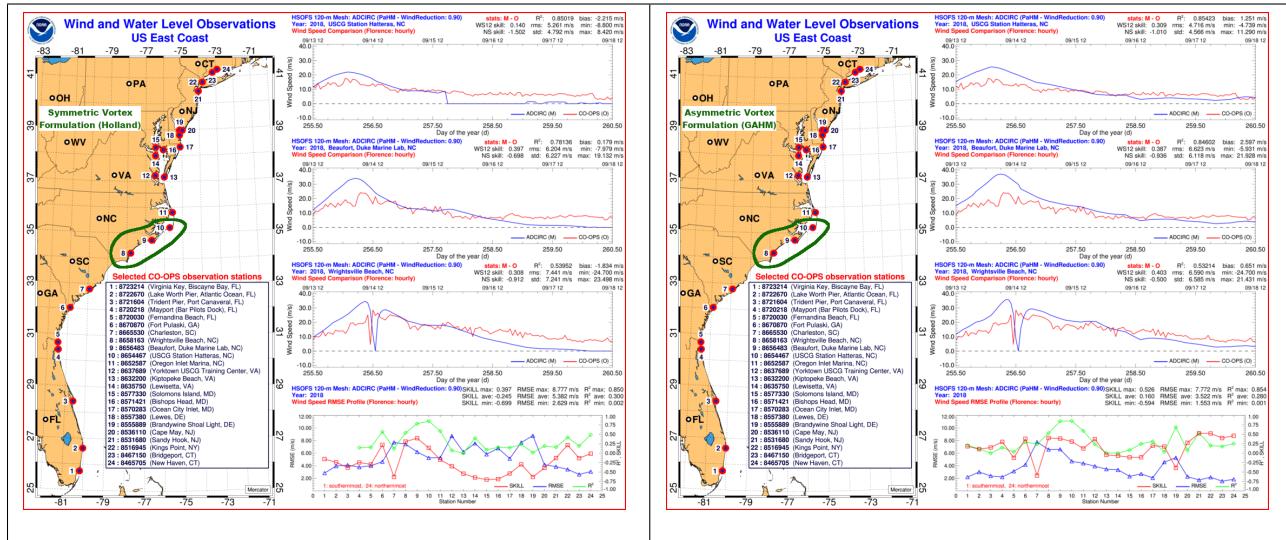
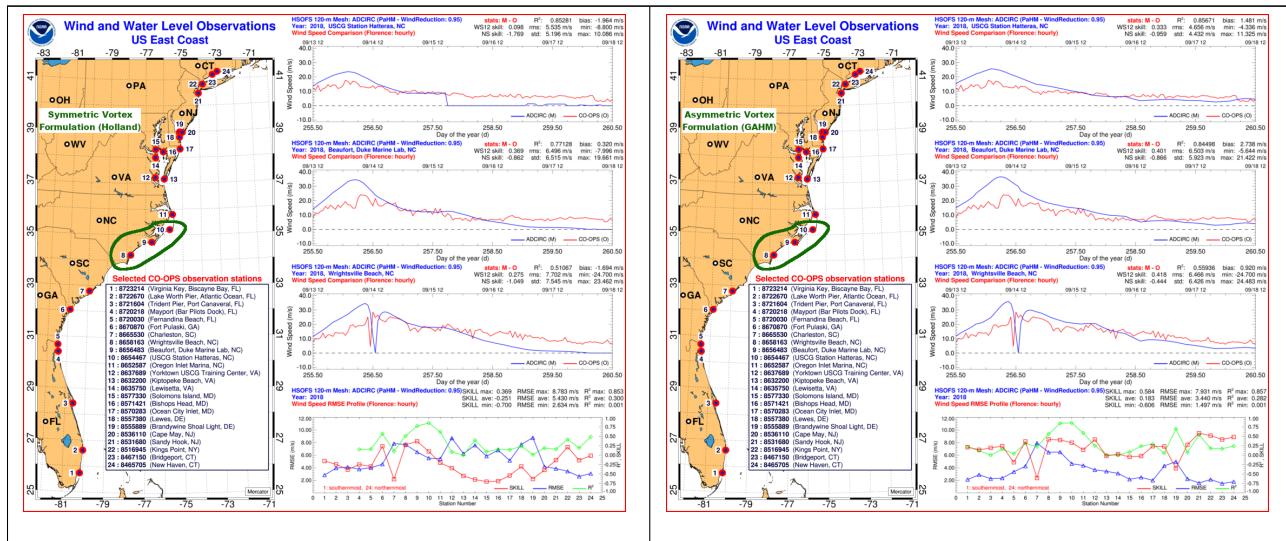


Table 9.3 Caption Text



9.3 Coupled Model Evaluation

Table 9.4 Caption Text

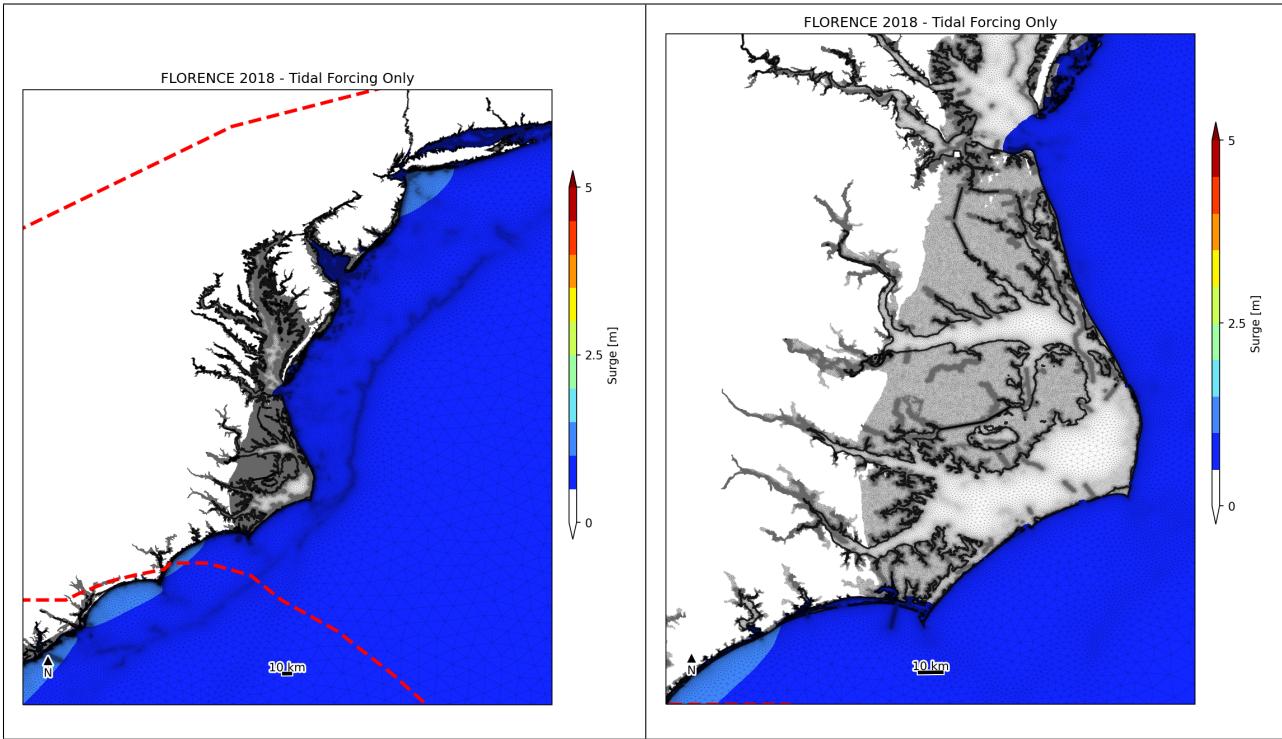


Table 9.5 Caption Text

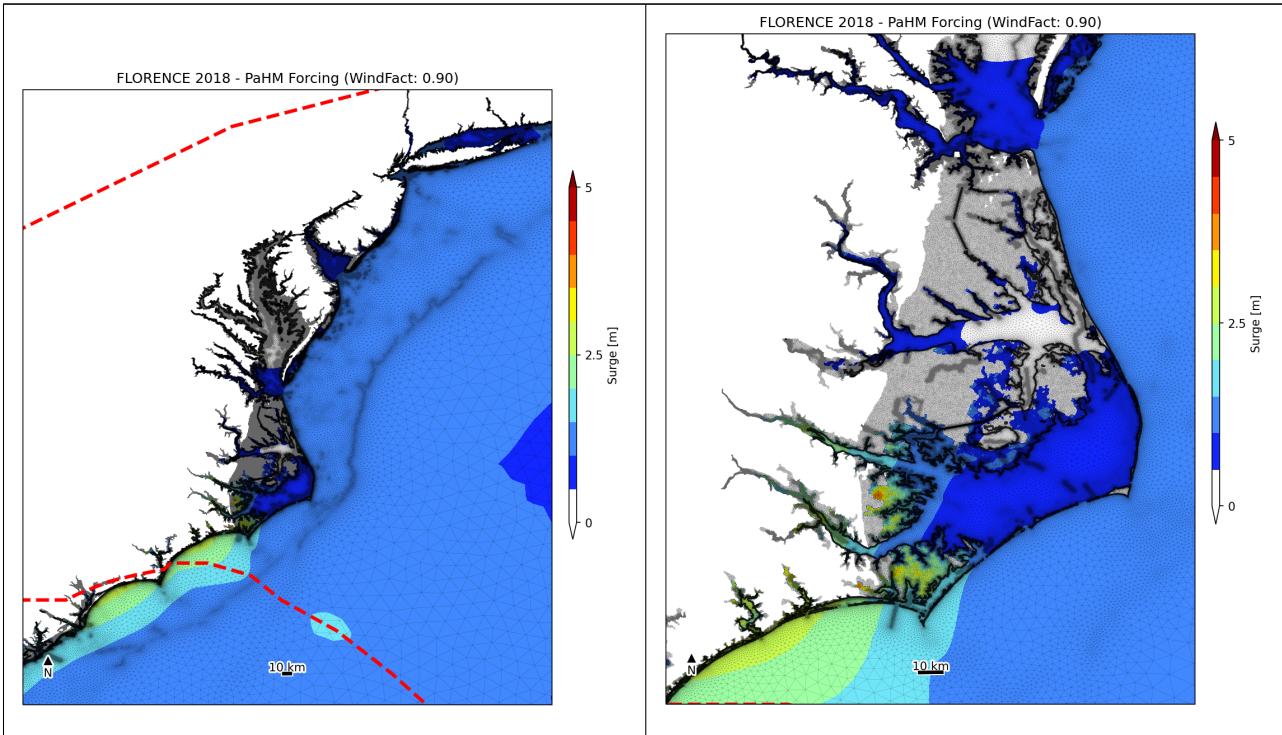
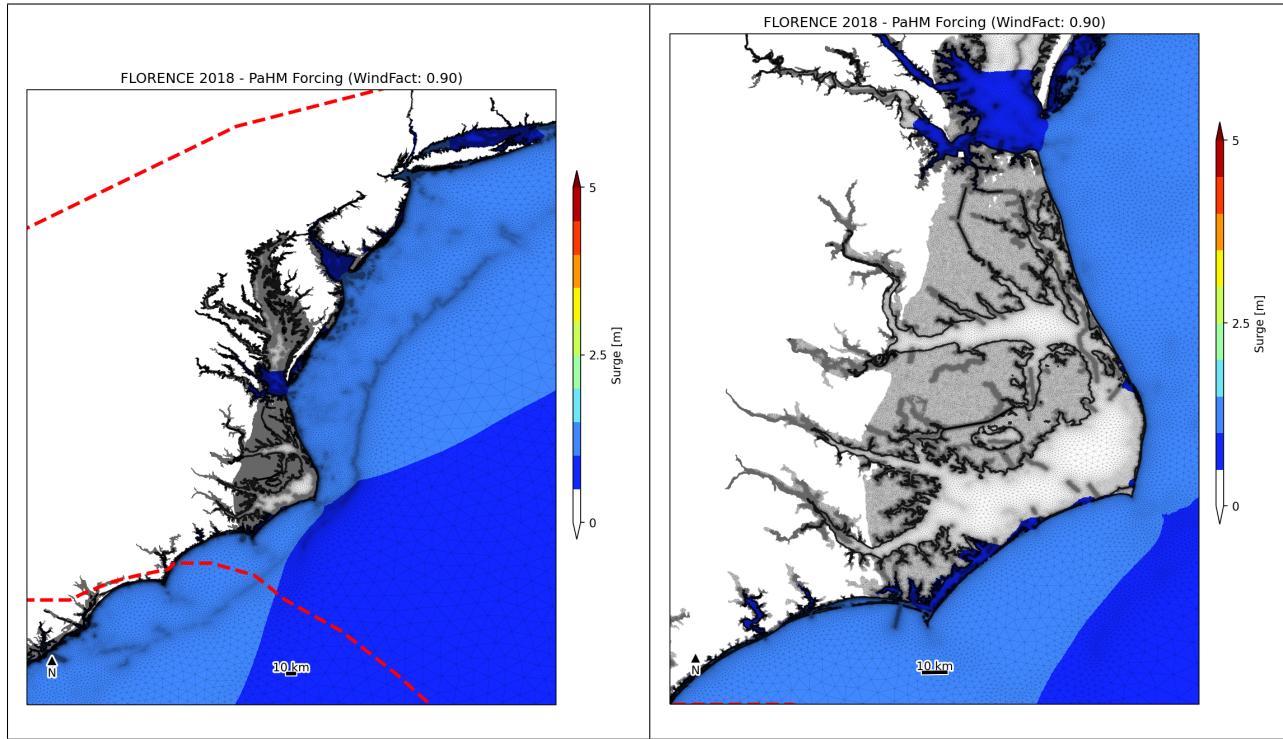


Table 9.6 Caption Text



9.3.1 Coupled Model Results and Discussion

Table 9.7 Caption Text

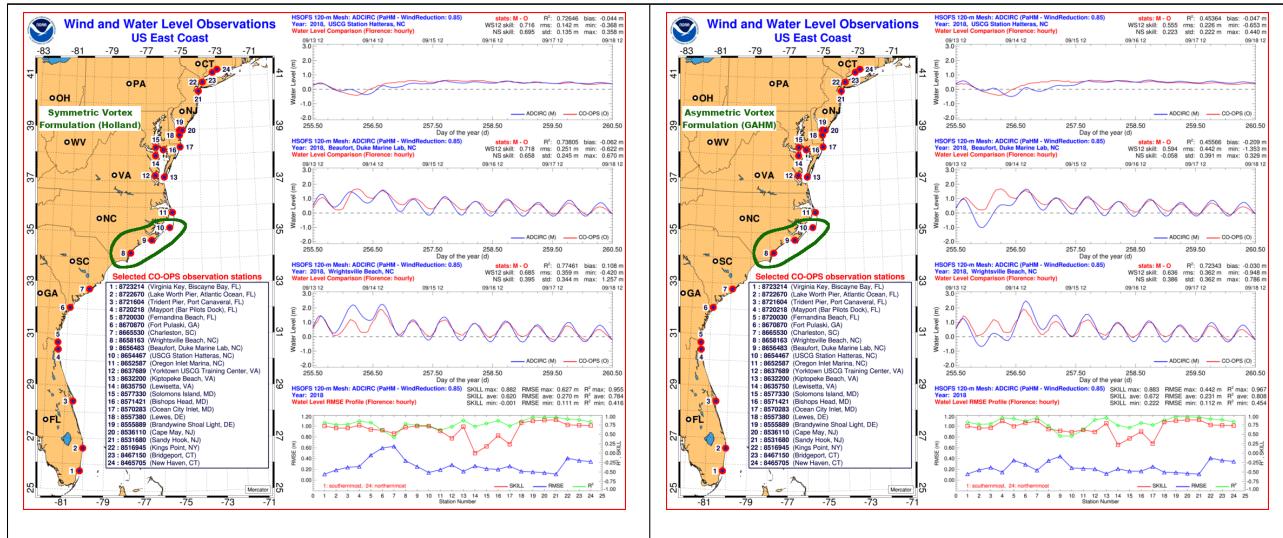


Table 9.8 Caption Text

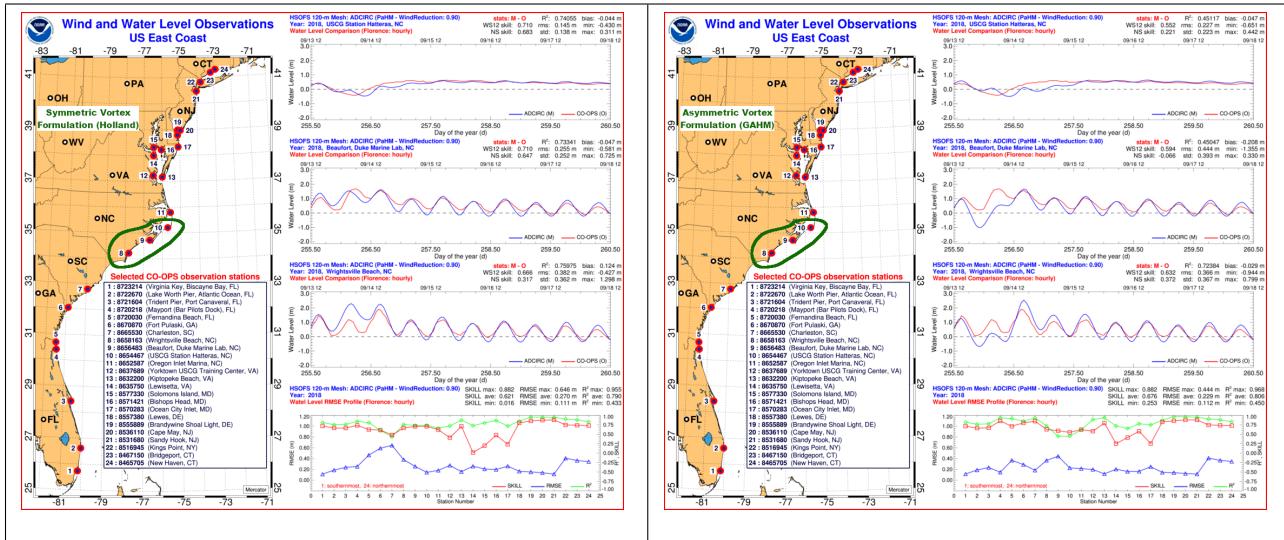
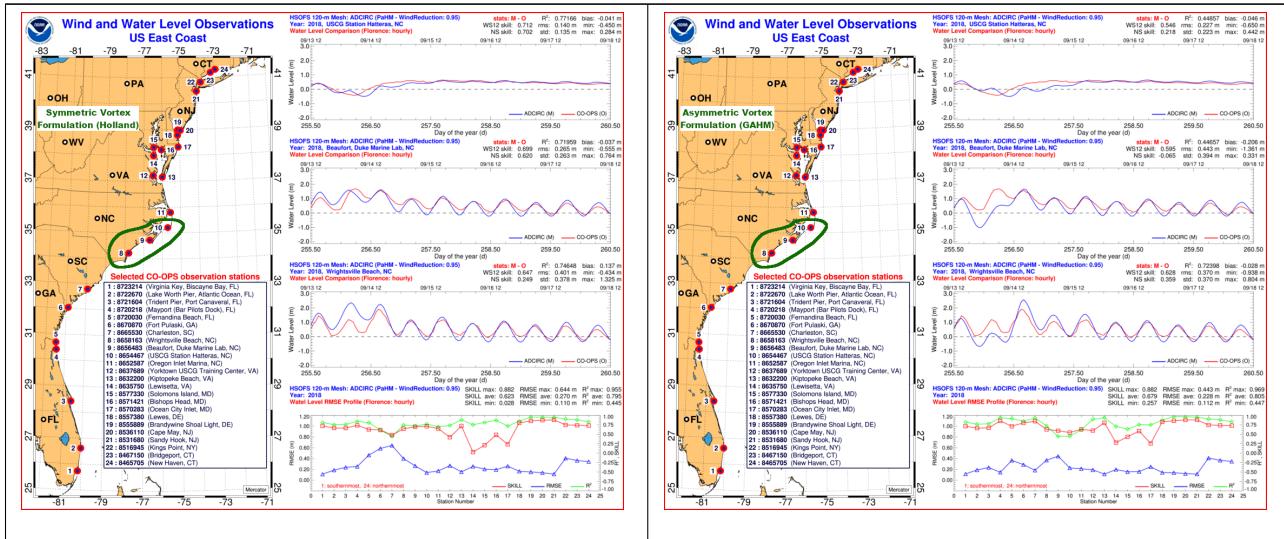


Table 9.9 Caption Text



9.4 Conclusions

Chapter 10

List of Deliverables

Chapter 11

Glossary

Chapter 12

Credits

Chapter 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.

Chapter 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

Chapter 15

Module Documentation

15.1 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

15.1.1 Function/Subroutine Documentation

15.1.1.1 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

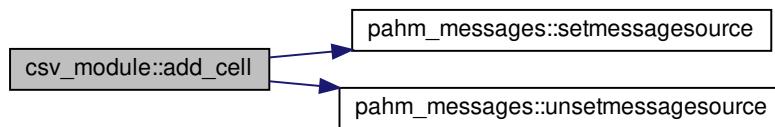
in,out	<i>me</i>	The input/output csv_file class
in	<i>val</i>	The value to add
in	<i>int_fmt</i>	If 'val' is an integer, use this format string (optional)
in	<i>real_fmt</i>	If 'val' is a real, use this format string (optional)
out	<i>trim_str</i>	If 'val' is a string, then trim it (optional)

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:



15.1.1.2 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

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



15.1.1.3 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.

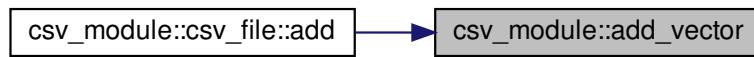
Parameters

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



15.1.1.4 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/output csv_file class
out	<i>status_ok</i>	Status flag

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

15.1.1.5 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

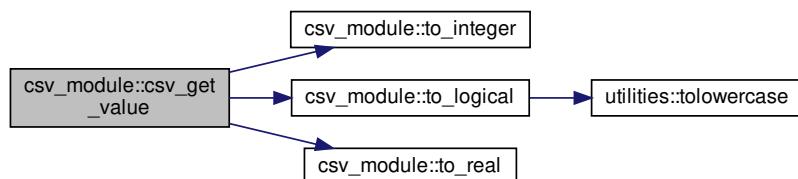
in,out	<i>me</i>	The input/output csv_file class
in	<i>row</i>	The row number
in	<i>col</i>	The column number
out	<i>val</i>	The returned value
out	<i>status_ok</i>	Status flag

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

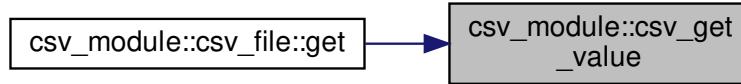
References [to_integer\(\)](#), [to_logical\(\)](#), [to_real\(\)](#), and [pahm_sizes::wp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.6 destroy_csv_file()

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

Destroy a [[csv_file(type)]].

Parameters

out	<i>me</i>	The ouput <code>csv_file</code> class
-----	-----------	---------------------------------------

Definition at line 229 of file `csv_module.F90`.

15.1.1.7 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

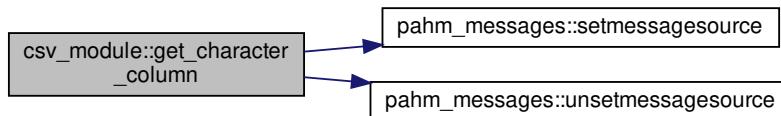
in,out	<i>me</i>	The input/ouput <code>csv_file</code> class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

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

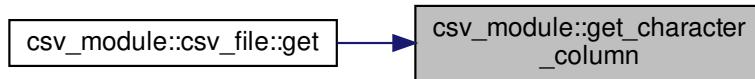
References `pahm_messages::error`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.8 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 <code>csv_file</code> 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

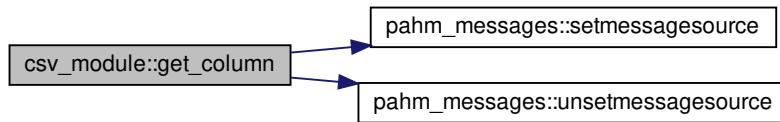
Note

This routine requires that the 'r' array already be allocated. This is because Fortran doesn't want to allow you to 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:

**15.1.1.9 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

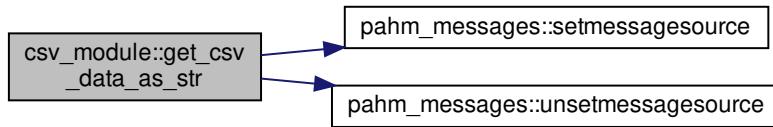
in,out	<i>me</i>	The input/output csv_file class
out	<i>csv_data</i>	The data
out	<i>status_ok</i>	Status flag

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

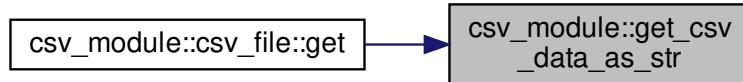
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.10 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

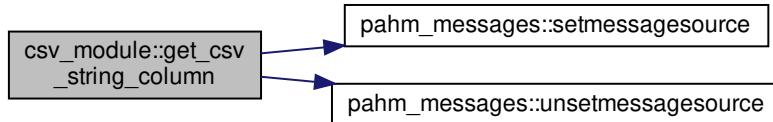
in,out	<i>me</i>	The input/output <code>csv_file</code> class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

Definition at line 1524 of file `csv_module.F90`.

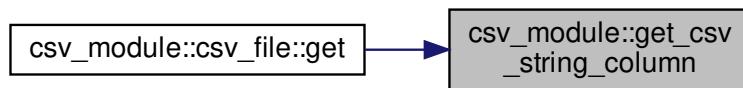
References `pahm_messages::error`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.11 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

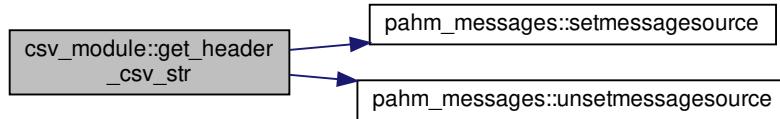
in,out	<i>me</i>	The input/ouput csv_file class
out	<i>header</i>	The header of the CSV file
out	<i>status_ok</i>	Status flag

Definition at line 799 of file [csv_module.F90](#).

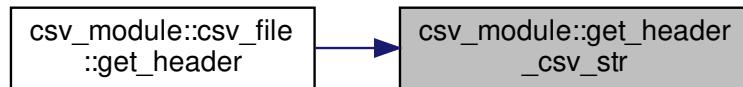
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.12 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

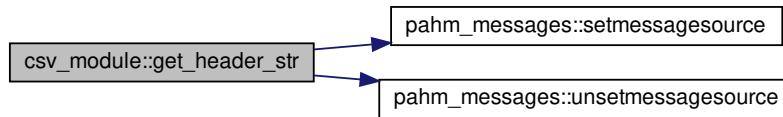
in,out	<i>me</i>	The input/output csv_file class
out	<i>header</i>	The header of the CSV file
out	<i>status_ok</i>	Status flag

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

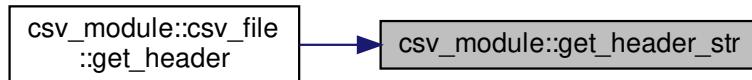
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.13 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

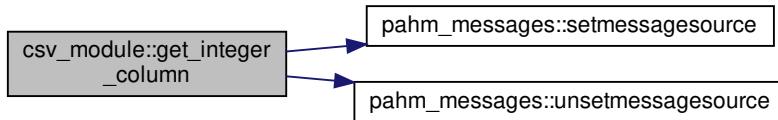
in,out	<i>me</i>	The input/output csv_file class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

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

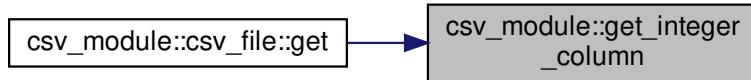
References `pahm_messages::error`, `pahm_messages::scratchmessage`, `pahm_messages::setmessagesource()`, and `pahm_messages::unsetmessagesource()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.14 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

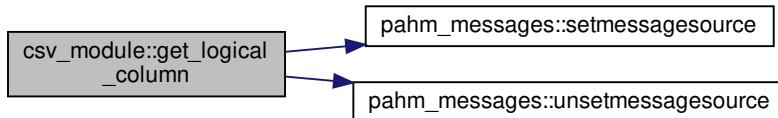
<code>in,out</code>	<code>me</code>	The input/output <code>csv_file</code> class
<code>in</code>	<code>icol</code>	The column number
<code>out</code>	<code>r</code>	The returned column
<code>out</code>	<code>status_ok</code>	Status flag

Definition at line 1436 of file [csv_module.F90](#).

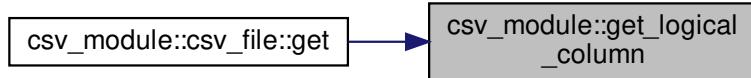
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.15 get_real_column()

```

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

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

Parameters

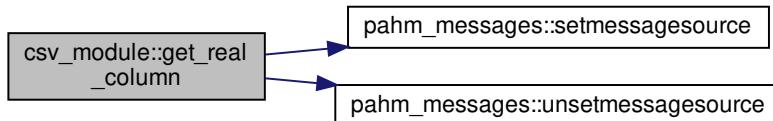
in,out	<i>me</i>	The input/output csv_file class
in	<i>icol</i>	The column number
out	<i>r</i>	The returned column
out	<i>status_ok</i>	Status flag

Definition at line 1348 of file [csv_module.F90](#).

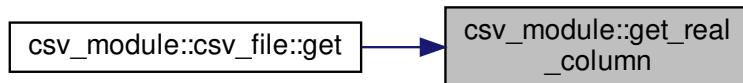
References [pahm_messages::error](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.16 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

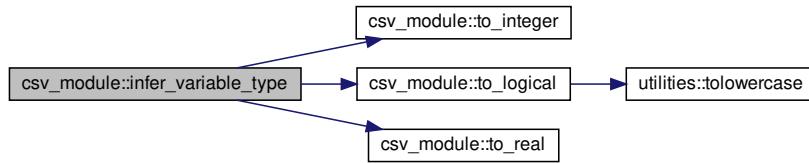
<code>in</code>	<code>str</code>	The input string
<code>out</code>	<code>itype</code>	The type of the cell value <small>Generated by Doxygen</small> <ul style="list-style-type: none"> 1: a character string cell 2: a 'real(wp)' cell 3: an 'integer(ip)' cell 4: a logical cell

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

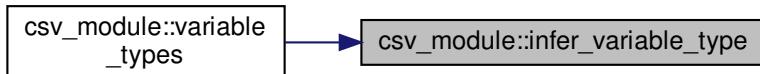
References [csv_type_double](#), [csv_type_integer](#), [csv_type_logical](#), [csv_type_string](#), [to_integer\(\)](#), [to_logical\(\)](#), and [to_real\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.17 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

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

Parameters

in	<i>quote</i>	Can only be one character (optional, default is "")
in	<i>delimiter</i>	Can only be one character (optional, default is ',')
in	<i>enclose_strings_in_quotes</i>	Logical flag; if true, all string cells will be enclosed in quotes (optional, default is 'T')
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](#).

15.1.1.18 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](#).

15.1.1.19 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

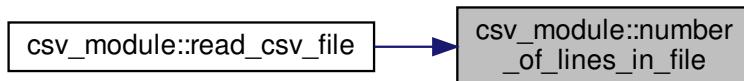
Note

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:

**15.1.1.20 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.

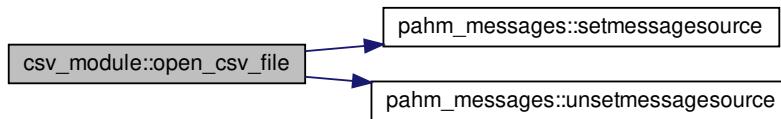
Parameters

<code>in,out</code>	<code>me</code>	The input/output csv_file class
<code>in</code>	<code>filename</code>	The CSV file to open
<code>in</code>	<code>n_cols</code>	The number of columns in the file
<code>out</code>	<code>status_ok</code>	Status flag
<code>in</code>	<code>append</code>	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:



15.1.1.21 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.

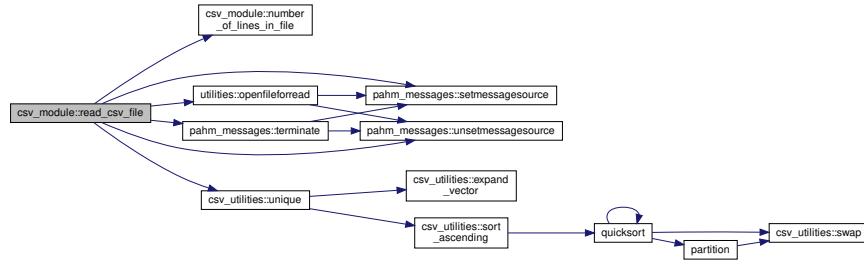
Parameters

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:



15.1.1.22 `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.

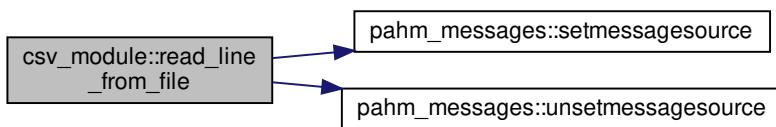
Parameters

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



15.1.1.23 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

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

Warning

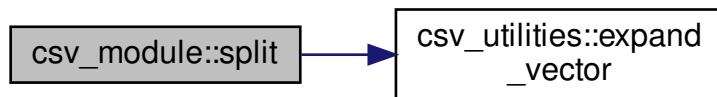
Does not account for tokens contained within quotes string

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

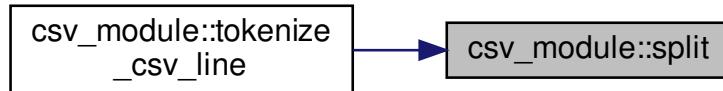
References [csv_utilities::expand_vector\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.1.1.24 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

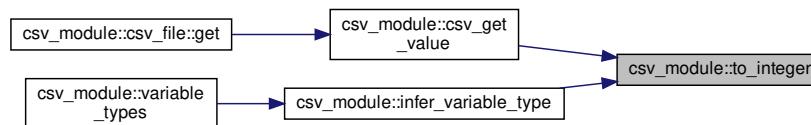
in	<i>str</i>	The input string
out	<i>val</i>	The converted value
out	<i>status_ok</i>	Status flag

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:



15.1.1.25 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

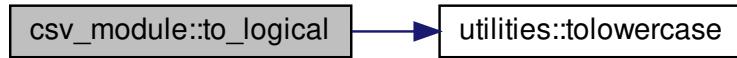
in	<i>str</i>	The input string
out	<i>val</i>	The converted value
out	<i>status_ok</i>	Status flag

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

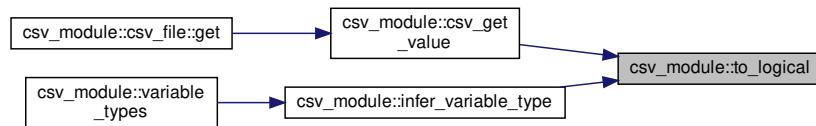
References [utilities::tolowercase\(\)](#).

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:



15.1.1.26 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

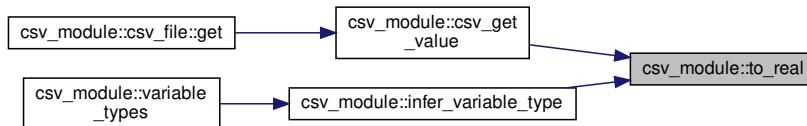
in	<i>str</i>	The input string
out	<i>val</i>	The converted value
out	<i>status_ok</i>	Status flag

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:



15.1.1.27 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

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

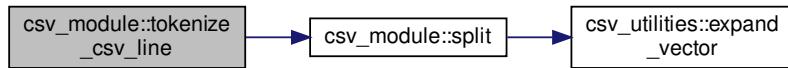
Warning

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:

**15.1.1.28 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

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

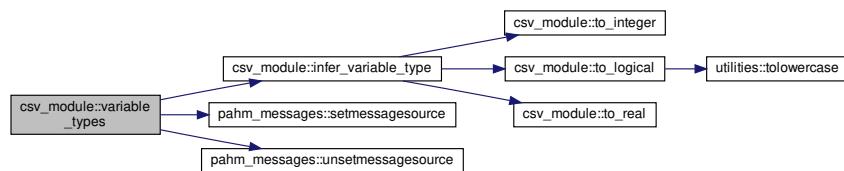
Note

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:



15.1.2 Variable Documentation

15.1.2.1 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\(\)](#).

15.1.2.2 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\(\)](#).

15.1.2.3 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\(\)](#).

15.1.2.4 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\(\)](#).

15.1.2.5 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\(\)](#).

15.2 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)'

15.2.1 Variable Documentation

15.2.1.1 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\(\)](#).

15.2.1.2 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\(\)](#).

15.2.1.3 max_integer_str_len

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

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

15.2.1.4 max_real_str_len

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

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

15.3 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

15.3.1 Function/Subroutine Documentation

15.3.1.1 `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.

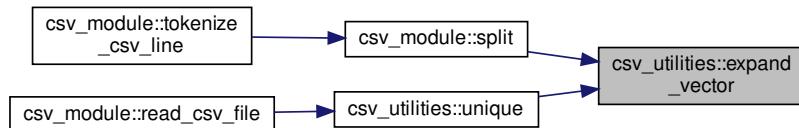
Parameters

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



15.3.1.2 sort_descending()

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

Sorts an integer array `ivec` in decreasing order.

Uses a basic recursive quicksort (with insertion sort for partitions with ≤ 20 elements).

Parameters

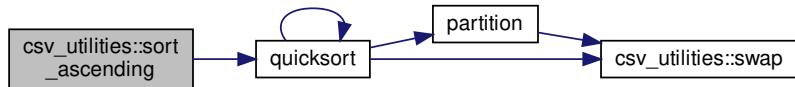
in,out	<code>ivec</code>	A vector of integers
--------	-------------------	----------------------

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

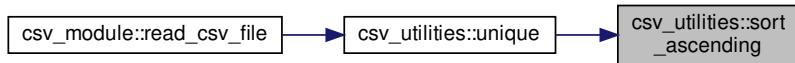
References [quicksort\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.3.1.3 swap()

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

Swap two integer values.

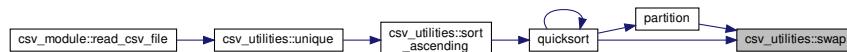
Parameters

in, out	<i>i1</i>	The first integer value to swap
in, out	<i>i2</i>	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:

**15.3.1.4 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

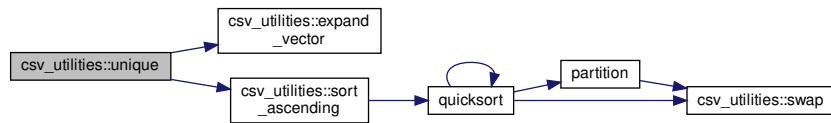
ivec_unique The unique elements of the vector "vec"

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

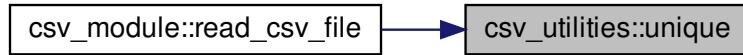
References [expand_vector\(\)](#), and [sort_descending\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.3.2 Variable Documentation

15.3.2.1 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\(\)](#).

15.4 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](#)

15.4.1 Function/Subroutine Documentation

15.4.1.1 `getprogramcmdlargs()`

```
subroutine pahm_drivermod::getprogramcmdlargs
```

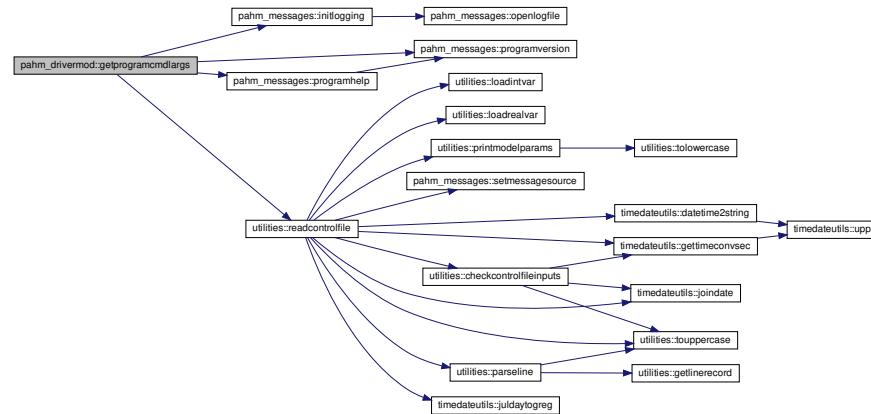
Prints on the screen the help system of the PaHM program.

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.4.1.2 pahm_finalize()

```
subroutine pahm_drivermod::pahm_finalize
```

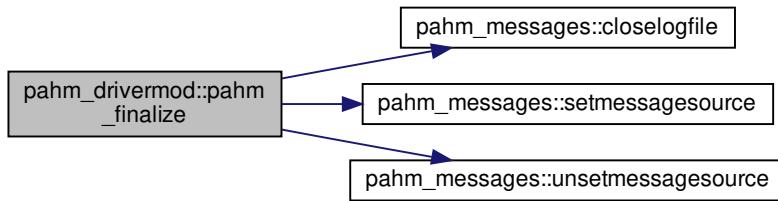
Subroutine to finalize a PaHM run.

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

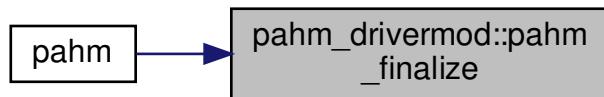
References [pahm_messages::closelogfile\(\)](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#)

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.4.1.3 pahm_init()

```
subroutine pahm_drivermod::pahm_init
```

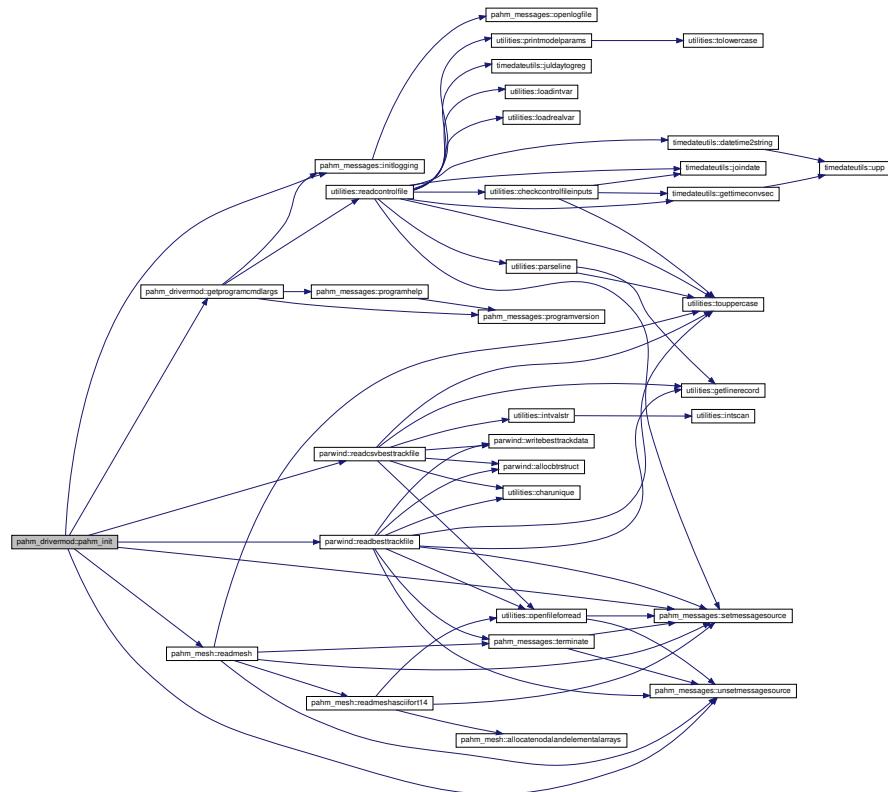
Subroutine to initialize a PaHM run.

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:



15.4.1.4 pahm_run()

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

Subroutine to run PaHM (timestepping).

Parameters

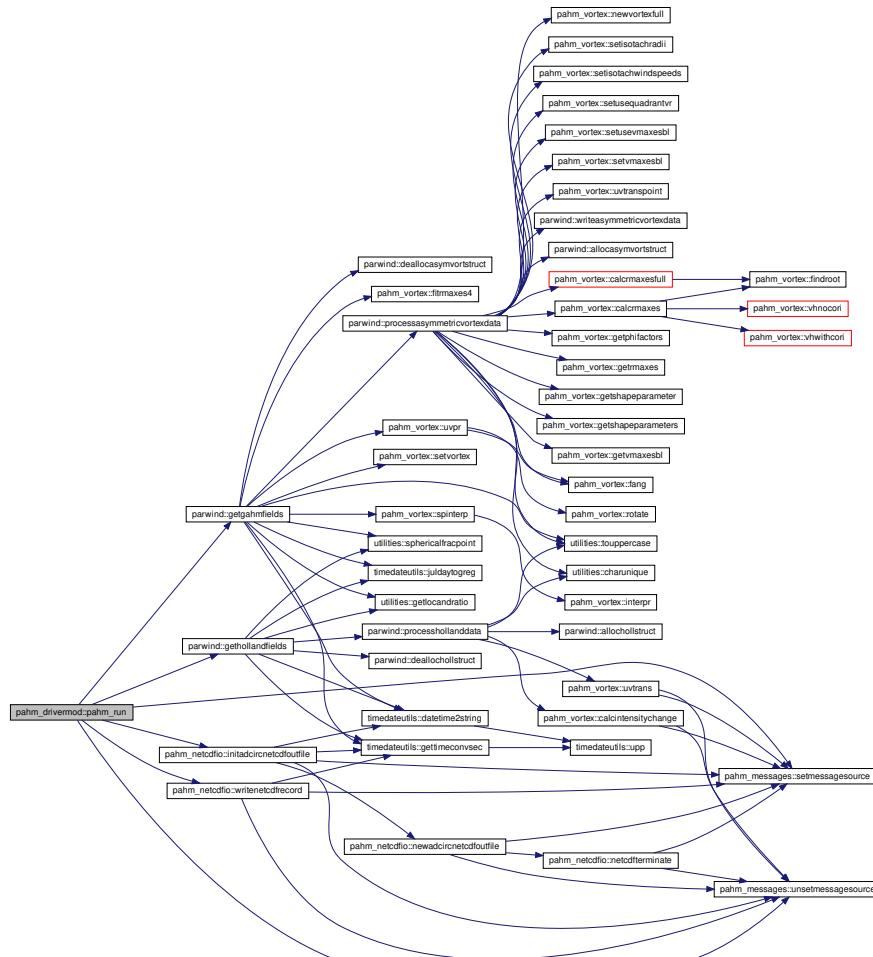
in	<i>nTimeSTP</i>	The timestep index (optional)
----	-----------------	-------------------------------

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::outfilenamespec](#), [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:



15.4.2 Variable Documentation

15.4.2.1 cnttimebegin

```
integer, save pahm_drivermod::cnttimebegin
```

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

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

15.4.2.2 cnttimeend

```
integer, save pahm_drivermod::cnttimeend
```

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

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

15.5 pahm_global Module Reference

Functions/Subroutines

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

Variables

- integer, parameter `lun_screen` = 6
- integer, parameter `lun_ctrl` = 10
- integer, parameter `lun_inp` = 14
- integer, parameter `lun_inp1` = 15
- integer, parameter `lun_log` = 35
- integer, parameter `lun_btrk` = 22
- integer, parameter `lun_btrk1` = 23
- integer, parameter `lun_out` = 25
- integer, parameter `lun_out1` = 26
- real(sz), parameter `defv_gravity` = 9.80665_SZ
- real(sz), parameter `defv_atmpress` = 1013.25_SZ
- real(sz), parameter `defv_rhoair` = 1.1478_SZ
- real(sz), parameter `defv_rhowater` = 1000.0000
- real(sz), parameter `one2ten` = 0.8928_SZ
- real(sz), parameter `ten2one` = 1.0_SZ / 0.8928_SZ
- real(sz), parameter `pi` = 3.141592653589793_SZ
- real(sz), parameter `deg2rad` = PI / 180.0_SZ
- real(sz), parameter `rad2deg` = 180.0_SZ / PI
- real(sz), parameter `basee` = 2.718281828459045_SZ
- real(sz), parameter `rearth` = 6378206.4_SZ
- real(sz), parameter `nm2m` = 1852.0_SZ
- real(sz), parameter `m2nm` = 1.0_SZ / NM2M
- real(sz), parameter `kt2ms` = NM2M / 3600.0_SZ
- real(sz), parameter `ms2kt` = 1.0_SZ / KT2MS
- real(sz), parameter `omega` = 2.0_SZ * PI / 86164.2_SZ
- real(sz), parameter `mb2pa` = 100.0_SZ
- real(sz), parameter `mb2kpa` = 0.1_SZ
- character(len=fnamelen) `logfilename` = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_model.log'
- character(fnamelen) `controlfilename` = TRIM(ADJUSTL(PROG_NAME_LOW)) // '_control.in'
- logical `meshfilenamespecified` = .FALSE.
- character(len=fnamelen) `meshfilename` = BLANK
- character(len=64) `meshfiletype` = BLANK
- character(len=64) `meshfileform` = BLANK
- logical `besttrackfilenamespecified` = .FALSE.
- integer `nbtrfiles` = IMISSV
- character(len=fnamelen), dimension(), allocatable `besttrackfilename`
- character(len=512) `title` = BLANK
- real(sz) `gravity` = DEFV_GRAVITY
- real(sz) `rhowater` = DEFV_RHOWATER
- real(sz) `rhoair` = DEFV_RHOAIR
- real(sz) `backgroundatmpress` = DEFV_ATMPRESS
- real(sz), parameter `defv_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) `outfilename` = 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`

15.5.1 Function/Subroutine Documentation

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

15.5.2 Variable Documentation

15.5.2.1 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\(\)](#).

15.5.2.2 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\(\)](#).

15.5.2.3 begdate

```
integer pahm_global::begdate = IMISSV
```

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

15.5.2.4 begdatespecified

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

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

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

15.5.2.5 begdatetime

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

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

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

15.5.2.6 begday

```
integer pahm_global::begday = 0
```

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

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

15.5.2.7 beghour

```
integer pahm_global::beghour = 0
```

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

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

15.5.2.8 begmin

```
integer pahm_global::begmin = 0
```

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

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

15.5.2.9 begmonth

```
integer pahm_global::begmonth = 0
```

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

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

15.5.2.10 begsec

```
integer pahm_global::begsec = 0
```

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

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

15.5.2.11 begsimspecified

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

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

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

15.5.2.12 begsimtime

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

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

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

15.5.2.13 begtime

```
integer pahm_global::begtime = IMISSV
```

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

15.5.2.14 begyear

```
integer pahm_global::begyear = IMISSV
```

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

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

15.5.2.15 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\(\)](#).

15.5.2.16 besttrackfilenamespecified

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

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

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

15.5.2.17 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::getprogramcmdlargs\(\)](#).

15.5.2.18 datestimes

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

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

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

15.5.2.19 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\(\)](#).

15.5.2.20 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\(\)](#).

15.5.2.21 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\(\)](#).

15.5.2.22 defv_atmpress

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

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

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

15.5.2.23 defv_gravity

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

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

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

15.5.2.24 defv_rhoair

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

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

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

15.5.2.25 defv_rhowater

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

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

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

15.5.2.26 defv_windreduction

```
real(sz), parameter pahm_global::defv_windreduction = 0.90_SZ
```

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

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

15.5.2.27 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\(\)](#).

15.5.2.28 enddate

```
integer pahm_global::enddate = IMISSV
```

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

15.5.2.29 enddatespecified

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

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

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

15.5.2.30 enddatetime

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

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

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

15.5.2.31 endday

```
integer pahm_global::endday = 0
```

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

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

15.5.2.32 endhour

```
integer pahm_global::endhour = 0
```

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

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

15.5.2.33 endmin

```
integer pahm_global::endmin = 0
```

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

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

15.5.2.34 endmonth

```
integer pahm_global::endmonth = 0
```

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

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

15.5.2.35 endsec

```
integer pahm_global::endsec = 0
```

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

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

15.5.2.36 endsimspecified

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

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

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

15.5.2.37 endsimtime

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

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

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

15.5.2.38 endtime

```
integer pahm_global::endtime = IMISSV
```

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

15.5.2.39 endyear

```
integer pahm_global::endyear = IMISSV
```

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

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

15.5.2.40 gravity

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

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

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

15.5.2.41 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::vhnocori\(\)](#), [pahm_vortex::vhwithcori\(\)](#), and [pahm_vortex::vhwithcorifull\(\)](#).

15.5.2.42 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\(\)](#).

15.5.2.43 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\(\)](#).

15.5.2.44 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\(\)](#).

15.5.2.45 lun_ctrl

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

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

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

15.5.2.46 lun_inp

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

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

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

15.5.2.47 lun_inp1

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

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

15.5.2.48 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::logmess](#) and [pahm_messages::openlogfile\(\)](#).

15.5.2.49 lun_out

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

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

15.5.2.50 lun_out1

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

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

15.5.2.51 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\(\)](#).

15.5.2.52 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\(\)](#).

15.5.2.53 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\(\)](#).

15.5.2.54 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\(\)](#).

15.5.2.55 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\(\)](#).

15.5.2.56 mdendsimtime

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

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

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

15.5.2.57 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\(\)](#).

15.5.2.58 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\(\)](#).

15.5.2.59 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\(\)](#).

15.5.2.60 meshfilenamespecified

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

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

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

15.5.2.61 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\(\)](#).

15.5.2.62 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\(\)](#).

15.5.2.63 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\(\)](#).

15.5.2.64 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\(\)](#).

15.5.2.65 ncdeflate

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

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

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

15.5.2.66 ncdlevel

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

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

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

15.5.2.67 ncshuffle

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

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

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

15.5.2.68 ncvarnam_pres

```
character(len=20) pahm_global::ncvarnam_pres = DEF_NCNAME_PRES
```

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

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

15.5.2.69 ncvarnam_wndx

```
character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAME_WNDX
```

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

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

15.5.2.70 ncvarnam_wndy

```
character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAME_WNDY
```

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

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

15.5.2.71 nm2m

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

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

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

15.5.2.72 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\(\)](#).

15.5.2.73 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\(\)](#).

15.5.2.74 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\(\)](#).

15.5.2.75 outdt

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

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

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

15.5.2.76 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\(\)](#).

15.5.2.77 outfilenamespecified

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

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

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

15.5.2.78 pi

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

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

15.5.2.79 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\(\)](#).

15.5.2.80 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_1d\(\)](#), [utilities::sphericaldistance::sphericaldistance_2d\(\)](#), [utilities::sphericaldistance::sphericaldistance_scalar\(\)](#), [utilities::sphericaldistanceharv\(\)](#), [utilities::sphericalfracpoint\(\)](#), and [pahm_vortex::uvp\(\)](#).

15.5.2.81 refdate

```
integer pahm_global::refdate = IMISSV
```

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

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

15.5.2.82 refdatespecified

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

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

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

15.5.2.83 refdatetime

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

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

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

15.5.2.84 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\(\)](#).

15.5.2.85 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\(\)](#).

15.5.2.86 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\(\)](#).

15.5.2.87 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\(\)](#).

15.5.2.88 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\(\)](#).

15.5.2.89 reftime

```
integer pahm_global::reftime = IMISSV
```

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

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

15.5.2.90 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\(\)](#).

15.5.2.91 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\(\)](#).

15.5.2.92 rhowater

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

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

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

15.5.2.93 tenZone

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

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

15.5.2.94 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::initadcircnetcdfoutfile\(\)](#), [pahm_vortex::uvtrans\(\)](#), and [pahm_netcdfio::writenetcdffrecord\(\)](#).

15.5.2.95 title

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

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

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

15.5.2.96 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::writenetcdffrecord\(\)](#).

15.5.2.97 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::processasymmetricvortexdata\(\)](#), [pahm_vortex::uvp\(\)](#), and [pahm_vortex::uvpr\(\)](#).

15.5.2.98 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::writenetcdffrecord\(\)](#).

15.5.2.99 writeparams

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

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

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

15.5.2.100 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\(\)](#).

15.5.2.101 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\(\)](#).

15.6 pahm_mesh Module Reference

Functions/Subroutines

- subroutine [readmesh\(\)](#)
Reads an input mesh file for the specified supported model type.
- subroutine [readmeshasciifort14\(\)](#)
Reads the ADCIRC fort.14 mesh file.
- subroutine [allocatenodalandelementalarrays\(\)](#)
Allocates memory to mesh arrays.

Variables

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

15.6.1 Function/Subroutine Documentation

15.6.1.1 `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:



15.6.1.2 readmesh()

```
subroutine pahm_mesh::readmesh
```

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

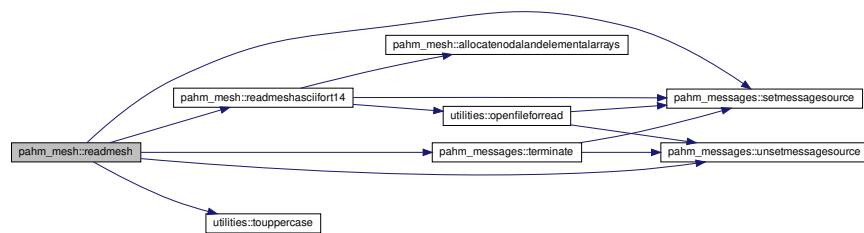
Read the mesh file for the specified model type (meshFileType) and in ASCII or NetCDF format (if applicable).

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

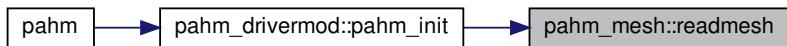
References [pahm_messages::error](#), [pahm_global::meshfileform](#), [pahm_global::meshfilename](#), [pahm_global::meshfilenames](#), [pahm_global::meshfiletype](#), [readmeshasciifort14\(\)](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource](#), [pahm_messages::terminate\(\)](#),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



15.6.1.3 readmeshasciifort14()

```
subroutine pahm_mesh::readmeshasciifort14
```

Reads the ADCIRC fort.14 mesh file.

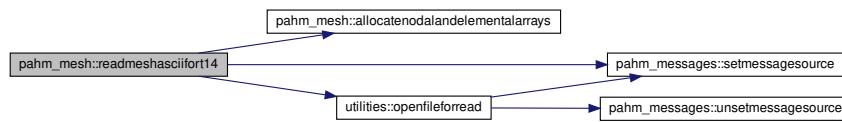
Reads the ADCIRC fort.14 mesh file and sets all mesh variables and arrays.

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

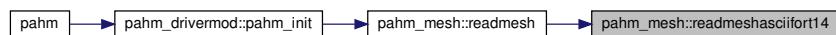
References [agrid](#), [allocatenodalandelementalarrays\(\)](#), [pahm_messages::info](#), [ismeshok](#), [pahm_global::lun_inp](#), [maxfacenodes](#), [pahm_global::meshfilename](#), [ne](#), [np](#), [utilities::openfileforread\(\)](#), [pahm_messages::setmessagesource\(\)](#), [sfea](#), [sfea0](#),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

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.6.2 Variable Documentation

15.6.2.1 agrid

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

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

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

15.6.2.2 dp

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

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

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

15.6.2.3 ics

```
integer pahm_mesh::ics
```

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

15.6.2.4 ismeshok

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

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

Referenced by [parwind::getgahmfIELDS\(\)](#), [parwind::gethollandFIELDS\(\)](#), and [readmeshasciifort14\(\)](#).

15.6.2.5 maxfacenodes

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

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

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

15.6.2.6 ne

```
integer pahm_mesh::ne = IMISSV
```

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

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

15.6.2.7 nfn

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

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

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

15.6.2.8 nm

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

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

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

15.6.2.9 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\(\)](#).

15.6.2.10 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\(\)](#).

15.6.2.11 sfea0

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

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

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

15.6.2.12 slam

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

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

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

15.6.2.13 slam0

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

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

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

15.6.2.14 xcslam

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

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

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

15.6.2.15 ycsfea

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

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

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

15.7 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.

15.7.1 Function/Subroutine Documentation

15.7.1.1 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

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

Definition at line 318 of file [messages.F90](#).

15.7.1.2 allmessage_2()

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

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

15.7.1.3 closelogfile()

```
subroutine pahm_messages::closelogfile
```

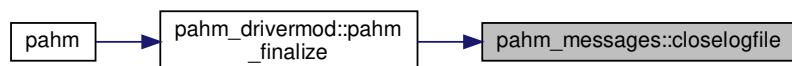
Closes an opened log file.

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:



15.7.1.4 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](#).

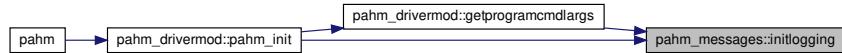
References [loginitcalled](#), [loglevelname](#)s, [openlogfile\(\)](#), and [sourcenumber](#).

Referenced by [pahm_drivermod::getprogramcmdargs\(\)](#), and [pahm_drivermod::pahm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.1.5 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](#).

15.7.1.6 logmessage_2()

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

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

15.7.1.7 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:



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

Here is the call graph for this function:



Here is the caller graph for this function:



15.7.1.9 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::getprogramcmdargs\(\)](#), and [programhelp\(\)](#).

Here is the caller graph for this function:



15.7.1.10 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	message	The message to display
----	---------	------------------------

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

15.7.1.11 screenmessage_2()

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

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

15.7.1.12 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

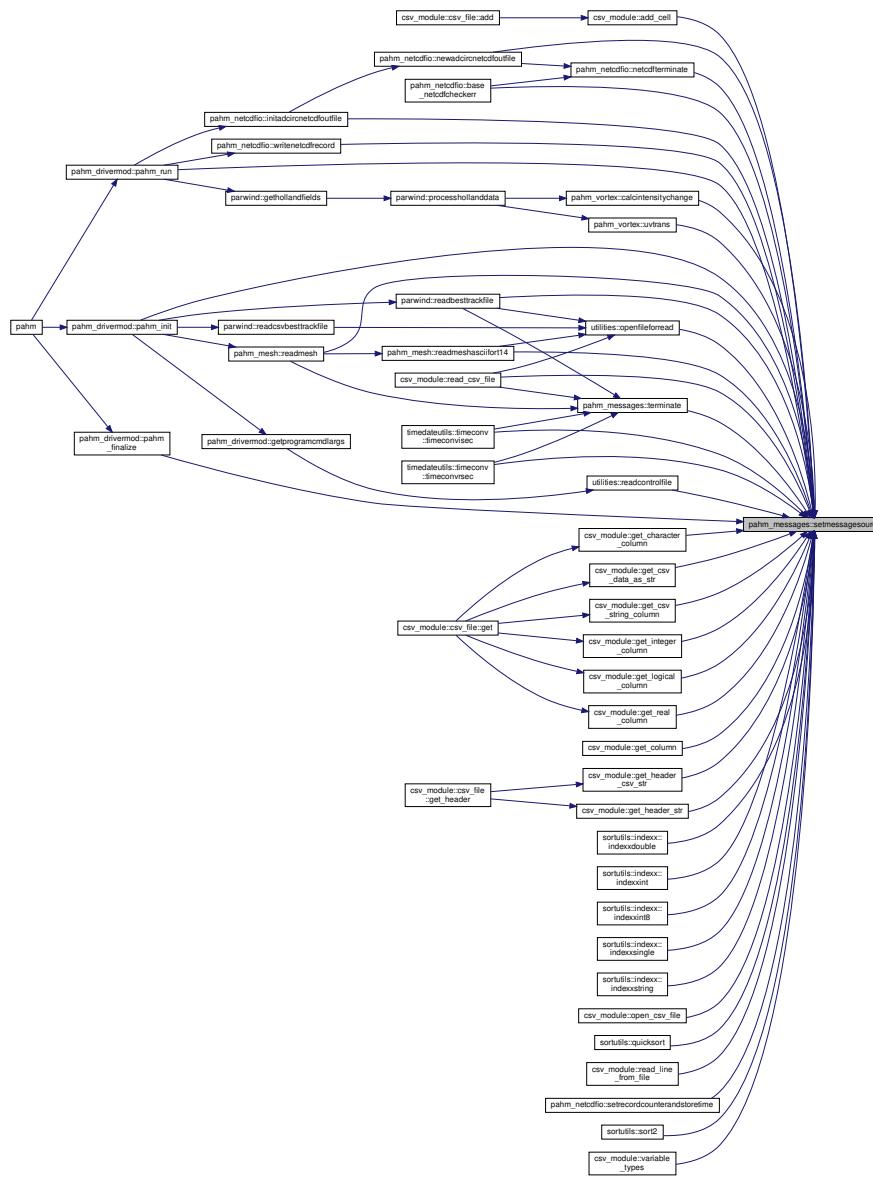
in	source	The name of the calling procedure
----	--------	-----------------------------------

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

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

Referenced by [csv_module::add_cell\(\)](#), [pahm_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::initadcircnetcdfoutfile\(\)](#), [pahm_netcdfio::netcdfterminate\(\)](#), [pahm_netcdfio::newadcircnetcdfoutfile\(\)](#), [csv_module::open_csv_file\(\)](#), [utilities::openfileforread\(\)](#), [pahm_drivermod::pahm_fi](#), [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_netcdfio::setrecordcounterandstoretime\(\)](#), [sortutils::sort2\(\)](#), [terminate\(\)](#), [timedateutils::timeconv::timeconvise\(\)](#), [timedateutils::timeconv::timeconvrsec\(\)](#), [pahm_vortex::uvtrans\(\)](#), [csv_module::variable_types\(\)](#), and [pahm_netcdfio::writenetcdfrecord\(\)](#).

Here is the caller graph for this function:



15.7.1.13 terminate()

```
subroutine pahm_messages::terminate
```

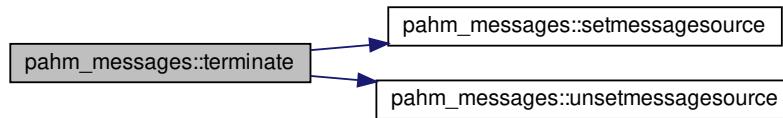
Terminates the calling program when a fatal error is encountered.

Definition at line 464 of file [messages.F90](#).

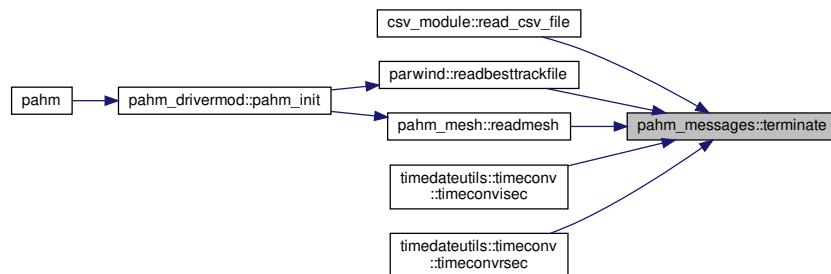
References [error](#), [setmessagesource\(\)](#), and [unsetmessagesource\(\)](#).

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:



15.7.1.14 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.

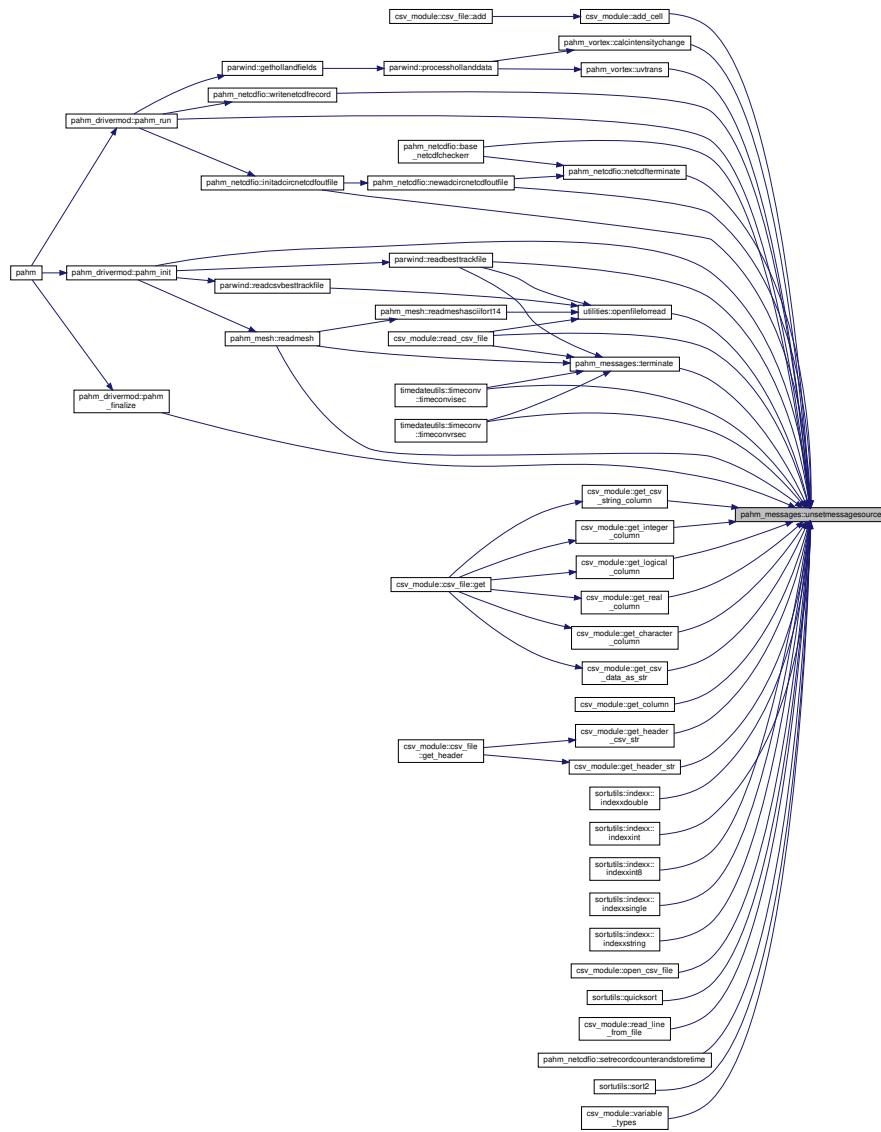
Definition at line 388 of file [messages.F90](#).

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_ncdfio::initadcircnetcdfoutfile()`, `pahm_ncdfio::netcdfterminate()`, `pahm_ncdfio::newadcircnetcdfoutfile()`, `csv_module::open_csv_file()`, `utilities::openfileforread()`, `pahm_drivermod::pahm_fi`, `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::setrecordcounter`, `sortutils::sort2()`, `terminate()`, `timedateutils::timeconv::timeconvise()`, `timedateutils::timeconv::timeconvrsec()`, `pahm_vortex::uvtrans()`, `csv_module::variable_types()`, and `pahm_ncdfio::writenetcdfrecord()`.

Here is the caller graph for this function:



15.7.2 Variable Documentation

15.7.2.1 debug

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

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

15.7.2.2 echo

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

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

15.7.2.3 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::indexxint\(\)](#), [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\(\)](#).

15.7.2.4 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\(\)](#).

15.7.2.5 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\(\)](#).

15.7.2.6 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\(\)](#).

15.7.2.7 loglevelname

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

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

Referenced by [initlogging\(\)](#), [pahm_messages::logmessage::logmessage_2\(\)](#), and [pahm_messages::screenmessage::screenmessage_1\(\)](#).

15.7.2.8 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\(\)](#).

15.7.2.9 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::screenmes](#)

15.7.2.10 scratchformat

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

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

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

15.7.2.11 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_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::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\(\)](#).

15.7.2.12 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\(\)](#).

15.7.2.13 warning

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

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

15.8 pahm_ncdfio 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](#) (nclID, adcircOutFile)
Creates a new NetCDF data file and puts it in define mode.
- subroutine [base_ncdfcheckerr](#) (ierr, file, line)
Checks the return value from netCDF calls.
- subroutine [netcdffterminate](#) ()
Terminates the program on NetCDF error.
- subroutine [writenetcdffrecord](#) (adcircOutFile, timeLoc)
Writes data to the NetCDF file.
- subroutine [setrecordcounterandstoretime](#) (nclID, 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 [crdtme](#)

- 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`

15.8.1 Function/Subroutine Documentation

15.8.1.1 `base_ncdfcheckerr()`

```
subroutine pahm_netcdfio::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.

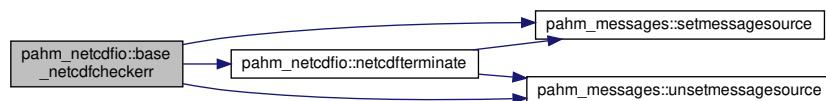
Parameters

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:



15.8.1.2 initadcircnetcdfoutfile()

```
subroutine pahm_netcdfio::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

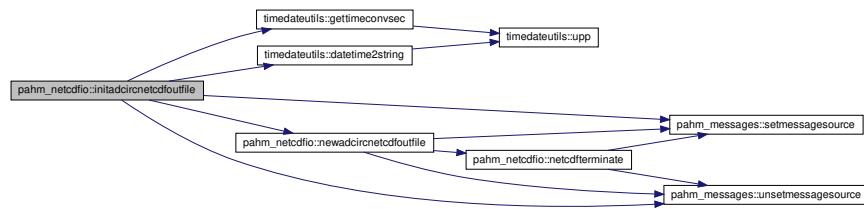
in,out	<i>adcircOutFile</i>	The name of the file to be initialized. The file is first created by calling NewAdcircNetCDFOutFile.
--------	----------------------	--

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

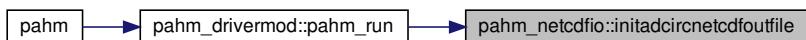
References [pahm_mesh::agrid](#), [crdlats](#), [crdlons](#), [crdtime](#), [crdxcs](#), [crdycs](#), [datatmpres](#), [datelements](#), [timedateutils::datetime2string\(\)](#), [datwindx](#), [datwindy](#), [elemdimid](#), [timedateutils::gettimeconvsec\(\)](#), [pahm_sizes::imissv](#), [meshdimid](#), [meshvarid](#), [myfile](#), [nc4form](#), [pahm_global::ncdeflate](#), [pahm_global::ncdlevel](#), [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::wvely](#), [pahm_mesh::xcslam](#), and [pahm_mesh::ycsfea](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.8.1.3 netcdfterminate()

```
subroutine pahm_netcdfio::netcdfterminate
```

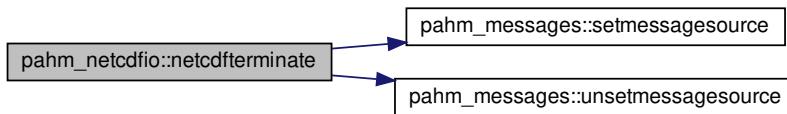
Terminates the program on NetCDF error.

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

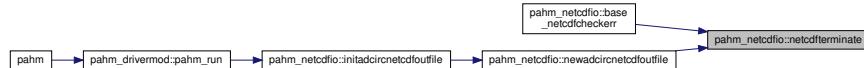
References [pahm_messages::info](#), [pahm_messages::setmessagesource\(\)](#), and [pahm_messages::unsetmessagesource\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.8.1.4 newadcircnetcdfoutfile()

```
subroutine pahm_netcdfio::newadcircnetcdfoutfile (
    integer, intent(out) ncID,
    character(len=*), intent(inout) adcircOutFile )
```

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

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

Parameters

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



15.8.1.5 setrecordcounterandstoretime()

```

subroutine pahm_netcdfio::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.

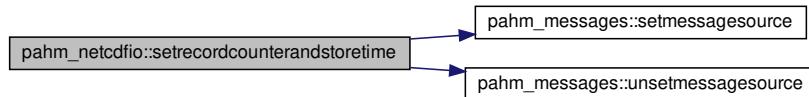
Parameters

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:



15.8.1.6 writenetcdfrecord()

```

subroutine pahm_netcdfio::writenetcdfrecord (
    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

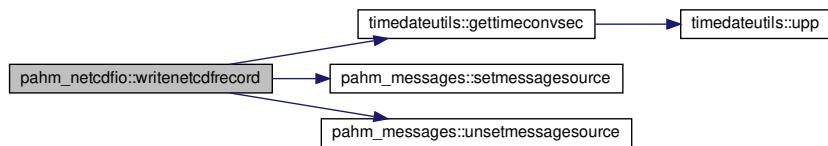
in	<i>adcircOutFile</i>	The name of the NetCDF file
in	<i>timeLoc</i>	The time record to write

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

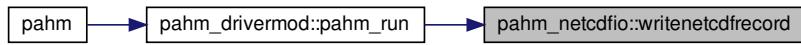
References [crdtime](#), [datatmpres](#), [datwindx](#), [datwindy](#), [timedateutils::gettimeconvsec\(\)](#), [nodedimid](#), [pahm_messages::setmessagesource\(\)](#), [pahm_global::times](#), [pahm_global::unitime](#), [pahm_messages::unsetmessagesource](#), [pahm_global::wpress](#), [pahm_global::wvelx](#), and [pahm_global::wvly](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.8.2 Variable Documentation

15.8.2.1 crdlats

```
type(adcirccoorddata_t), save, private pahm_netcdfio::crlats [private]
```

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

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

15.8.2.2 crdlons

```
type(adcirccoorddata_t), save, private pahm_netcdfio::cndlons [private]
```

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

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

15.8.2.3 crdtime

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

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

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

15.8.2.4 crdxcs

```
type(adcirccoorddata\_t), save, private pahm_ncdfio::crdxcs [private]
```

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

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

15.8.2.5 crdycs

```
type(adcirccoorddata\_t), save, private pahm_ncdfio::crdycs [private]
```

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

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

15.8.2.6 datatmpres

```
type(adcircvardata\_t), save, private pahm_ncdfio::datatmpres [private]
```

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

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

15.8.2.7 datelements

```
type(adcircvardata\_t), save, private pahm_ncdfio::datelements [private]
```

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

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

15.8.2.8 datwindx

```
type(adcircvardata\_t), save, private pahm_ncdfio::datwindx [private]
```

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

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

15.8.2.9 datwindy

```
type(adcircvardata\_t), save, private pahm_ncdfio::datwindy [private]
```

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

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

15.8.2.10 elemdimid

```
integer, private pahm_ncdfio::elemdimid [private]
```

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

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

15.8.2.11 meshdimid

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

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

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

15.8.2.12 meshvarid

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

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

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

15.8.2.13 myfile

```
type(filedata\_t), save pahm_ncdfio::myfile [private]
```

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

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

15.8.2.14 mytime

```
type(timedata\_t), save pahm_ncdfio::mytime
```

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

15.8.2.15 nc3form

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

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

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

15.8.2.16 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\(\)](#).

15.8.2.17 ncformat

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

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

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

15.8.2.18 nodedimid

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

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

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

15.8.2.19 projvarid

```
integer, private pahm_netcdfio::projvarid [private]
```

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

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

15.8.2.20 vertdimid

```
integer, private pahm_netcdfio::vertdimid [private]
```

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

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

15.9 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 [fixnearwhole singlereal](#) (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

15.9.1 Function/Subroutine Documentation

15.9.1.1 comparedoublereals()

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

Compares two double precision numbers.

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

Parameters

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

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

Returns

myValOut

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

Note

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

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

15.9.1.3 fixnearwholedoublereal()

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

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

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

Parameters

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](#).

15.9.1.4 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](#).

15.9.2 Variable Documentation

15.9.2.1 blank

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

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

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

15.9.2.2 fnameLEN

```
integer, parameter pahm_sizes::fnameLEN = 1024
```

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

15.9.2.3 hp

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

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

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

15.9.2.4 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\(\)](#).

15.9.2.5 int1

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

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

15.9.2.6 int16

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

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

15.9.2.7 int2

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

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

15.9.2.8 int4

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

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

15.9.2.9 int8

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

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

15.9.2.10 ip

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

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

15.9.2.11 llong

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

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

15.9.2.12 long

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

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

15.9.2.13 nbytes

```
integer, parameter pahm_sizes::nbytes = 8
```

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

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

15.9.2.15 sp

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

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

15.9.2.16 sz

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

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

15.9.2.17 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\(\)](#).

15.10 pahm_vortex Module Reference

Functions/Subroutines

- subroutine, public [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 [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 interpr (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 vhwithcorifull (testRMax)`
External function $f(x) = 0$ for which a root is sought using Brent's root-finding method.
- `real(sz) function vhwithcori (testRMax)`
External function $f(x) = 0$ for which a root is sought using Brent's root-finding method.
- `real(sz) function vhnocori (testRMax)`
- `real(sz) function findroot (func, x1, x2, dx, a, b)`
Use brute-force marching to find a root the interval [x1,x2].

Variables

- integer, parameter `nquads = 4`
- integer, parameter `npoints = NQUADS + 2`
- `real(sz), dimension(npoints) rmaxes`
- `real(sz), dimension(npoints, 4), 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) vmb1`
- `real(sz), dimension(npoints, 4), public vmb14`
- `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`

15.10.1 Function/Subroutine Documentation

15.10.1.1 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

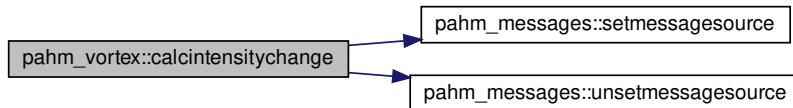
in	<i>var</i>	The input variable (vector)
in	<i>times</i>	Time values (vector) at the center locations
out	<i>calcInt</i>	The calculated intensity change (df/dt)
out	<i>status</i>	Error status (0 means no error)
in	<i>order</i>	The accuracy order required for the calculations (1, 2) order <= 1: first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



15.10.1.2 calcrmaxes()

subroutine, public pahm_vortex::calcrmaxes

Calculates the radius of maximum winds for all storm quadrants.

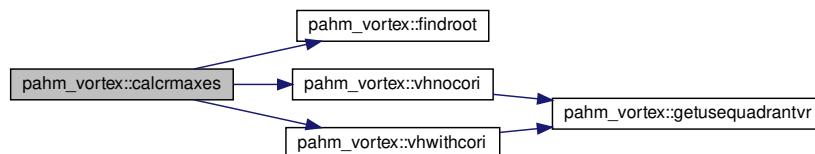
Calculates rMax, the radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity

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

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.10.1.3 calcrmaxesfull()

```
subroutine, public pahm_vortex::calcrmaxesfull
```

Calculates the radius of maximum winds for all storm quadrants.

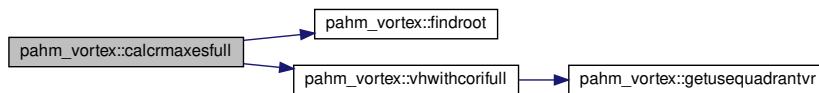
Solves the full gradient wind equation without the assumption of cyclostrophic balance. Calculates rMax, the radius of maximum winds (nm) in all quadrants, plus 2 extra values to tie down circular periodicity.

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 [vmbl](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.10.1.4 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

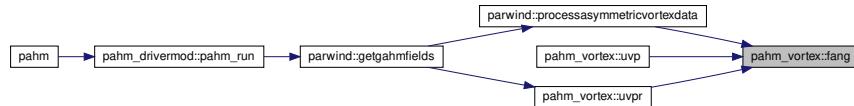
Returns

`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:

**15.10.1.5 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:



15.10.1.6 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](#).

15.10.1.7 fitrmaxes4()

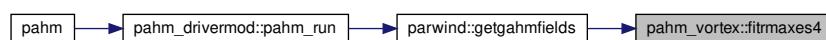
```
subroutine, public pahm_vortex::fitrmaxes4
```

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

References [bs4](#), [phis4](#), [quadflag4](#), [quadir4](#), [rmaxes4](#), and [vmb14](#).

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

Here is the caller graph for this function:



15.10.1.8 getlatestangle()

```
real(sz) function, public pahm_vortex::getlatestangle
```

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

References [latestangle](#).

15.10.1.9 getlatestrmax()

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

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

References [latestrmax](#).

15.10.1.10 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:



15.10.1.11 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:



15.10.1.12 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:



15.10.1.13 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:



15.10.1.14 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:



15.10.1.15 getvmaxesbl()

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

Definition at line 1020 of file [vortex.F90](#).

References [vmbl](#).

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

Here is the caller graph for this function:



15.10.1.16 `interp()`

```
real(sz) function, public pahm_vortex::interp (
    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 [spinterp\(\)](#).

Here is the caller graph for this function:



15.10.1.17 `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](#).

15.10.1.18 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 [vmbi](#).

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

Here is the caller graph for this function:



15.10.1.19 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:

**15.10.1.20 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
Generated by Doxygen		
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:



15.10.1.21 setisotachradii()

```
subroutine, public pahm_vortex::setisotachradii (
    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:



15.10.1.22 setisotachwindspeed()

```
subroutine, public pahm_vortex::setisotachwindspeed (
    real(sz), intent(in) sp )
```

Definition at line 1154 of file [vortex.F90](#).

References [vr](#).

15.10.1.23 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:



15.10.1.24 setrmaxes()

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

Definition at line 697 of file [vortex.F90](#).

References [rmaxes](#).

15.10.1.25 setshapeparameter()

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

Definition at line 1052 of file [vortex.F90](#).

References [b](#).

15.10.1.26 setusequadrantvr()

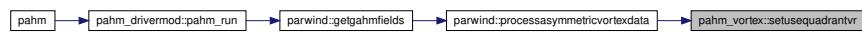
```
subroutine, public pahm_vortex::setusequadrantvr (
    logical, intent(in) u )
```

Definition at line 1169 of file [vortex.F90](#).

References [usequadrantvr](#).

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

Here is the caller graph for this function:



15.10.1.27 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:



15.10.1.28 setvmaxesbl()

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

Definition at line 1001 of file [vortex.F90](#).

References [vmbl](#).

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

Here is the caller graph for this function:



15.10.1.29 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:



15.10.1.30 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](#), [interp\(\)](#), [rmaxes4](#), and [vmb14](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.10.1.31 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.

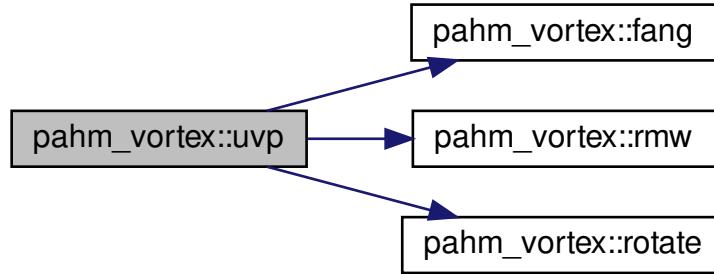
Parameters

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:



15.10.1.32 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

Parameters

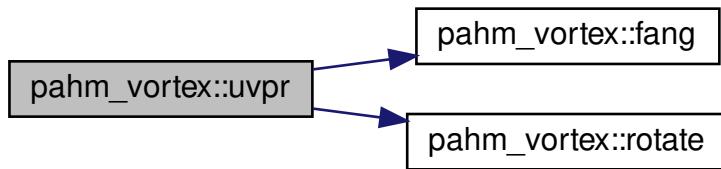
in	<i>uTrans</i>	x component of translational velocity (knts)
in	<i>vTrans</i>	y component of translational velocity (knts)
in	<i>geof</i>	Factor to calculate wind parameters from the asymmetric hurricane vortex (<i>geof</i> = 1)
out	<i>u</i>	x component of wind velocity at nodal point (m/s)
out	<i>v</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:



15.10.1.33 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

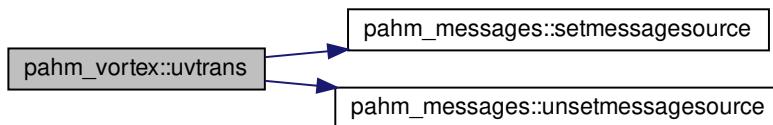
in	<i>lat</i>	Latitude values (vector) of the center (degrees north)
in	<i>lon</i>	Longitude values (vector) of the center (degrees east)
in	<i>times</i>	Time values (vector) at the center locations (seconds)
out	<i>u</i>	x component of the translational velocities (m/s)
out	<i>v</i>	y component of the 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::setmess](#), [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:



15.10.1.34 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:



15.10.1.35 vhnocori()

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

Definition at line 1954 of file [vortex.F90](#).

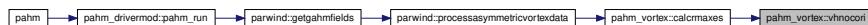
References [b](#), [getusequadrantvr\(\)](#), [pahm_global::kt2ms](#), [pahm_global::ms2kt](#), [quad](#), [radius](#), [vmax](#), [vr](#), and [vrquadrant](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.10.1.36 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:



15.10.1.37 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

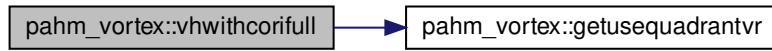
myValOut The function's result

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:



15.10.2 Variable Documentation

15.10.2.1 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\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

15.10.2.2 bs

```
real(sz), dimension(npoin) pahm_vortex::bs [private]
```

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

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [getshapeparameters\(\)](#), [newvortex\(\)](#), and [newvortexfull\(\)](#).

15.10.2.3 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\(\)](#).

15.10.2.4 clat

```
real(sz) pahm_vortex::clat [private]
```

Definition at line 50 of file [vortex.F90](#).

Referenced by [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

15.10.2.5 clon

```
real(sz) pahm_vortex::clon [private]
```

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

Referenced by [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), and [uvp\(\)](#).

15.10.2.6 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\(\)](#).

15.10.2.7 latestangle

```
real(sz) pahm_vortex::latestangle [private]
```

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

Referenced by [getlatestangle\(\)](#), and [uvp\(\)](#).

15.10.2.8 latestrmax

```
real(sz) pahm_vortex::latestrmax [private]
```

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

Referenced by [getlatestrmax\(\)](#), and [uvp\(\)](#).

15.10.2.9 npoints

```
integer, parameter pahm_vortex::npoints = NQUADS + 2 [private]
```

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

15.10.2.10 nquads

```
integer, parameter pahm_vortex::nquads = 4 [private]
```

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

Referenced by [calcrmaxes\(\)](#), and [calcrmaxesfull\(\)](#).

15.10.2.11 pc

```
real(sz) pahm_vortex::pc [private]
```

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

Referenced by [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

15.10.2.12 phi

```
real(sz) pahm_vortex::phi [private]
```

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

Referenced by [calcrmaxesfull\(\)](#), [newvortexfull\(\)](#), [uvpr\(\)](#), and [vhwithcorifull\(\)](#).

15.10.2.13 phis

```
real(sz), dimension(npoints) pahm_vortex::phis [private]
```

Definition at line 58 of file [vortex.F90](#).

Referenced by [calcrmaxesfull\(\)](#), [getphifactors\(\)](#), and [newvortexfull\(\)](#).

15.10.2.14 phis4

```
real(sz), dimension(npoints, 4) pahm_vortex::phis4 [private]
```

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

Referenced by [firrmaxes4\(\)](#).

15.10.2.15 pn

```
real(sz) pahm_vortex::pn [private]
```

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

Referenced by [calcrmaxesfull\(\)](#), [newvortex\(\)](#), [newvortexfull\(\)](#), [setvortex\(\)](#), [uvp\(\)](#), and [uvpr\(\)](#).

15.10.2.16 quad

```
integer pahm_vortex::quad [private]
```

Definition at line 70 of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [vhncori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

15.10.2.17 quadflag4

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

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

Referenced by [firrmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [interpr\(\)](#).

15.10.2.18 quadir4

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

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

Referenced by [firrmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [interp\(\)](#).

15.10.2.19 radius

```
real(sz), dimension(nquads) pahm_vortex::radius [private]
```

Definition at line 68 of file [vortex.F90](#).

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [setisotachradii\(\)](#), [vhncori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

15.10.2.20 rmaxes

```
real(sz), dimension(npoints) pahm_vortex::rmaxes [private]
```

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

Referenced by [calcrmaxes\(\)](#), [calcrmaxesfull\(\)](#), [firrmaxes\(\)](#), [getrmaxes\(\)](#), [rmw\(\)](#), and [setrmaxes\(\)](#).

15.10.2.21 rmaxes4

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

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

Referenced by [firrmaxes4\(\)](#), [parwind::getgahmfields\(\)](#), and [sinterp\(\)](#).

15.10.2.22 usequadrantvr

```
logical pahm_vortex::usequadrantvr [private]
```

Definition at line 74 of file [vortex.F90](#).

Referenced by [getusequadrantvr\(\)](#), and [setusequadrantvr\(\)](#).

15.10.2.23 usevmaxesbl

```
logical pahm_vortex::usevmaxesbl [private]
```

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

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

15.10.2.24 vmax

```
real(sz) pahm_vortex::vmax [private]
```

Definition at line 52 of file [vortex.F90](#).

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

15.10.2.25 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\(\)](#).

15.10.2.26 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\(\)](#).

15.10.2.27 vr

```
real(sz) pahm_vortex::vr [private]
```

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

Referenced by [setisotachwindspeed\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

15.10.2.28 vrquadrant

```
real(sz), dimension(nquads) pahm_vortex::vrquadrant [private]
```

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

Referenced by [setisotachwindspeeds\(\)](#), [vhnocori\(\)](#), [vhwithcori\(\)](#), and [vhwithcorifull\(\)](#).

15.11 parwind Module Reference

Data Types

- type [asymetricvortexdata_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 `deallocbtrstruct` (str)
 - Subroutine to allocate memory for a best track structure.*
- subroutine `allochollstruct` (str, nRec)
 - Subroutine to deallocate the memory allocated for a best track structure.*
- subroutine `deallochollstruct` (str)
 - Subroutine to allocate memory for a holland structure.*
- subroutine `allocasymvortstruct` (str, nRec)
 - Subroutine to deallocate memory of an allocated holland structure.*
- subroutine `deallocasymvortstruct` (str)
 - 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`

15.11.1 Function/Subroutine Documentation

15.11.1.1 `allocasymvortstruct()`

```
subroutine parwind::allocasymvortstruct (
    type(asymmetricvortexdata_t), intent(inout) str,
    integer, intent(in) nRec )
```

Subroutine to allocate memory for an asymmetric vortex structure.

Parameters

<code>in, out</code>	<code>str</code>	The asymmetric vortex structure of type AsymmetricVortexData_T
<code>in</code>	<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:



15.11.1.2 allocbtrstruct()

```

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

Subroutine to allocate memory for a best track structure.

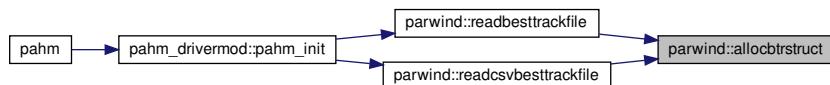
Parameters

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:



15.11.1.3 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:

**15.11.1.4 deallocasymvortstruct()**

```
subroutine parwind::dallocasymvortstruct (
    type(asymmetricvortexdata_t), intent(inout) str )
```

Subroutine to deallocate memory of an allocated asymmetric vortex structure.

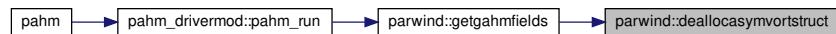
Parameters

in,out	<i>str</i>	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:



15.11.1.5 deallocbtrstruct()

```
subroutine parwind::dallocbtrstruct (
    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](#).

15.11.1.6 deallochollstruct()

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

Subroutine to deallocate memory of an allocated holland structure.

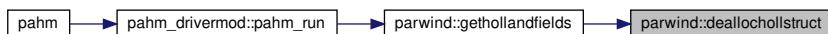
Parameters

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:

**15.11.1.7 getgahmfields()**

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

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

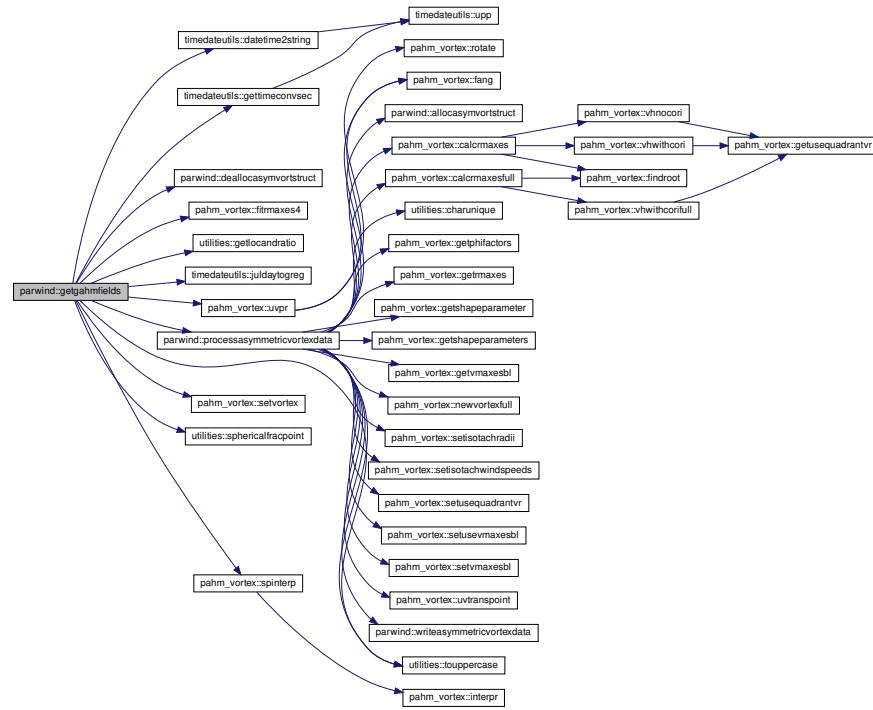
in	<i>timeIDX</i>	The time location to generate the fields for
----	----------------	--

Definition at line 2356 of file [parwind.F90](#).

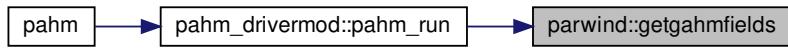
References [asyvortstru](#), [pahm_global::backgroundatmpress](#), [pahm_global::baseee](#), [pahm_global::besttrackfilename](#), [pahm_vortex::bs4](#), [pahm_global::datestimes](#), [timedateutils::datetime2string\(\)](#), [deallocasymvortstruct\(\)](#), [pahm_global::deg2rad](#), [pahm_vortex::firmaxes4\(\)](#), [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::vmlb4](#), [pahm_global::wpress](#), [pahm_global::wvelx](#), and [pahm_global::wvly](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.11.1.8 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

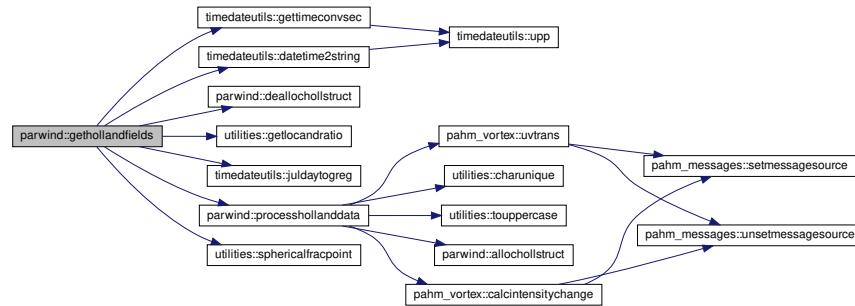
in	<i>timeIDX</i>	The time location to generate the fields for
----	----------------	--

Definition at line 1954 of file [parwind.F90](#).

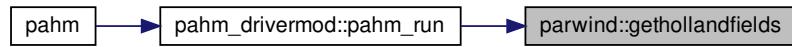
References [pahm_global::backgroundatmpress](#), [pahm_global::baseee](#), [pahm_global::besttrackfilename](#), [pahm_global::datestimes](#), [timedateutils::datetime2string\(\)](#), [deallochollstruct\(\)](#), [pahm_global::deg2rad](#), [utilities::getlocandratio\(\)](#), [timedateutils::gettmeconsec\(\)](#), [holstru](#), [pahm_mesh::ismeshok](#), [timedateutils::juldaytoreg\(\)](#), [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:



15.11.1.9 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

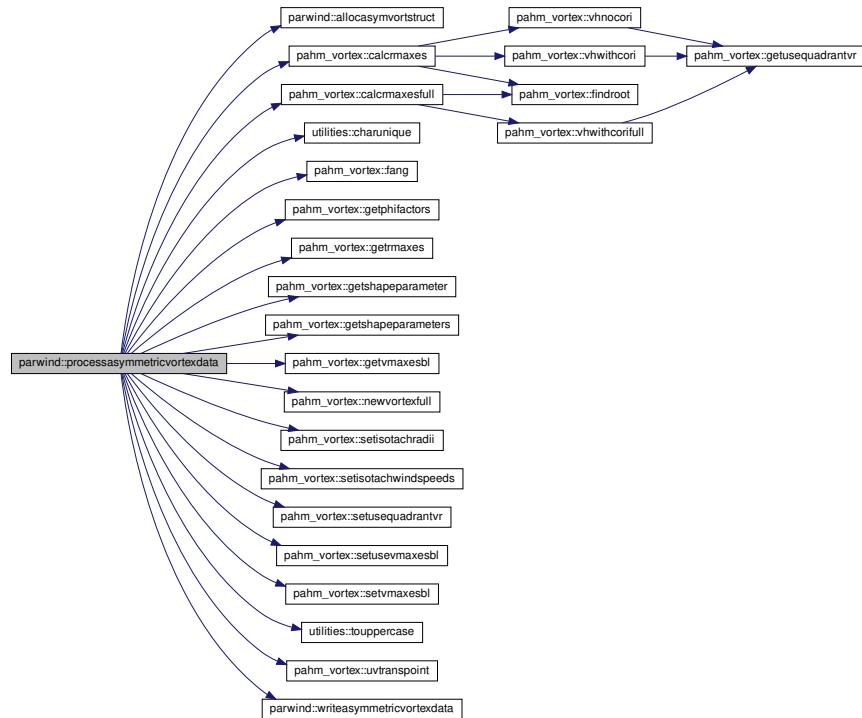
in	<i>idTrFile</i>	The ID of the input track file (1, 2, ...)
out	<i>strOut</i>	The AsymmetricVortexData_T structure that stores all model generated data (output)
Generated by Doxygen out	<i>status</i>	Error status, 0 = no error (output)

Definition at line 1118 of file [parwind.F90](#).

References `allocasymvortstruct()`, `approach`, `pahm_global::backgroundatmpress`, `besttrackdata`, `pahm_global::besttrackfile`, `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:



15.11.1.10 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

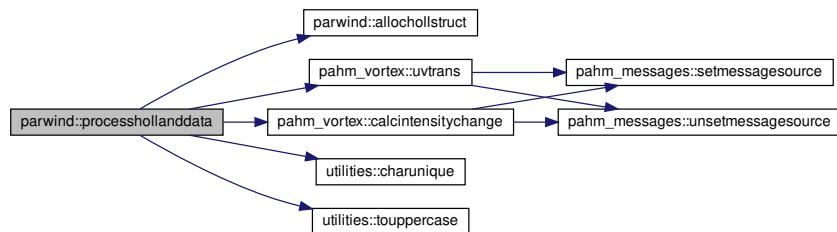
in	<i>idTrFile</i>	The ID of the input track file (1, 2, ...)
out	<i>strOut</i>	The HollandData_T structure that stores all Holland model generated data (output)
out	<i>status</i>	Error status, 0 = no error (output)

Definition at line 894 of file [parwind.F90](#).

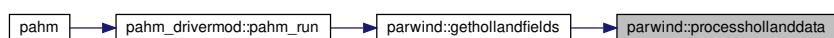
References [allochollstruct\(\)](#), [besttrackdata](#), [pahm_vortex::calcintensitychange\(\)](#), [utilities::charunique\(\)](#), [pahm_global::kt2ms](#), [pahm_global::nbtrfiles](#), [pahm_global::nm2m](#), [utilities::touppercase\(\)](#), and [pahm_vortex::uvtrans\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.11.1.11 `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:

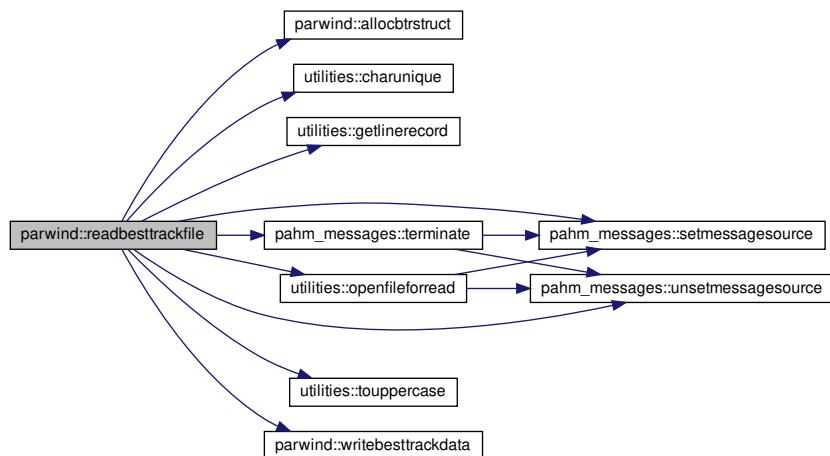
- a-deck: guidance information
- b-deck: best track information
- Skips lines that are time repeats.
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 266 of file [parwind.F90](#).

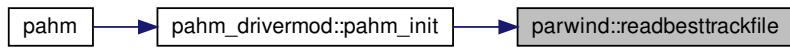
References `allocbtrstruct()`, `besttrackdata`, `pahm_global::besttrackfilename`, `utilities::charunique()`, `pahm_messages::error`, `utilities::getlinerecord()`, `pahm_messages::info`, `pahm_global::lun_btrk`, `pahm_global::lun_btrk1`, `pahm_global::nbtrfiles`, `utilities::openfileforread()`, `pahm_messages::scratchmessage`,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

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.11.1.12 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:

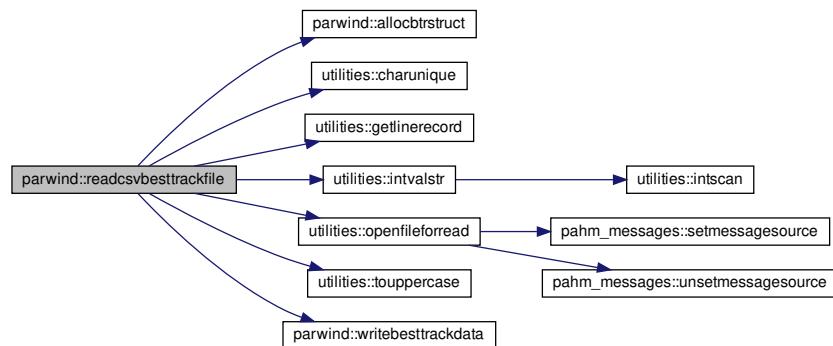
- a-deck: guidance information
- b-deck: best track information
- Skips lines that are time repeats. ???PV check
- Converts parameter values to the proper units.
- Assumes longitude is WEST longitude, latitude is NORTH latitude.

Definition at line 582 of file [parwind.F90](#).

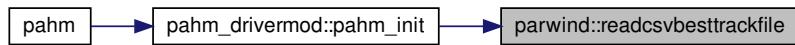
References `allocbtrstruct()`, `besttrackdata`, `pahm_global::besttrackfilename`, `utilities::charunique()`, `utilities::getline()`, `utilities::intvalstr()`, `pahm_global::nbtrfiles`, `utilities::openfileforread()`, `utilities::touppercase()`, and `writebesttrackdata()`.

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.11.1.13 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:



15.11.1.14 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

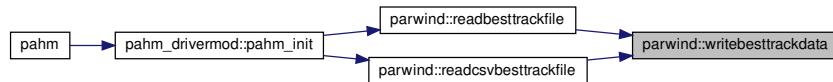
in	<i>inpFile</i>	The name of the input best track file
in	<i>trackStruc</i>	The "adjusted" best track data structure that corresponds to the <i>inpFile</i>
in	<i>suffix</i>	The suffix (optional) to be appended to the <i>inpFile</i> (default '_adj')

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:



15.11.2 Variable Documentation

15.11.2.1 approach

```
integer parwind::approach = 2
```

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

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

15.11.2.2 asyvortstru

```
type(asymmetricvortexdata_t), dimension(:), allocatable parwind::asyvortstru
```

Definition at line 243 of file [parwind.F90](#).

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

15.11.2.3 besttrackdata

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

Definition at line 118 of file [parwind.F90](#).

Referenced by [processasymmetricvortexdata\(\)](#), [processhollanddata\(\)](#), [readbesttrackfile\(\)](#), and [readcsvbesttrackfile\(\)](#).

15.11.2.4 geofactor

```
integer parwind::geofactor = 1
```

Definition at line 25 of file [parwind.F90](#).

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

15.11.2.5 geostrophicswitch

```
logical parwind::geostrophicswitch = .TRUE.
```

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

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

15.11.2.6 holstru

```
type(hollanddata_t), dimension(:), allocatable parwind::holstru
```

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

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

15.11.2.7 method

```
integer parwind::method = 4
```

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

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

15.11.2.8 stormnamelen

```
integer, parameter, private parwind::stormnamelen = 10 [private]
```

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

15.12 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 **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".
- 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) **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").

15.12.1 Function/Subroutine Documentation

15.12.1.1 **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](#).

15.12.1.2 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](#).

15.12.1.3 arraycopysingle()

```
subroutine sortutils::arraycopysingle (
    real(sp), dimension(:), intent(in) src,
```

```
real(sp), dimension(:), intent(out) dest,
integer, intent(out) nCP,
integer, intent(out) nNCP )
```

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

Parameters

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](#).

15.12.1.4 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](#).

15.12.1.5 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](#).

15.12.1.6 arrayqualsingle()

```
logical function sortutils::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](#).

15.12.1.7 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](#).

15.12.1.8 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](#).

15.12.1.9 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](#).

15.12.1.10 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](#).

15.12.1.11 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](#).

15.12.1.12 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](#).

15.12.1.13 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](#).

15.12.1.14 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](#).

15.12.1.15 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)

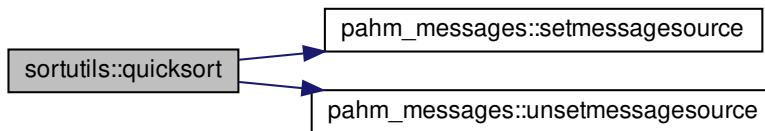
Note

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:

**15.12.1.16 sort2()**

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

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

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

Parameters

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

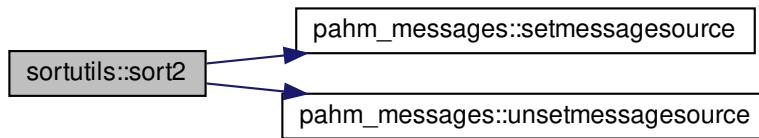
Note

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:



15.12.1.17 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](#).

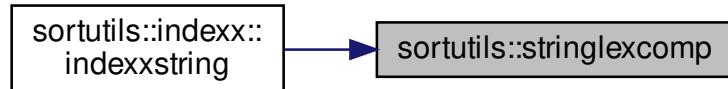
References [utilities::touppercase\(\)](#).

Referenced by [sortutils::indexx::indexxstring\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.12.1.18 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

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](#).

15.12.1.19 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](#).

15.12.1.20 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](#).

15.12.1.21 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](#).

15.12.1.22 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](#).

15.12.1.23 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](#).

15.13 timedateutils Module Reference

Data Types

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

Functions/Subroutines

- subroutine [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 [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 [gregtojuldayise](#) (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
Determines the Julian date from a Gregorian date.
- real(sz) function [gregtojuldaysec](#) (iYear, iMonth, iDay, iHour, iMin, rSec, mJD)
Determines the Julian date from a Gregorian date.
- real(sz) function [gregtojulday2](#) (iDate, iTime, mJD)
Determines the Julian date from a Gregorian date.
- subroutine [juldaytogreg](#) (julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
Determines the Julian date from a Gregorian date.
- subroutine [dayofyeartogreg](#) (inYR, inDY, iYear, iMonth, iDay)
Determines the Gregorian date (year, month, day) from a day of the year.
- subroutine [splitdatetimestring](#) (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
Splits a date string into components.
- subroutine [splitdatetimestring2](#) (inDateTime, iDate, iTime)

Splits a date string into two components.

- character(len=len(indatetime)) function [preprocessdatetimestring](#) (inDateTime)
Pre-processes an arbitrary date string.
- integer function [joindate](#) (iYear, iMonth, iDay)
Pre-processes an arbitrary date string.
- subroutine [splitdate](#) (inDate, iYear, iMonth, iDay)
Pre-processes an arbitrary date string.
- character(len=64) function [datetime2string](#) (year, month, day, hour, min, sec, sep, units, zone, err)
Constructs a NetCDF time string.
- real(sz) function [gettimeconvsec](#) (units, invert)
Calculates the conversion factor between time units and seconds.
- real(sz) function [elapsedsecs](#) (inTime1, inTime2, inUnits)
Calculates the elapsed time in seconds.
- 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 [mdjtime](#) = UNIXTIME
- real(hp), parameter [mdjoffset](#) = OFFUNIXJULDAY

15.13.1 Function/Subroutine Documentation

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

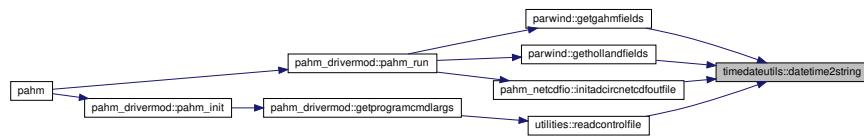
References [upp\(\)](#).

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:



15.13.1.2 dayofyear()

```

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

Determines the day of the year.

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

Parameters

in	<i>iYear</i>	The year (YYYY, integer, $1582 \leq \text{YYYY} \leq 9999$)
in	<i>iMonth</i>	The month of the year (MM, integer, $1 \leq \text{MM} \leq 12$)
in	<i>iDay</i>	The day of the month (DD, integer, $1 \leq \text{DD} \leq 31$)

Returns

myVal The day of the year number (also erroneously known as Julian day). This is 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](#).

15.13.1.3 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.

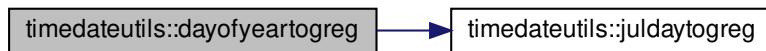
Parameters

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:



15.13.1.4 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

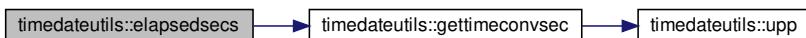
myVal The elapsed time in seconds (real). If this value is very close, within a tolerance, to the nearest whole number, it is set equal to that number.

Definition at line 1516 of file [timedateutils.F90](#).

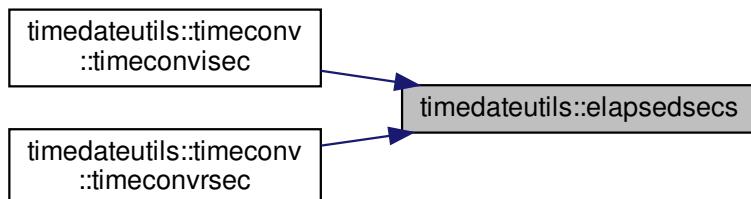
References [gettimeconvsec\(\)](#).

Referenced by [timedateutils::timeconv::timeconvise\(\)](#), and [timedateutils::timeconv::timeconvr\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.1.5 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](#).

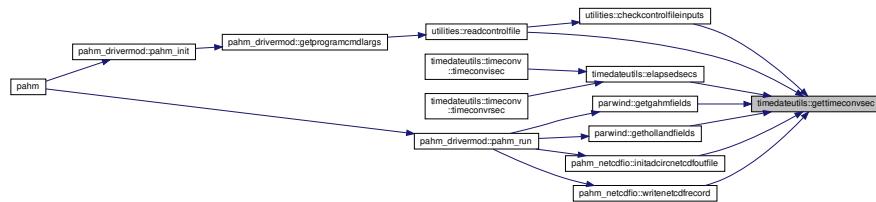
References [upp\(\)](#).

Referenced by `utilities::checkcontrolfileinputs()`, `elapsedsecs()`, `parwind::getgahmfields()`, `parwind::gethollandfields()`, `pahm_netcdfio::initadcircnetcdfoutfile()`, `utilities::readcontrolfile()`, and `pahm_netcdfio::writenetcdfrecord()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.1.6 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

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](#).

15.13.1.7 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	<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>iSec</i>	iSec The second of the minute (ss, integer, 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 536 of file [timedateutils.F90](#).

15.13.1.8 gregtojuldayrsec()

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

Determines the Julian date from a Gregorian date.

This function returns the so called Julian day number given a Gregorian date (after 10/05/1582), or the value RMISSV (-999999.0) if an error occurred.

The Julian day number of a date is the number of days that has passed since January 1, 4712 BC at 12h00 (Gregorian). It is usefull to compute differences between dates.

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

Parameters

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](#).

15.13.1.9 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)

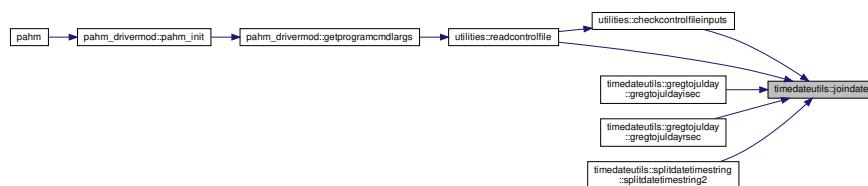
Returns

myValOut The integer date (YYYYMMDD)

Definition at line 1240 of file [timedateutils.F90](#).

Referenced by [utilities::checkcontrolfileinputs\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec2\(\)](#), [utilities::readcontrolfile\(\)](#), and [timedateutils::splidatetimestring::splidatetimestring2\(\)](#).

Here is the caller graph for this function:



15.13.1.10 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 useful 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: <i>mJD</i> >= 1 otherwise use: <i>mJD</i> < 1 default: <i>mJD</i> = 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 <i>MJD</i> = <i>JD</i> - 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

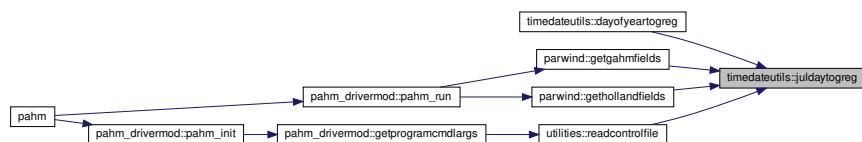
The code was adopted from the D-Flow FM source (time_module.f90/JULIAN)

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:



15.13.1.11 **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 \leq YYYY)
----	--------------	--

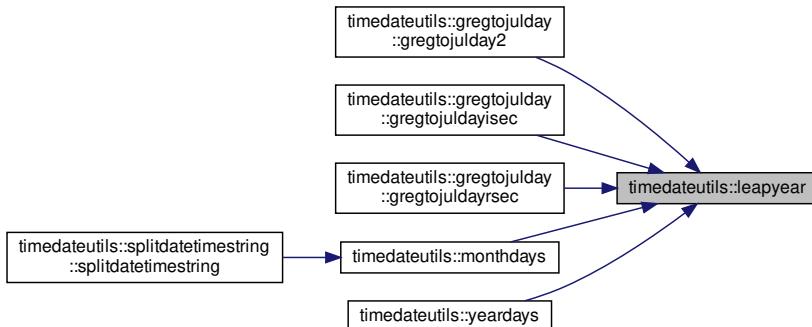
Returns

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::gregtojuldayrsec\(\)](#), [monthdays\(\)](#), and [yeardays\(\)](#).

Here is the caller graph for this function:



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

Returns

myVal The days of the month

Definition at line 403 of file [timedateutils.F90](#).

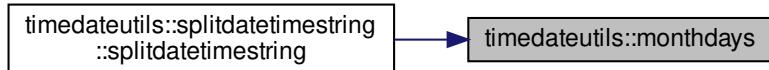
References [pahm_sizes::imissv](#), and [leapyear\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.1.13 **preprocessdatetimestring()**

```
character(len=len(indatetime)) function timedateutils::preprocessdatetimestring (  
    character(len=*) , intent(in) inDateTime )
```

Pre-processes an arbitrary date string.

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

Parameters

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

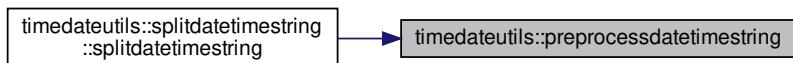
Returns

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

Definition at line 1185 of file [timedateutils.F90](#).

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

Here is the caller graph for this function:



15.13.1.14 **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:

**15.13.1.15 `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	<i>inDateTime</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](#).

15.13.1.16 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](#).

15.13.1.17 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](#).

15.13.1.18 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 TimeConvISEC 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](#).

15.13.1.19 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
----	------------------	------------------

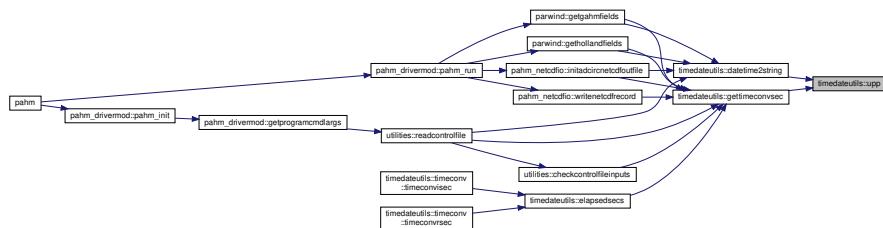
Returns

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:



15.13.1.20 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	<i>iYear</i>	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:



15.13.2 Variable Documentation

15.13.2.1 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::gregtojulday3\(\)](#) and [timedateutils::gregtojulday::gregtojuldaysec\(\)](#).

15.13.2.2 firstgregtime

```
integer, parameter timedateutils::firstgregtime = 0 * 10000 + 0 * 100 + 0
```

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

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

15.13.2.3 mdjdate

```
integer, parameter timedateutils::mdjdate = UNIXDATE
```

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

15.13.2.4 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::gregtojuldaysec\(\)](#), and [juldaytogreg\(\)](#).

15.13.2.5 mdjtime

```
integer, parameter timedateutils::mdjtime = UNIXTIME
```

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

15.13.2.6 modeldate

```
integer, parameter timedateutils::modeldate = 1990 * 10000 + 1 * 100 + 1
```

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

15.13.2.7 modeltime

```
integer, parameter timedateutils::modeltime = 0 * 10000 + 0 * 100 + 0
```

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

15.13.2.8 modjuldate

```
integer, parameter timedateutils::modjuldate = 1858 * 10000 + 11 * 100 + 17
```

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

15.13.2.9 modjultime

```
integer, parameter timedateutils::modjultime = 0 * 10000 + 0 * 100 + 0
```

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

15.13.2.10 offfirstgregday

```
real(hp), parameter timedateutils::offfirstgregday = 2299150.5_HP
```

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

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

15.13.2.11 offmodeljulday

```
real(hp), parameter timedateutils::offmodeljulday = 2447892.5_HP
```

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

15.13.2.12 offmodjulday

```
real(hp), parameter timedateutils::offmodjulday = 2400000.5_HP
```

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

15.13.2.13 offunixjulday

```
real(hp), parameter timedateutils::offunixjulday = 2440587.5_HP
```

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

15.13.2.14 unixdate

```
integer, parameter timedateutils::unixdate = 1970 * 10000 + 1 * 100 + 1
```

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

15.13.2.15 unixtime

```
integer, parameter timedateutils::unixtime = 0 * 10000 + 0 * 100 + 0
```

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

15.13.2.16 usemodjulday

```
integer, parameter timedateutils::usemodjulday = 0
```

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

Referenced by [timedateutils::gregtojulday::gregtojulday2\(\)](#), [timedateutils::gregtojulday::gregtojuldayisec\(\)](#), [timedateutils::gregtojulday::gregtojuldaysec\(\)](#), and [juldaytogram\(\)](#).

15.14 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 [cptogegeo_scalar](#) (x, y, lat0, lon0, lat, lon)

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- subroutine [cptogegeo_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**

15.14.1 Function/Subroutine Documentation

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

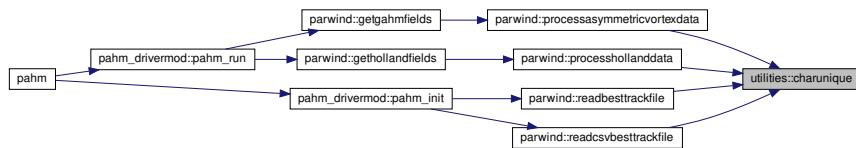
Returns

`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:



15.14.1.2 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

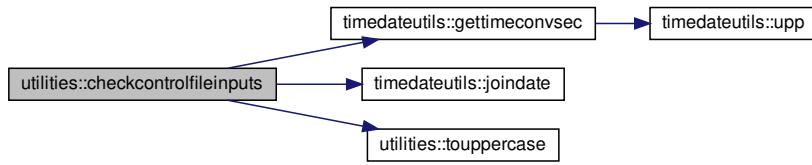
myStatus: The error status, no error: status = 0

Definition at line 1147 of file [utilities.F90](#).

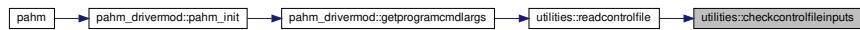
References [pahm_global::backgroundatmpress](#), [pahm_global::begsimspecified](#), [pahm_global::begsimtime](#), [pahm_global::besttrackfilename](#), [pahm_global::besttrackfilenamespecified](#), [closetol](#), [pahm_global::def_ncnam_pres](#), [pahm_global::def_ncnam_wndx](#), [pahm_global::def_ncnam_wndy](#), [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:



15.14.1.3 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](#).

15.14.1.4 cpptogeo_1d()

```
subroutine utilities::cpptogeo_1d (
    real(sz), dimension(:), intent(in) x,
    real(sz), dimension(:), intent(in) y,
```

```

real(sz), intent(in) lat0,
real(sz), intent(in) lon0,
real(sz), dimension(:), intent(out) lat,
real(sz), dimension(:), intent(out) lon )

```

Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.

Transforms 1D CPP coordinates into 1D geographical coordinates. This is the Equidistant Cylindrical projection, also called equirectangular projection, equidirectional projection, geographic projection, plate carree or carte parallelogrammatique projection.

Parameters

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](#).

15.14.1.5 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](#).

15.14.1.6 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

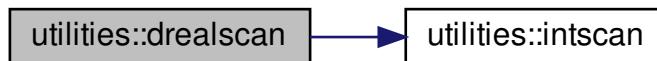
https://www-nds.iaea.org/workshops/smr1939/Codes/ENSDF_Codes/mswindows/nsdflib/nsdflib__win.html

Definition at line 2919 of file [utilities.F90](#).

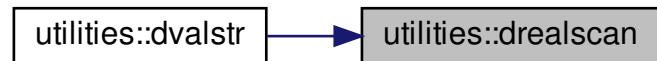
References [intscan\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.1.7 dvalstr()

```
real(hp) function utilities::dvalstr (
    character(len=*) , intent(in) String )
```

Returns the value of the leading double precision real numeric string.

Parameters

in	String	The input string
----	--------	------------------

Returns

myVal: The value of the real number in double precision as extracted from the input string

Author

C. L. Dunford - November 19, 2003
 NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

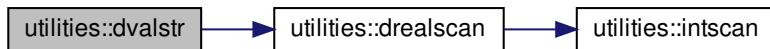
See also

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

Definition at line 2661 of file [utilities.F90](#).

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

Here is the call graph for this function:

**15.14.1.8 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)
Generated by Doxygen		Calculated Y coordinate: y (m) - real, 1D array (output)

Definition at line 1854 of file [utilities.F90](#).

15.14.1.9 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](#).

15.14.1.10 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
----	----------------	---------------------

Parameters

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)

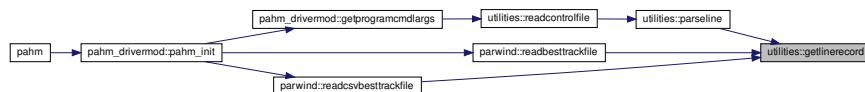
Returns

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:

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

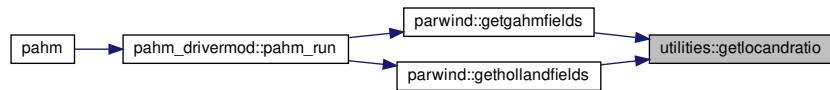
Parameters

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 Generated by Doxygen	<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:



15.14.1.12 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

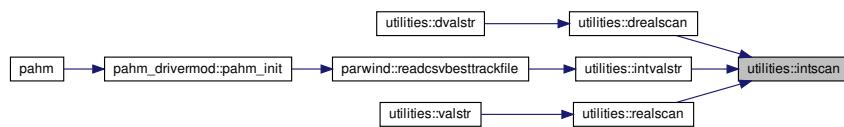
See also

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

Here is the caller graph for this function:



15.14.1.13 intvalstr()

```
integer function utilities::intvalstr (
    character(len=*), intent(in) String )
```

Returns the value of the leading integer numeric string.

Parameters

in	<i>String</i>	The input string
----	---------------	------------------

Returns

`myVal`: The value of the integer number as extracted from the input string

Author

C. L. Dunford - November 19, 2003
NSDFLIB, FORTRAN UTILITY SUBROUTINE PACKAGE

See also

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

Definition at line 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:



15.14.1.14 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

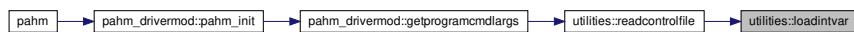
Note

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:

**15.14.1.15 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_l)

Definition at line 1551 of file [utilities.F90](#).

15.14.1.16 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

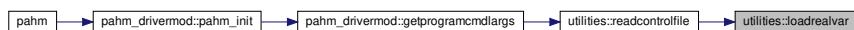
Note

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:



15.14.1.17 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

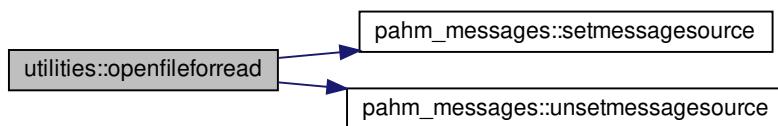
in	<i>lun</i>	The logical unit number (LUN) to use
in	<i>fileName</i>	The full pathname of the input file
out	<i>errorIO</i>	The error status, no error: status = 0 (output)

Definition at line 68 of file [utilities.F90](#).

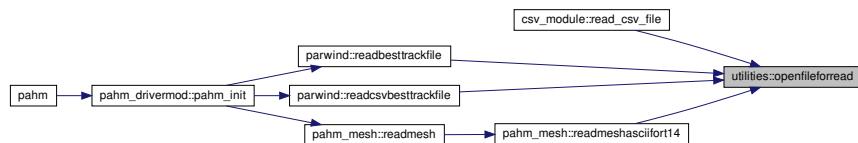
References [pahm_messages::error](#), [pahm_messages::info](#), [pahm_messages::scratchmessage](#), [pahm_messages::setmessagesource](#) and [pahm_messages::unsetmessagesource\(\)](#).

Referenced by [csv_module::read_csv_file\(\)](#), [parwind::readbesttrackfile\(\)](#), [parwind::readcsvbesttrackfile\(\)](#), and [pahm_mesh::readmeshascifort14\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.1.18 parseline()

```

integer function utilities::parseline (
    character(len=*), intent(in)  inLine,
    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)
<i>in,out</i>	<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

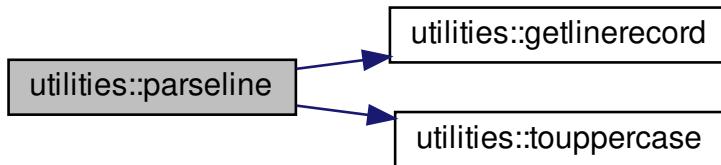
Adopted from the ROMS source (Utility/inp_par.F, decode_line)

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:



15.14.1.19 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](#).

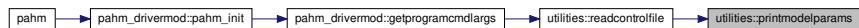
References [pahm_global::backgroundatmpress](#), [pahm_global::begdatespecified](#), [pahm_global::begdatetime](#), [pahm_global::begday](#), [pahm_global::beghour](#), [pahm_global::begmin](#), [pahm_global::begmonth](#), [pahm_global::begsec](#), [pahm_global::begsimspecified](#), [pahm_global::begsimtime](#), [pahm_global::begyear](#), [pahm_global::besttrackfilename](#), [pahm_global::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::mesh filetype](#), [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::outfile name](#), [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:



15.14.1.20 `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

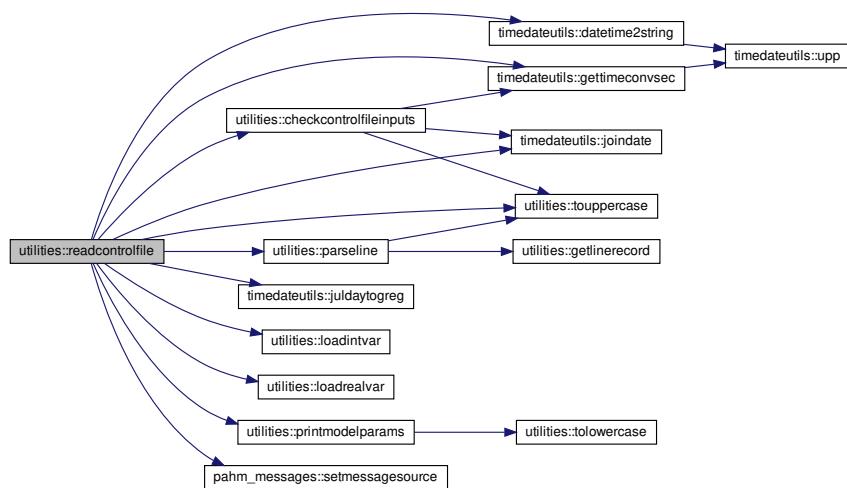
in	<i>inpFile</i>	The full pathname of the input file
----	----------------	-------------------------------------

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

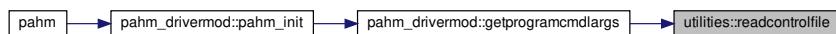
References `pahm_sizes::blank`, `checkcontrolfileinputs()`, `closetol`, `timedateutils::datetime2string()`, `timedateutils::gettimeconvsec()`, `pahm_sizes::imissv`, `timedateutils::joindate()`, `timedateutils::juldaytoreg()`, `loadintvar()`, `loadrealvar()`, `pahm_global::lun_ctrl`, `pahm_global::lun_screen`, `numbitfiles`, `parseline()`, `printmodelparams()`, `pahm_messages::setmessagesource()`, and `touppercase()`.

Referenced by `pahm_drivermod::getprogramcmdlargs()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.1.21 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](#).

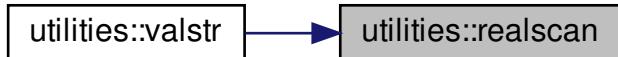
References [intscan\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.1.22 sphericaldistance_1d()

```
real(sz) function, dimension(:), allocatable utilities::sphericaldistance_1d (
    real(sz), dimension(:), intent(in) lats,
    real(sz), dimension(:), intent(in) lons,
    real(sz), intent(in) lat0,
    real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas
https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".

Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points (Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

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](#).

15.14.1.23 sphericaldistance_2d()

```
real(sz) function, dimension(:, :, ), allocatable utilities::sphericaldistance_2d (
    real(sz), dimension(:, :, ), intent(in) lats,
    real(sz), dimension(:, :, ), intent(in) lons,
    real(sz), intent(in) lat0,
    real(sz), intent(in) lon0 )
```

Calculates the distance of points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas
https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".

Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points (Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

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](#).

15.14.1.24 sphericaldistance_scalar()

```
real(sz) function utilities::sphericaldistance_scalar (
    real(sz), intent(in) lat1,
    real(sz), intent(in) lon1,
    real(sz), intent(in) lat2,
    real(sz), intent(in) lon2 )
```

Calculates the distance of two points along the great circle using the Vincenty formula.

Function to get the great-circle distance along the surface of a sphere (the earth's surface in this case). Compute the great-circle distance using the Vincenty formula for distance along a sphere.

See also

https://en.wikipedia.org/wiki/Great-circle_distance#Computational_formulas
https://en.wikipedia.org/wiki/Vincenty's_formulae

Vincenty, Thaddeus (April 1975a). "Direct and Inverse Solutions of Geodesics \n on the Ellipsoid with application of nested equations".

Survey Review. XXIII (176): 88-93. doi:10.1179/sre.1975.23.176.88.

Vincenty, Thaddeus (August 1975b). Geodetic inverse solution between antipodal points (Technical report). DMAAC Geodetic Survey Squadron. doi:10.5281/zenodo.32999.

Parameters

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](#).

15.14.1.25 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](#).

15.14.1.26 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

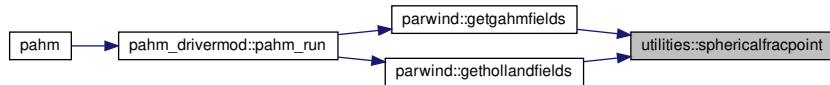
in	<i>lat1</i>	Latitude of the first point (degrees north)
in	<i>lon1</i>	Longitude of the first point (degrees east)
in	<i>lat2</i>	Latitude of the second point (degrees north)
in	<i>lon2</i>	Longitude of the second point (degrees east)
in	<i>fraction</i>	The fraction of the distance between points 1 and 2 where the intermediate point is located ($0 \leq \text{fraction} \leq 1$)
out	<i>latt</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:



15.14.1.27 `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
----	------------------	------------------

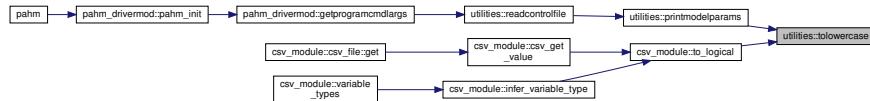
Returns

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:



15.14.1.28 `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
----	------------------	------------------

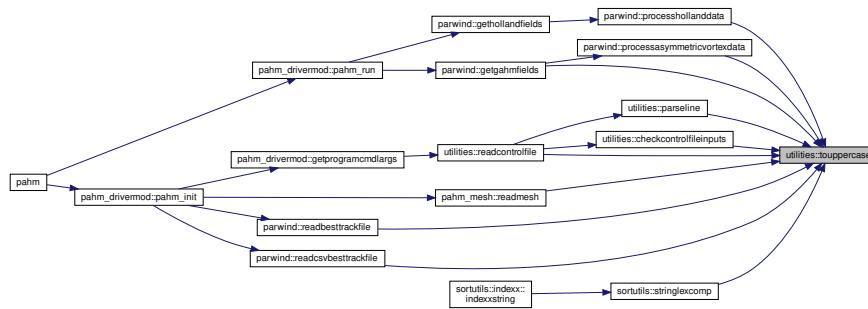
Returns

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:



15.14.1.29 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

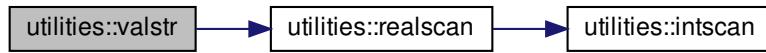
See also

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:



15.14.2 Variable Documentation

15.14.2.1 closetol

```
real(sz), parameter utilities::closetol = 0.001_sz
```

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

Referenced by [checkcontrolfileinputs\(\)](#), and [readcontrolfile\(\)](#).

15.14.2.2 numbtfiles

```
integer, private utilities::numbtfiles = 0 [private]
```

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

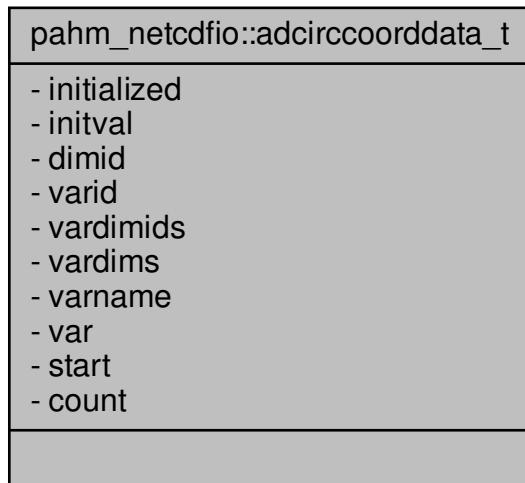
Referenced by [checkcontrolfileinputs\(\)](#), and [readcontrolfile\(\)](#).

Chapter 16

Data Type Documentation

16.1 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`

16.1.1 Detailed Description

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

16.1.2 Member Data Documentation

16.1.2.1 count

```
integer, dimension(1) pahm_netcdfio::adcirccoorddata_t::count [private]
```

Definition at line 64 of file [netcdfio.F90](#).

16.1.2.2 dimid

```
integer pahm_netcdfio::adcirccoorddata_t::dimid [private]
```

Definition at line 58 of file [netcdfio.F90](#).

16.1.2.3 initialized

```
logical pahm_netcdfio::adcirccoorddata_t::initialized = .FALSE. [private]
```

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

16.1.2.4 initval

```
real(sz) pahm_netcdfio::adcirccoorddata_t::initval [private]
```

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

16.1.2.5 start

```
integer, dimension(1) pahm_netcdfio::adcirccoorddata_t::start [private]
```

Definition at line 64 of file [netcdfio.F90](#).

16.1.2.6 var

```
real(sz), dimension(:), allocatable pahm_netcdfio::adcirccoorddata_t::var [private]
```

Definition at line 63 of file [netcdfio.F90](#).

16.1.2.7 vardimids

```
integer pahm_netcdfio::adcirccoorddata_t::vardimids [private]
```

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

16.1.2.8 vardims

```
integer pahm_netcdfio::adcirccoorddata_t::vardims [private]
```

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

16.1.2.9 varid

```
integer pahm_netcdfio::adcirccoorddata_t::varid [private]
```

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

16.1.2.10 varname

```
character(50) pahm_netcdfio::adcircvardata3d_t::varname [private]
```

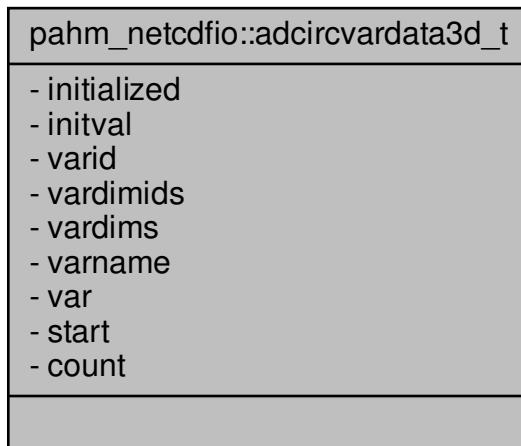
Definition at line 62 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

16.2 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](#)

16.2.1 Detailed Description

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

16.2.2 Member Data Documentation

16.2.2.1 count

```
integer, dimension(3) pahm_ncdfio::adcircvardata3d_t::count [private]
```

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

16.2.2.2 initialized

```
logical pahm_ncdfio::adcircvardata3d_t::initialized = .FALSE. [private]
```

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

16.2.2.3 initval

```
real(sz) pahm_ncdfio::adcircvardata3d_t::initval [private]
```

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

16.2.2.4 start

```
integer, dimension(3) pahm_ncdfio::adcircvardata3d_t::start [private]
```

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

16.2.2.5 var

```
real(sz), dimension(:, :, :), allocatable pahm_ncdfio::adcircvardata3d_t::var [private]
```

Definition at line 85 of file [netcdfio.F90](#).

16.2.2.6 vardimids

```
integer, dimension(3) pahm_ncdfio::adcircvardata3d_t::vardimids [private]
```

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

16.2.2.7 vardims

```
integer, dimension(3) pahm_ncdfio::adcircvardata3d_t::vardims [private]
```

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

16.2.2.8 varid

```
integer pahm_ncdfio::adcircvardata3d_t::varid [private]
```

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

16.2.2.9 varname

```
character(50) pahm_ncdfio::adcircvardata3d_t::varname [private]
```

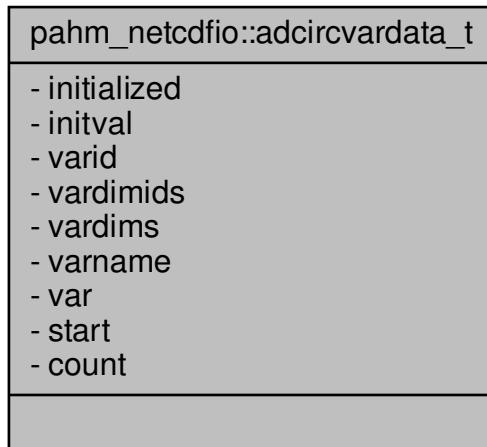
Definition at line 84 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

16.3 pahm_netcdfio::adcircvardata_t Type Reference

Collaboration diagram for pahm_netcdfio::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`

16.3.1 Detailed Description

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

16.3.2 Member Data Documentation

16.3.2.1 count

```
integer, dimension(2) pahm_netcdfio::adcircvardata_t::count [private]
```

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

16.3.2.2 initialized

```
logical pahm_netcdfio::adcircvardata_t::initialized = .FALSE. [private]
```

Definition at line 68 of file [netcdfio.F90](#).

16.3.2.3 initval

```
real(sz) pahm_netcdfio::adcircvardata_t::initval [private]
```

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

16.3.2.4 start

```
integer, dimension(2) pahm_netcdfio::adcircvardata_t::start [private]
```

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

16.3.2.5 var

```
real(sz), dimension(:, :), allocatable pahm_netcdfio::adcircvardata_t::var [private]
```

Definition at line 74 of file [netcdfio.F90](#).

16.3.2.6 vardimids

```
integer, dimension(2) pahm_netcdfio::adcircvardata_t::vardimids [private]
```

Definition at line 71 of file [netcdfio.F90](#).

16.3.2.7 vardims

```
integer, dimension(2) pahm_netcdfio::adcircvardata_t::vardims [private]
```

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

16.3.2.8 varid

```
integer pahm_netcdfio::adcircvardata_t::varid [private]
```

Definition at line 70 of file [netcdfio.F90](#).

16.3.2.9 varname

```
character(50) pahm_netcdfio::adcircvardata_t::varname [private]
```

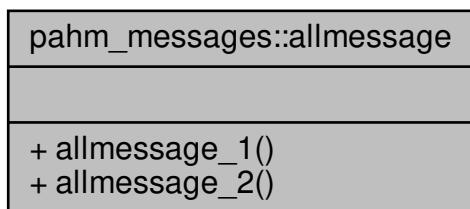
Definition at line 73 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

16.4 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)

16.4.1 Detailed Description

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

16.4.2 Member Function/Subroutine Documentation

16.4.2.1 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](#).

16.4.2.2 allmessage_2()

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

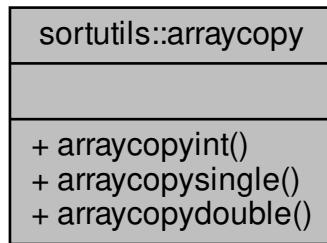
Definition at line [330](#) of file [messages.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[messages.F90](#)

16.5 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".

16.5.1 Detailed Description

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

16.5.2 Member Function/Subroutine Documentation

16.5.2.1 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](#).

16.5.2.2 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](#).

16.5.2.3 arraycopysingle()

```
subroutine sortutils::arraycopy::arraycopysingle (
    real(sp), dimension(:), intent(in) src,
```

```
real(sp), dimension(:), intent(out) dest,
integer, intent(out) nCP,
integer, intent(out) nNCP )
```

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

Parameters

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

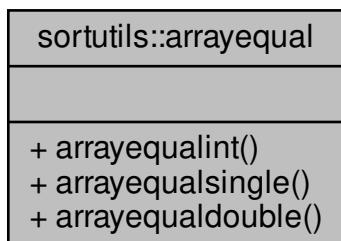
Definition at line 1205 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

16.6 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 `arrayequaldouble` (`arr1, arr2`)
Compares two one-dimensional double precision arrays for equality.

16.6.1 Detailed Description

Definition at line 44 of file `sortutils.F90`.

16.6.2 Member Function/Subroutine Documentation

16.6.2.1 `arrayequaldouble()`

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

Compares two one-dimensional double precision arrays for equality.

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

Parameters

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`.

16.6.2.2 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](#).

16.6.2.3 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.

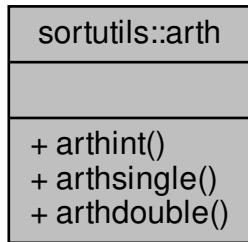
Definition at line 1326 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

16.7 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").

16.7.1 Detailed Description

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

16.7.2 Member Function/Subroutine Documentation

16.7.2.1 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](#).

16.7.2.2 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](#).

16.7.2.3 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

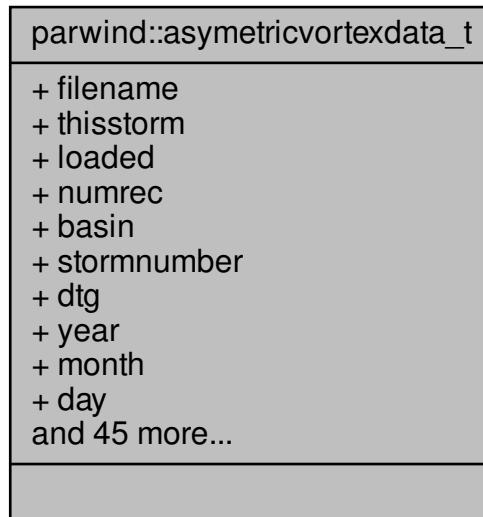
Definition at line 1876 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

16.8 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 `trvy`
- 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 `hollbs`
- real(sz), dimension(:, :), allocatable `vmaxesbl`

16.8.1 Detailed Description

Definition at line 164 of file `parwind.F90`.

16.8.2 Member Data Documentation

16.8.2.1 `basin`

```
character(len=2), dimension(:), allocatable parwind::asymmetricvortexdata_t::basin
```

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

16.8.2.2 `casttime`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::casttime
```

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

16.8.2.3 `casttype`

```
character(len=4), dimension(:), allocatable parwind::asymmetricvortexdata_t::casttype
```

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

16.8.2.4 `casttypenum`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::casttypenum
```

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

16.8.2.5 `cpress`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::cpress
```

Definition at line 187 of file [parwind.F90](#).

16.8.2.6 `day`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::day
```

Definition at line 173 of file [parwind.F90](#).

16.8.2.7 dir

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::dir
```

Definition at line [224](#) of file [parwind.F90](#).

16.8.2.8 dtg

```
character(len=10), dimension(:), allocatable parwind::asymmetricvortexdata_t::dtg
```

Definition at line [172](#) of file [parwind.F90](#).

16.8.2.9 ew

```
character(len=1), dimension(:), allocatable parwind::asymmetricvortexdata_t::ew
```

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

16.8.2.10 eye

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::eye
```

Definition at line [206](#) of file [parwind.F90](#).

16.8.2.11 fcstinc

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::fcstinc
```

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

16.8.2.12 filename

```
character(len=fnamelen) parwind::asymmetricvortexdata_t::filename
```

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

16.8.2.13 `gusts`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::gusts
```

Definition at line 205 of file [parwind.F90](#).

16.8.2.14 `hollb`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::hollb
```

Definition at line 238 of file [parwind.F90](#).

16.8.2.15 `hollsbs`

```
real(sz), dimension(:, :), allocatable parwind::asymmetricvortexdata_t::hollsbs
```

Definition at line 239 of file [parwind.F90](#).

16.8.2.16 `hour`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::hour
```

Definition at line 173 of file [parwind.F90](#).

16.8.2.17 `icpress`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::icpress
```

Definition at line 186 of file [parwind.F90](#).

16.8.2.18 `idir`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::idir
```

Definition at line 223 of file [parwind.F90](#).

16.8.2.19 ilat

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ilat
```

Definition at line 179 of file [parwind.F90](#).

16.8.2.20 ilon

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ilon
```

Definition at line 179 of file [parwind.F90](#).

16.8.2.21 initials

```
character(len=3), dimension(:), allocatable parwind::asymmetricvortexdata_t::initials
```

Definition at line 218 of file [parwind.F90](#).

16.8.2.22 iprp

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::iprp
```

Definition at line 196 of file [parwind.F90](#).

16.8.2.23 ir

```
integer, dimension(:, :), allocatable parwind::asymmetricvortexdata_t::ir
```

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

16.8.2.24 irmw

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::irmw
```

Definition at line 202 of file [parwind.F90](#).

16.8.2.25 irrp

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::irrp
```

Definition at line 199 of file [parwind.F90](#).

16.8.2.26 isotachspercycle

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::isotachspercycle
```

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

16.8.2.27 ispeed

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ispeed
```

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

16.8.2.28 istormspeed

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::istormspeed
```

Definition at line 225 of file [parwind.F90](#).

16.8.2.29 ivr

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::ivr
```

Definition at line 191 of file [parwind.F90](#).

16.8.2.30 lat

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::lat
```

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

16.8.2.31 loaded

```
logical parwind::asymmetricvortexdata_t::loaded = .FALSE.
```

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

16.8.2.32 lon

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::lon
```

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

16.8.2.33 maxseas

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::maxseas
```

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

16.8.2.34 month

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::month
```

Definition at line 173 of file [parwind.F90](#).

16.8.2.35 ncycles

```
integer parwind::asymmetricvortexdata_t::ncycles
```

Definition at line 233 of file [parwind.F90](#).

16.8.2.36 ns

```
character(len=1), dimension(:), allocatable parwind::asymmetricvortexdata_t::ns
```

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

16.8.2.37 numcycle

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::numcycle
```

Definition at line 234 of file [parwind.F90](#).

16.8.2.38 numrec

```
integer parwind::asymmetricvortexdata_t::numrec
```

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

16.8.2.39 prp

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::prp
```

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

16.8.2.40 quadflag

```
integer, dimension(:, :), allocatable parwind::asymmetricvortexdata_t::quadflag
```

Definition at line 236 of file [parwind.F90](#).

16.8.2.41 rmaxw

```
real(sz), dimension(:, :, :), allocatable parwind::asymmetricvortexdata_t::rmaxw
```

Definition at line 237 of file [parwind.F90](#).

16.8.2.42 rmw

```
real(sz), dimension(:, :, :, :), allocatable parwind::asymmetricvortexdata_t::rmw
```

Definition at line 203 of file [parwind.F90](#).

16.8.2.43 `rrp`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::rrp
```

Definition at line 200 of file [parwind.F90](#).

16.8.2.44 `speed`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::speed
```

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

16.8.2.45 `stormname`

```
character(len=stormnamelen), dimension(:), allocatable parwind::asymmetricvortexdata_t::stormname
```

Definition at line 227 of file [parwind.F90](#).

16.8.2.46 `stormnumber`

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::stormnumber
```

Definition at line 171 of file [parwind.F90](#).

16.8.2.47 `stormspeed`

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::stormspeed
```

Definition at line 226 of file [parwind.F90](#).

16.8.2.48 `subregion`

```
character(len=3), dimension(:), allocatable parwind::asymmetricvortexdata_t::subregion
```

Definition at line 207 of file [parwind.F90](#).

16.8.2.49 thisstorm

```
character(len=10) parwind::asymmetricvortexdata_t::thisstorm
```

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

16.8.2.50 trvx

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::trvx
```

Definition at line 220 of file [parwind.F90](#).

16.8.2.51 trvy

```
real(sz), dimension(:), allocatable parwind::asymmetricvortexdata_t::trvy
```

Definition at line 220 of file [parwind.F90](#).

16.8.2.52 ty

```
character(len=2), dimension(:), allocatable parwind::asymmetricvortexdata_t::ty
```

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

16.8.2.53 vmaxesbl

```
real(sz), dimension(:, :), allocatable parwind::asymmetricvortexdata_t::vmaxesbl
```

Definition at line 240 of file [parwind.F90](#).

16.8.2.54 windcode

```
character(len=3), dimension(:), allocatable parwind::asymmetricvortexdata_t::windcode
```

Definition at line 192 of file [parwind.F90](#).

16.8.2.55 year

```
integer, dimension(:), allocatable parwind::asymmetricvortexdata_t::year
```

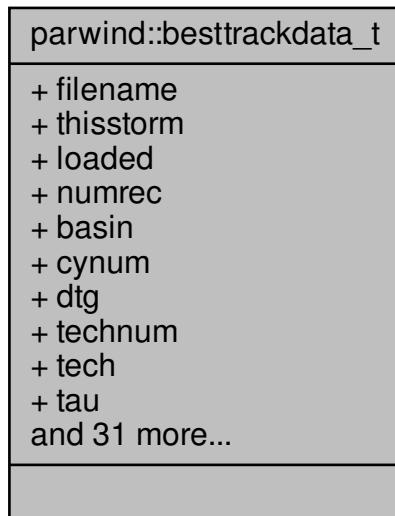
Definition at line 173 of file [parwind.F90](#).

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

- [/home/takis/CSDL/parwinds-doc/src/parwind.F90](#)

16.9 parwind::besttrackdata_t Type Reference

Collaboration diagram for parwind::besttrackdata_t:



Public Attributes

- character(len=fnamelen) **filename**
- character(len=10) **thisstorm**
- logical **loaded** = .FALSE.
- integer **numrec**
- character(len=2), dimension(:), allocatable **basin**
- integer, dimension(:), allocatable **cnum**
- 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 `inrmw`
- 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`

16.9.1 Detailed Description

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

16.9.2 Member Data Documentation

16.9.2.1 basin

```
character(len=2), dimension(:), allocatable parwind::besttrackdata_t::basin
```

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

16.9.2.2 cyclenum

```
integer, dimension(:), allocatable parwind::besttrackdata_t::cyclenum
```

Definition at line 110 of file [parwind.F90](#).

16.9.2.3 cnum

```
integer, dimension(:), allocatable parwind::besttrackdata_t::cnum
```

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

16.9.2.4 day

```
integer, dimension(:), allocatable parwind::besttrackdata_t::day
```

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

16.9.2.5 dir

```
integer, dimension(:), allocatable parwind::besttrackdata_t::dir
```

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

16.9.2.6 dtg

```
character(len=10), dimension(:), allocatable parwind::besttrackdata_t::dtg
```

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

16.9.2.7 ew

```
character(len=1), dimension(:), allocatable parwind::besttrackdata_t::ew
```

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

16.9.2.8 eye

```
integer, dimension(:), allocatable parwind::besttrackdata_t::eye
```

Definition at line 91 of file [parwind.F90](#).

16.9.2.9 filename

```
character(len=fnamelen) parwind::besttrackdata_t::filename
```

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

16.9.2.10 gusts

```
integer, dimension(:), allocatable parwind::besttrackdata_t::gusts
```

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

16.9.2.11 hour

```
integer, dimension(:), allocatable parwind::besttrackdata_t::hour
```

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

16.9.2.12 initials

```
character(len=3), dimension(:), allocatable parwind::besttrackdata_t::initials
```

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

16.9.2.13 intlat

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intlat
```

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

16.9.2.14 intlon

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intlon
```

Definition at line 50 of file [parwind.F90](#).

16.9.2.15 intmslp

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intmslp
```

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

16.9.2.16 intpouter

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intpouter
```

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

16.9.2.17 intrad1

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intrad1
```

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

16.9.2.18 intrad2

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intrad2
```

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

16.9.2.19 intrad3

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intrad3
```

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

16.9.2.20 intrad4

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intrad4
```

Definition at line 85 of file [parwind.F90](#).

16.9.2.21 intrmw

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intrmw
```

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

16.9.2.22 introuter

```
integer, dimension(:), allocatable parwind::besttrackdata_t::introuter
```

Definition at line 88 of file [parwind.F90](#).

16.9.2.23 intspeed

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intspeed
```

Definition at line 105 of file [parwind.F90](#).

16.9.2.24 intvmax

```
integer, dimension(:), allocatable parwind::besttrackdata_t::intvmax
```

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

16.9.2.25 lat

```
real(sz), dimension(:), allocatable parwind::besttrackdata_t::lat
```

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

16.9.2.26 loaded

```
logical parwind::besttrackdata_t::loaded = .FALSE.
```

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

16.9.2.27 lon

```
real(sz), dimension(:), allocatable parwind::besttrackdata_t::lon
```

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

16.9.2.28 maxseas

```
integer, dimension(:), allocatable parwind::besttrackdata_t::maxseas
```

Definition at line 102 of file [parwind.F90](#).

16.9.2.29 month

```
integer, dimension(:), allocatable parwind::besttrackdata_t::month
```

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

16.9.2.30 ns

```
character(len=1), dimension(:), allocatable parwind::besttrackdata_t::ns
```

Definition at line 52 of file [parwind.F90](#).

16.9.2.31 numrec

```
integer parwind::besttrackdata_t::numrec
```

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

16.9.2.32 rad

```
integer, dimension(:), allocatable parwind::besttrackdata_t::rad
```

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

16.9.2.33 stormname

```
character(len=stormnamelen), dimension(:), allocatable parwind::besttrackdata_t::stormname
```

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

16.9.2.34 subregion

```
character(len=3), dimension(:), allocatable parwind::besttrackdata_t::subregion
```

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

16.9.2.35 tau

```
integer, dimension(:), allocatable parwind::besttrackdata_t::tau
```

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

16.9.2.36 tech

```
character(len=4), dimension(:), allocatable parwind::besttrackdata_t::tech
```

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

16.9.2.37 technum

```
integer, dimension(:), allocatable parwind::besttrackdata_t::technum
```

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

16.9.2.38 thisstorm

```
character(len=10) parwind::besttrackdata_t::thisstorm
```

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

16.9.2.39 ty

```
character(len=2), dimension(:), allocatable parwind::besttrackdata_t::ty
```

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

16.9.2.40 windcode

```
character(len=3), dimension(:), allocatable parwind::besttrackdata_t::windcode
```

Definition at line 76 of file [parwind.F90](#).

16.9.2.41 year

```
integer, dimension(:), allocatable parwind::besttrackdata_t::year
```

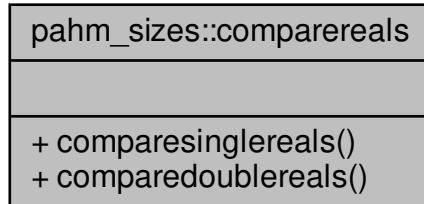
Definition at line 113 of file [parwind.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[parwind.F90](#)

16.10 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.

16.10.1 Detailed Description

Definition at line 22 of file `sizes.F90`.

16.10.2 Member Function/Subroutine Documentation

16.10.2.1 comparedoublereals()

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

Compares two double precision numbers.

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

Parameters

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

Returns

myValOut

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

Note

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

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

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

Returns

myValOut

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

Note

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

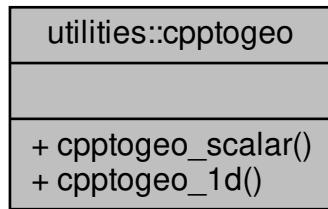
Definition at line 168 of file [sizes.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sizes.F90](#)

16.11 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.

16.11.1 Detailed Description

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

16.11.2 Member Function/Subroutine Documentation

16.11.2.1 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 parallelographmatique 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](#).

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

16.11.2.2 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 parallelographmatique 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](#).

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](#)

16.12 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'`

16.12.1 Detailed Description

Definition at line 45 of file `csv_module.F90`.

16.12.2 Member Function/Subroutine Documentation

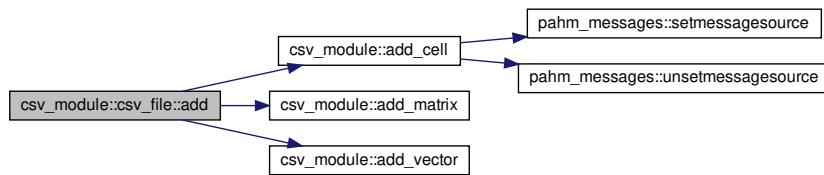
16.12.2.1 add()

```
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:



16.12.2.2 add_cell()

```
procedure csv_module::csv_file::add_cell
```

Definition at line 118 of file [csv_module.F90](#).

16.12.2.3 add_matrix()

```
procedure csv_module::csv_file::add_matrix
```

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

16.12.2.4 add_vector()

```
procedure csv_module::csv_file::add_vector
```

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

16.12.2.5 close()

```
procedure, public csv_module::csv_file::close
```

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

16.12.2.6 csv_get_value()

```
procedure csv_module::csv_file::csv_get_value
```

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

16.12.2.7 destroy()

```
procedure, public csv_module::csv_file::destroy
```

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

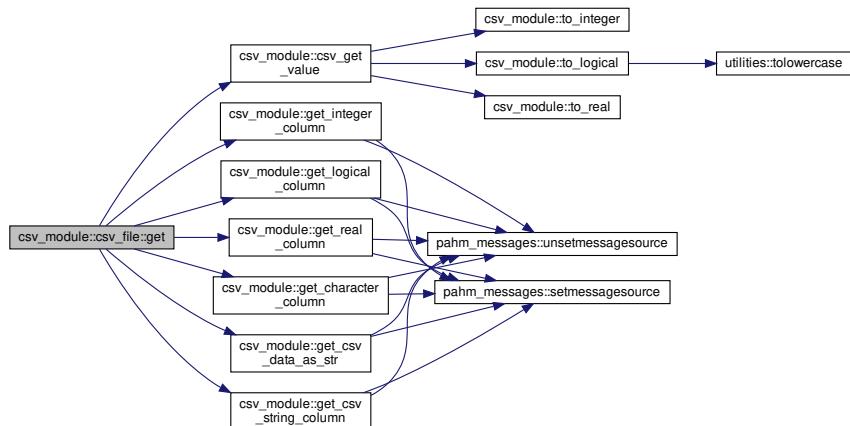
16.12.2.8 get()

```
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:



16.12.2.9 get_character_column()

```
procedure csv_module::csv_file::get_character_column
```

Definition at line 110 of file [csv_module.F90](#).

16.12.2.10 get_column()

```
procedure csv_module::csv_file::get_column
```

Definition at line 127 of file [csv_module.F90](#).

16.12.2.11 get_csv_data_as_str()

```
procedure csv_module::csv_file::get_csv_data_as_str
```

Definition at line 105 of file [csv_module.F90](#).

16.12.2.12 get_csv_string_column()

```
procedure csv_module::csv_file::get_csv_string_column
```

Definition at line 111 of file [csv_module.F90](#).

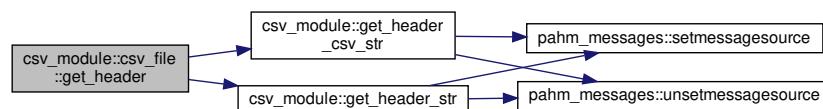
16.12.2.13 get_header()

```
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:



16.12.2.14 get_header_csv_str()

```
procedure csv_module::csv_file::get_header_csv_str
```

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

16.12.2.15 get_header_str()

```
procedure csv_module::csv_file::get_header_str
```

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

16.12.2.16 get_integer_column()

```
procedure csv_module::csv_file::get_integer_column
```

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

16.12.2.17 get_logical_column()

```
procedure csv_module::csv_file::get_logical_column
```

Definition at line 109 of file [csv_module.F90](#).

16.12.2.18 get_real_column()

```
procedure csv_module::csv_file::get_real_column
```

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

16.12.2.19 initialize()

```
procedure, public csv_module::csv_file::initialize
```

Definition at line 85 of file [csv_module.F90](#).

16.12.2.20 next_row()

```
procedure, public csv_module::csv_file::next_row
```

Definition at line 122 of file [csv_module.F90](#).

16.12.2.21 open()

```
procedure, public csv_module::csv_file::open
```

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

16.12.2.22 read()

```
procedure, public csv_module::csv_file::read
```

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

16.12.2.23 read_line_from_file()

```
procedure csv_module::csv_file::read_line_from_file
```

Definition at line 126 of file [csv_module.F90](#).

16.12.2.24 tokenize()

```
procedure csv_module::csv_file::tokenize
```

Definition at line 125 of file [csv_module.F90](#).

16.12.2.25 variable_types()

```
procedure, public csv_module::csv_file::variable_types
```

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

16.12.3 Member Data Documentation

16.12.3.1 chunk_size

```
integer csv_module::csv_file::chunk_size = 100
```

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

16.12.3.2 csv_data

```
type(csv_string), dimension(:, :), allocatable csv_module::csv_file::csv_data
```

Definition at line 63 of file [csv_module.F90](#).

16.12.3.3 delimiter

```
character(len=1) csv_module::csv_file::delimiter = ','
```

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

16.12.3.4 enclose_all_in_quotes

```
logical csv_module::csv_file::enclose_all_in_quotes = .false.
```

Definition at line 70 of file [csv_module.F90](#).

16.12.3.5 enclose_strings_in_quotes

```
logical csv_module::csv_file::enclose_strings_in_quotes = .true.
```

Definition at line 68 of file [csv_module.F90](#).

16.12.3.6 header

```
type(csv_string), dimension(:), allocatable csv_module::csv_file::header
```

Definition at line 62 of file [csv_module.F90](#).

16.12.3.7 icol

```
integer csv_module::csv_file::icol = 0
```

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

16.12.3.8 iunit

```
integer csv_module::csv_file::iunit = LUN_BTRK
```

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

16.12.3.9 logical_false_string

```
character(len=1) csv_module::csv_file::logical_false_string = 'F'
```

Definition at line 76 of file [csv_module.F90](#).

16.12.3.10 logical_true_string

```
character(len=1) csv_module::csv_file::logical_true_string = 'T'
```

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

16.12.3.11 n_cols

```
integer, public csv_module::csv_file::n_cols = 0
```

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

16.12.3.12 n_rows

```
integer, public csv_module::csv_file::n_rows = 0
```

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

16.12.3.13 quote

```
character(len=1) csv_module::csv_file::quote = ''''
```

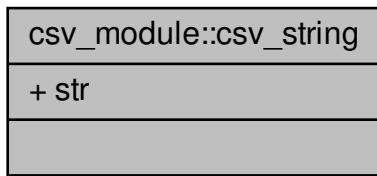
Definition at line 55 of file [csv_module.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[csv_module.F90](#)

16.13 csv_module::csv_string Type Reference

Collaboration diagram for csv_module::csv_string:



Public Attributes

- character(len=::), allocatable str

16.13.1 Detailed Description

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

16.13.2 Member Data Documentation

16.13.2.1 str

```
character(len=:) , allocatable csv_module::csv_string::str
```

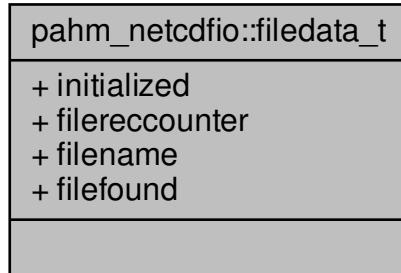
Definition at line 42 of file [csv_module.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[csv_module.F90](#)

16.14 pahm_ncdfio::filedata_t Type Reference

Collaboration diagram for pahm_ncdfio::filedata_t:



Public Attributes

- logical **initialized** = .FALSE.
- integer **filereccounter** = 0
- character(len=fnamelen) **filename**
- logical **filefound** = .FALSE.

16.14.1 Detailed Description

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

16.14.2 Member Data Documentation

16.14.2.1 filefound

```
logical pahm_ncdfio::filedata_t::filefound = .FALSE.
```

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

16.14.2.2 filename

```
character(len=fnamelen) pahm_ncdfio::filedata_t::filename
```

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

16.14.2.3 filereccounter

```
integer pahm_ncdfio::filedata_t::filereccounter = 0
```

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

16.14.2.4 initialized

```
logical pahm_ncdfio::filedata_t::initialized = .FALSE.
```

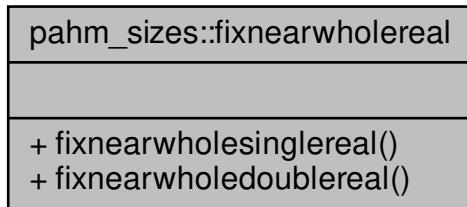
Definition at line 40 of file [netcdfio.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[netcdfio.F90](#)

16.15 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.

16.15.1 Detailed Description

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

16.15.2 Member Function/Subroutine Documentation

16.15.2.1 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](#).

16.15.2.2 fixnearholesinglereal()

```
real(sp) function pahm_sizes::fixnearwholereal::fixnearholesinglereal (
    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
```

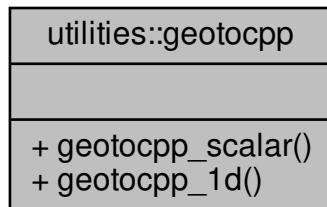
Definition at line 291 of file [sizes.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sizes.F90](#)

16.16 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.

16.16.1 Detailed Description

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

16.16.2 Member Function/Subroutine Documentation

16.16.2.1 [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](#).

16.16.2.2 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](#).

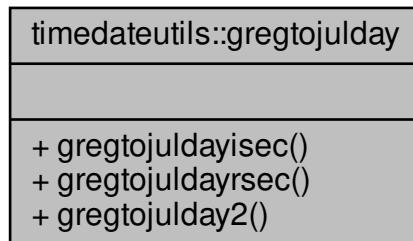
References [pahm_global::deg2rad](#), and [pahm_global::rearth](#).

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

- /home/takis/CSDL/parwinds-doc/src/[utilities.F90](#)

16.17 `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.

16.17.1 Detailed Description

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

16.17.2 Member Function/Subroutine Documentation

16.17.2.1 `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 usefull 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

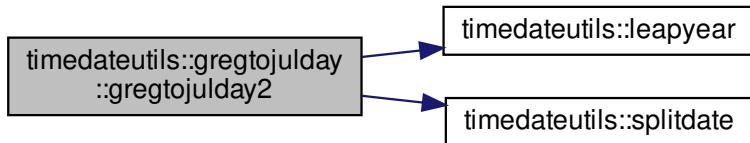
Note

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

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

References `timedateutils::firstgregdate`, `timedateutils::leapyear()`, `timedateutils::mdjoffset`, `timedateutils::splitdate()`, and `timedateutils::usemodjulday`.

Here is the call graph for this function:



16.17.2.2 `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	<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>iSec</i>	iSec The second of the minute (ss, integer, 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

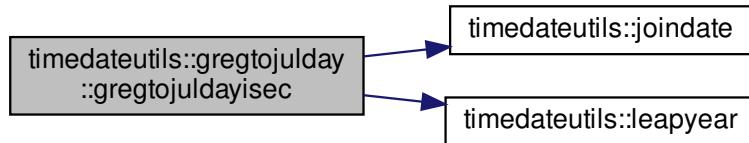
Note

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

Definition at line 536 of file [timedateutils.F90](#).

References [timedateutils::firstgregdate](#), [pahm_sizes::hp](#), [timedateutils::joindate\(\)](#), [timedateutils::leapyear\(\)](#), [timedateutils::mdoffset](#), [pahm_sizes::rmissv](#), and [timedateutils::usemodjulday](#).

Here is the call graph for this function:



16.17.2.3 `gregtojuldaysec()`

```
real(sz) function timedateutils::gregtojulday::gregtojuldaysec (
    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 (-9999999.0) 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.
Generated by Doxygen		

Returns

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

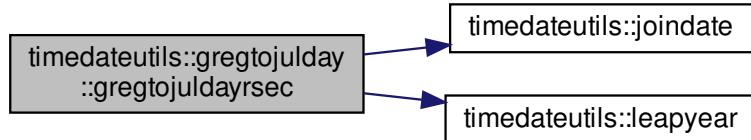
Note

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](#)

16.18 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`

16.18.1 Detailed Description

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

16.18.2 Member Data Documentation

16.18.2.1 basin

```
character(len=2), dimension(:), allocatable parwind::hollanddata_t::basin
```

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

16.18.2.2 casttime

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::casttime
```

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

16.18.2.3 casttype

```
character(len=4), dimension(:), allocatable parwind::hollanddata_t::casttype
```

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

16.18.2.4 cprdt

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::cprdt
```

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

16.18.2.5 cpress

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::cpress
```

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

16.18.2.6 day

```
integer, dimension(:), allocatable parwind::hollanddata_t::day
```

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

16.18.2.7 dtg

```
character(len=10), dimension(:), allocatable parwind::hollanddata_t::dtg
```

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

16.18.2.8 fcstinc

```
integer, dimension(:), allocatable parwind::hollanddata_t::fcstinc
```

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

16.18.2.9 filename

```
character(len=fnamelen) parwind::hollanddata_t::filename
```

Definition at line 125 of file [parwind.F90](#).

16.18.2.10 hour

```
integer, dimension(:), allocatable parwind::hollanddata_t::hour
```

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

16.18.2.11 icpress

```
integer, dimension(:), allocatable parwind::hollanddata_t::icpress
```

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

16.18.2.12 ilat

```
integer, dimension(:), allocatable parwind::hollanddata_t::ilat
```

Definition at line 138 of file [parwind.F90](#).

16.18.2.13 ilon

```
integer, dimension(:), allocatable parwind::hollanddata_t::ilon
```

Definition at line 138 of file [parwind.F90](#).

16.18.2.14 irmw

```
integer, dimension(:), allocatable parwind::hollanddata_t::irmw
```

Definition at line 150 of file [parwind.F90](#).

16.18.2.15 irrp

```
integer, dimension(:), allocatable parwind::hollanddata_t::irrp
```

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

16.18.2.16 ispeed

```
integer, dimension(:), allocatable parwind::hollanddata_t::ispeed
```

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

16.18.2.17 lat

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::lat
```

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

16.18.2.18 loaded

```
logical parwind::hollanddata_t::loaded = .FALSE.
```

Definition at line 127 of file [parwind.F90](#).

16.18.2.19 lon

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::lon
```

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

16.18.2.20 month

```
integer, dimension(:), allocatable parwind::hollanddata_t::month
```

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

16.18.2.21 numrec

```
integer parwind::hollanddata_t::numrec
```

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

16.18.2.22 rmw

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::rmw
```

Definition at line 151 of file [parwind.F90](#).

16.18.2.23 rrp

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::rrp
```

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

16.18.2.24 speed

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::speed
```

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

16.18.2.25 stormnumber

```
integer, dimension(:), allocatable parwind::hollanddata_t::stormnumber
```

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

16.18.2.26 thisstorm

```
character(len=10) parwind::hollanddata_t::thisstorm
```

Definition at line 126 of file [parwind.F90](#).

16.18.2.27 trvx

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::trvx
```

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

16.18.2.28 trvy

```
real(sz), dimension(:), allocatable parwind::hollanddata_t::trvy
```

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

16.18.2.29 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](#)

16.19 sortutils::indexx Interface Reference

Collaboration diagram for sortutils::indexx:

sortutils::indexx
+ indexxint() + indexxint8() + indexxstring() + indexxsingle() + indexxdouble()

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.

16.19.1 Detailed Description

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

16.19.2 Member Function/Subroutine Documentation

16.19.2.1 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)

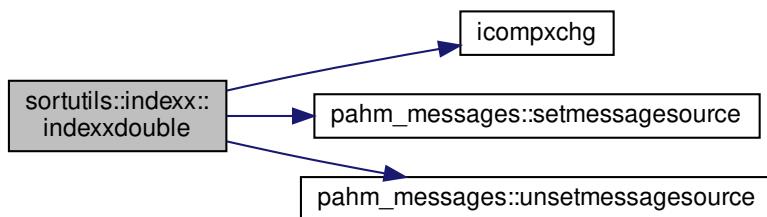
Note

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:



16.19.2.2 indexxit()

```
subroutine sortutils::indexx::indexxit (
    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)

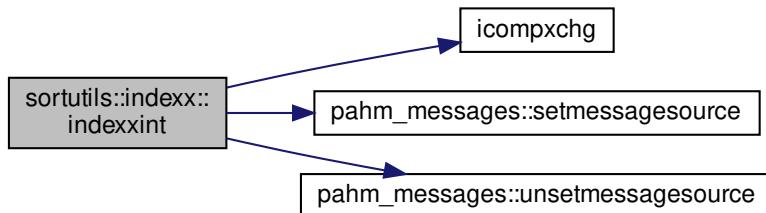
Note

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:



16.19.2.3 indexxit8()

```
subroutine sortutils::indexx::indexxit8 (
    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)

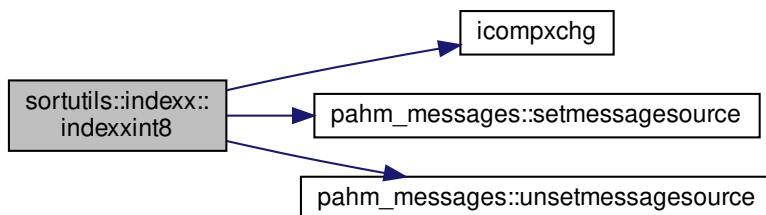
Note

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:



16.19.2.4 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)

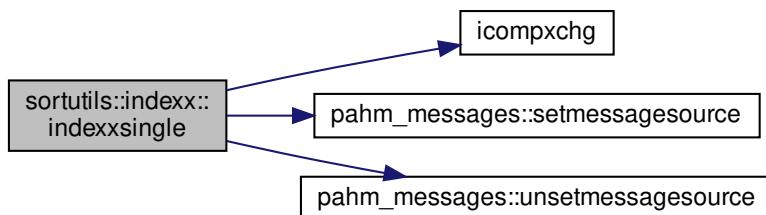
Note

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:



16.19.2.5 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

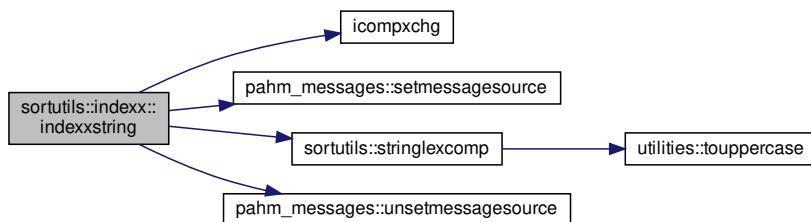
Parameters

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:

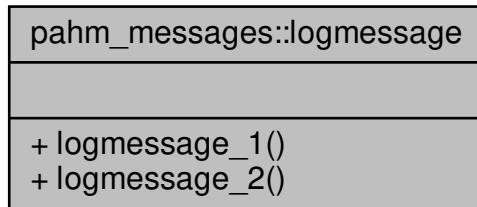


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

16.20 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)

16.20.1 Detailed Description

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

16.20.2 Member Function/Subroutine Documentation

16.20.2.1 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	<i>message</i>	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](#).

16.20.2.2 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](#).

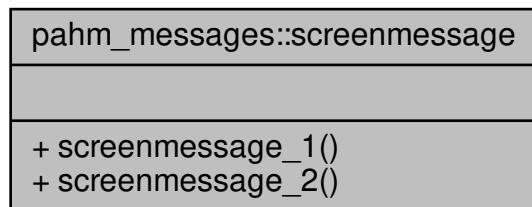
References [pahm_messages::logfileopened](#), [pahm_messages::loginitcalled](#), [pahm_messages::loglevelname](#)s, [pahm_global::lun_log](#), [pahm_messages::messagesources](#), and [pahm_messages::sourcenumber](#).

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

- /home/takis/CSDL/parwinds-doc/src/[messages.F90](#)

16.21 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)

16.21.1 Detailed Description

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

16.21.2 Member Function/Subroutine Documentation

16.21.2.1 [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](#).

16.21.2.2 [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](#).

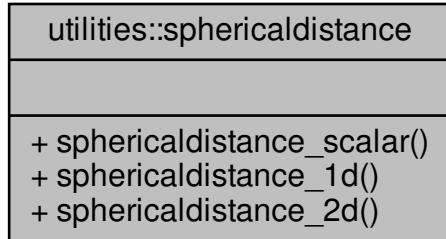
References `pahm_messages::loginitcalled`, `pahm_messages::loglevelname`, `pahm_global::lun_screen`, `pahm_messages::messagesources`, `pahm_messages::nscreen`, and `pahm_messages::sourcenumber`.

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

- [/home/takis/CSDL/parwinds-doc/src/messages.F90](#)

16.22 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.

16.22.1 Detailed Description

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

16.22.2 Member Function/Subroutine Documentation

16.22.2.1 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](#).

16.22.2.2 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 \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, 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](#).

16.22.2.3 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 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](#).

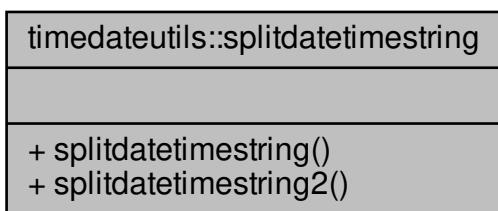
References [pahm_global::deg2rad](#), and [pahm_global::rearth](#).

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

- /home/takis/CSDL/parwinds-doc/src/[utilities.F90](#)

16.23 `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.

16.23.1 Detailed Description

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

16.23.2 Constructor & Destructor Documentation

16.23.2.1 `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)".

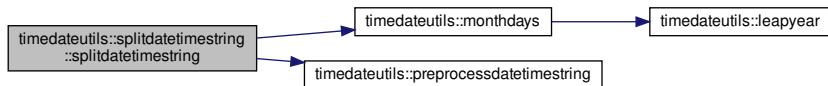
Parameters

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



16.23.3 Member Function/Subroutine Documentation

16.23.3.1 `splitdatetimestring2()`

```

subroutine timedateutils::splitdatetimestring::splitdatetimestring2 (
    character(len=*) , intent(in)  inDateTime,
    integer, intent(out) iDate,
    integer, intent(out) iTime )

```

Splits a date string into two components.

This subroutine splits the string `inDate` (YYYYMMDDhhmmss) in two integers that is, "iDate (YYYYMMDD)" and "iTime (hhmmss)".

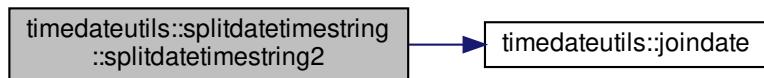
Parameters

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

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

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

Here is the call graph for this function:

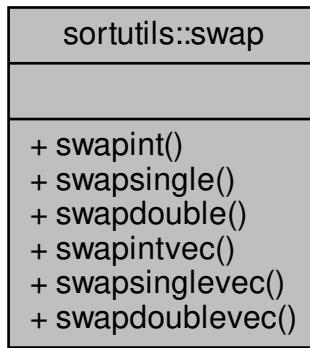


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[timedateutils.F90](#)

16.24 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)
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).

16.24.1 Detailed Description

Definition at line 50 of file [sortutils.F90](#).

16.24.2 Member Function/Subroutine Documentation

16.24.2.1 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](#).

16.24.2.2 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](#).

16.24.2.3 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](#).

16.24.2.4 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](#).

16.24.2.5 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](#).

16.24.2.6 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

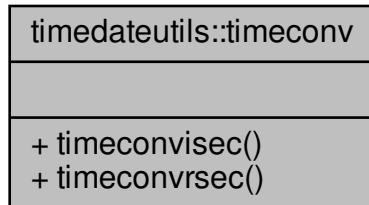
Definition at line 1709 of file [sortutils.F90](#).

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

- /home/takis/CSDL/parwinds-doc/src/[sortutils.F90](#)

16.25 `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.

16.25.1 Detailed Description

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

16.25.2 Member Function/Subroutine Documentation

16.25.2.1 `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.

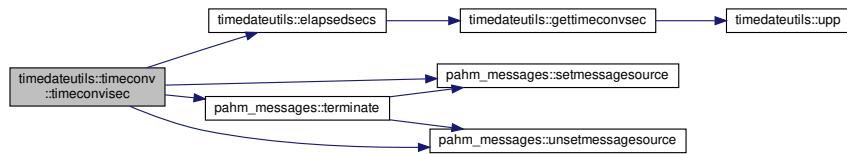
Parameters

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:



16.25.2.2 `timeconvrsec()`

```

subroutine timedateutils::timeconv::timeconvrsec (
    integer, intent(in) iYear,
    integer, intent(in) iMonth,
    integer, intent(in) iDay,
    integer, intent(in) iHour,
    integer, intent(in) iMin,
    real(sz), intent(in) rSec,
    real(sz), intent(out) timeSec )

```

Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.

The reference date is defined by the global variables: `refYear`, `refMonth`, `refDay`, `refHour`, `refMin` and `refSec`. It uses `GregToJulDay` and `ElapsedSecs` functions to calculate the elapsed time from the reference date. Similar to `TimeConvISEC` but seconds are entered as real numbers to allow for fractions of a second.

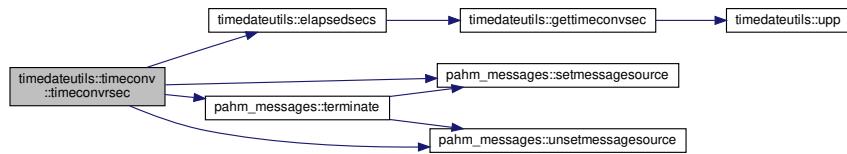
Parameters

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](#).

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:

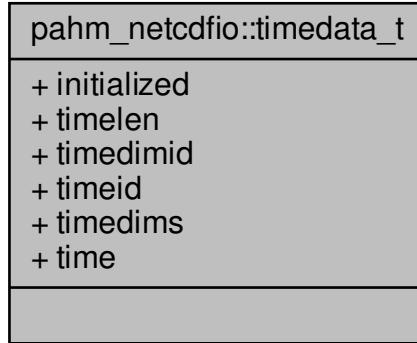


The documentation for this interface was generated from the following file:

- /home/takis/CSDL/parwinds-doc/src/[timedateutils.F90](#)

16.26 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`

16.26.1 Detailed Description

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

16.26.2 Member Data Documentation

16.26.2.1 initialized

```
logical pahm_netcdfio::timedata_t::initialized = .FALSE.
```

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

16.26.2.2 time

```
real(sz), dimension(:), allocatable pahm_ncdfio::timedata_t::time
```

Definition at line 52 of file [netcdfio.F90](#).

16.26.2.3 timedimid

```
integer pahm_ncdfio::timedata_t::timedimid
```

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

16.26.2.4 timedims

```
integer, dimension(1) pahm_ncdfio::timedata_t::timedims
```

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

16.26.2.5 timeid

```
integer pahm_ncdfio::timedata_t::timeid
```

Definition at line 50 of file [netcdfio.F90](#).

16.26.2.6 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](#)

Chapter 17

File Documentation

17.1 user-guide/abstract.md File Reference

17.2 user-guide/application.md File Reference

17.3 user-guide/code.md File Reference

17.4 user-guide/credits.md File Reference

17.5 user-guide/deliverables.md File Reference

17.6 user-guide/evaluation.md File Reference

17.7 user-guide/features.md File Reference

17.8 user-guide/figures.md File Reference

17.9 user-guide/glossary.md File Reference

17.10 user-guide/intro.md File Reference

17.11 user-guide/models.md File Reference

17.12 user-guide/pahm_manual.md File Reference

17.13 user-guide/references-dox.md File Reference

17.14 user-guide/tables.md File Reference

17.15 /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

17.15.1 Detailed Description

For reading and writing CSV files.

Author

Jacob Williams

Copyright

License BSD

Definition in file `csv_module.F90`.

17.16 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_btrk1  

00020 USE pahm_messages  

00021 USE utilities, ONLY : openfileforread, tolowercase  

00022 USE csv_utilities  

00023 USE csv_parameters  

00024  

00025 implicit none  

00026  

00027 PRIVATE  

00028  

00029 ! the different types of variables that can be in a CSV file.  

00030 INTEGER, PARAMETER, PUBLIC :: csv_type_string = 1 ! 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
00140
00141
00142
00143
00144
00145
00146   subroutine initialize_csv_file(me,quote,delimiter,&
00147                                     enclose_strings_in_quotes,&
00148                                     enclose_all_in_quotes,&
00149                                     logical_true_string,&
00150                                     logical_false_string,&
00151                                     chunk_size)
00152
00153   implicit none
00154
00155   class(csv_file),intent(out) :: me
00156   character(len=1),intent(in),optional :: quote           ! note: can only be one character
00157                                         ! (Default is '')
00158   character(len=1),intent(in),optional :: delimiter        ! note: can only be one character
00159                                         ! (Default is ',')
00160   logical,intent(in),optional :: enclose_strings_in_quotes ! if true, all string cells
00161                                         ! will be enclosed in quotes.
00162                                         ! (Default is True)
00163   logical,intent(in),optional :: enclose_all_in_quotes    ! if true, *all* cells will
00164                                         ! be enclosed in quotes.
00165                                         ! (Default is False)
00166   character(len=1),intent(in),optional :: logical_true_string ! when writing a logical 'true'
00167                                         ! value to a CSV file, this
00168                                         ! is the string to use
00169                                         ! (default is 'T')
00170   character(len=1),intent(in),optional :: logical_false_string ! when writing a logical 'false'
00171                                         ! value to a CSV file, this
00172                                         ! is the string to use
00173                                         ! (default is 'F')
00174   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      character(len=*),intent(in) :: filename
00239      logical,intent(out) :: status_ok
00240      integer,intent(in),optional :: header_row
00241      integer,dimension(:),intent(in),optional :: skip_rows ! rows to skip
00242
00243      class(csv_file),intent(inout) :: me
00244      type(csv_string),dimension(:),allocatable :: row_data ! a tokenized row
00245      type(csv_string) :: empty_data
00246      integer,dimension(:),allocatable :: rows_to_skip ! the actual rows to skip
00247      character(len=:),allocatable :: line ! a line from the file
00248      integer :: i ! counter
00249      integer :: j ! counter
00250      integer :: irow ! row counter
00251      integer :: n_rows_in_file ! number of lines in the file
00252      integer :: n_rows ! number of rows in the output data matrix
00253      integer :: n_cols ! number of columns in the file (and output data matrix)
00254      integer :: istat ! open status flag
00255      integer :: line_n_cols ! number of columns in the line (not necessarily equal to n_cols)
00256      integer :: iunit ! open file unit
00257      logical :: arrays_allocated ! if the arrays in the
00258      ! class have been allocated
00259      integer :: iheader ! row number of header row
00260      ! (0 if no header specified)
00261      character(len=1) :: tmp ! for skipping a row
00262
00263      empty_data%str = ' '
00264      iunit = lun_btrk
00265
00266      CALL setmessagesource("read_csv_file")
00267
00268      call me%destroy()
00269      arrays_allocated = .false.
00270
00271      CALL openfileforread(iunit, trim(adjustl(filename)), istat)
00272
00273      IF (istat /= 0) THEN
00274          WRITE(scratchmessage, '(a)') 'Error opening the file: ' // trim(adjustl(filename))
00275          CALL allmessage(error, scratchmessage)
00276
00277
00278
00279
00280
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 !-----
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%quote
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%quote
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 !-----
00658 !
00659 subroutine add_vector(me, val, int_fmt, real_fmt, trim_str)
00660
00661     implicit none
00662
00663     class(csv_file),intent(inout) :: me
00664     class(*),dimension(:),intent(in) :: val
00665     character(len=*),intent(in),optional :: int_fmt
00666     character(len=*),intent(in),optional :: real_fmt
00667
00668     logical,intent(in),optional :: trim_str
00669
00670     integer :: i ! counter
00671
00672     do i=1,size(val)
00673
00674 #if defined __GFORTRAN__
00675     ! This is a stupid workaround for gfortran bugs (tested with 7.2.0)
00676     select type (val)
00677         type is (character(len=*))
00678             call me%add(val(i),int_fmt,real_fmt,trim_str)
00679         class default
00680             call me%add(val(i),int_fmt,real_fmt,trim_str)
00681     end select
00682 #else
00683     call me%add(val(i),int_fmt,real_fmt,trim_str)
00684 #endif
00685
00686     end do
00687
00688 end subroutine add_vector
00689 !=====

```

```

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 !-----
00833
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 !-----
00886 !-----
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
00921     else
00922         scratchmessage = 'Error: class has not been initialized'
00923         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=*),dimension(4),parameter :: true_str = ['1    ',&
01040                                         't    ',&
01041                                         'true ',&
01042                                         '.true.']
01043   character(len=*),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 ! S U B R O U T I N E   V A R I A B L E _ T Y P E S
01065 !-----
01067 !-----
01088 subroutine variable_types(me, itypes, status_ok)
01089
01090     implicit none
01091
01092     class(csv_file),intent(inout) :: me
01093     integer,dimension(:),allocatable,intent(out) :: itypes
01094     logical,intent(out) :: status_ok
01095
01096     integer :: i ! counter
01097
01098     CALL setmessagesource("variable_types")
01099
01100     if (allocated(me%csv_data)) then
01101         allocate(itypes(me%n_cols))
01102         do i=1,me%n_cols
01103             call infer_variable_type(me%csv_data(1,i)%str,itypes(i))
01104         end do
01105     else
01106         scratchmessage = 'Error: class has not been initialized'
01107         CALL allmessage(error, scratchmessage)
01108         status_ok = .false.
01109     end if
01110
01111     CALL unsetmessagesource()
01112
01113 end subroutine variable_types
01114 !=====
01115
01116 !-----
01117 ! S U B R O U T I N E   I N F E R _ V A R I A B L E _ T Y P E
01118 !-----
01141 !-----
01142 subroutine infer_variable_type(str, itype)
01143
01144     implicit none
01145
01146     character(len=*),intent(in) :: str
01147     integer,intent(out) :: itype
01148
01149     real(wp)      :: rval      ! a real value
01150     integer(ip)   :: ival      ! an integer value
01151     logical       :: lval      ! a logical value
01152     logical       :: status_ok ! status flag
01153
01154     call to_integer(str,ival,status_ok)
01155     if (status_ok) then
01156         itype = csv_type_integer
01157         return
01158     end if
01159
01160     call to_real(str,rval,status_ok)
01161     if (status_ok) then
01162         itype = csv_type_double
01163         return
01164     end if
01165
01166     call to_logical(str,lval,status_ok)
01167     if (status_ok) then
01168         itype = csv_type_logical
01169         return
01170     end if

```

```

01171      ! default is string:
01172      itype = 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 ! -----
01199 !
01200 subroutine csv_get_value(me, row, col, val, status_ok)
01201
01202     implicit none
01203
01204     class(csv_file),intent(inout) :: me
01205     integer,intent(in)    :: row
01206     integer,intent(in)    :: col
01207     class(*),intent(out)  :: val
01208     logical,intent(out)   :: status_ok
01209
01210     select type (val)
01211     type is (integer(ip))
01212         call to_integer(me%csv_data(row,col)%str,val,status_ok)
01213     type is (real(wp))
01214         call to_real(me%csv_data(row,col)%str,val,status_ok)
01215     type is (logical)
01216         call to_logical(me%csv_data(row,col)%str,val,status_ok)
01217     type is (character(len=*))
01218         status_ok = .true.
01219         if (allocated(me%csv_data(row,col)%str)) then
01220             val = me%csv_data(row,col)%str
01221         else
01222             val = ""
01223         end if
01224     type is (csv_string)
01225         status_ok = .true.
01226         val = me%csv_data(row,col)
01227     class default
01228         status_ok = .false.
01229     end select
01230
01231     end subroutine csv_get_value
01232  =====
01233
01234 ! -----
01235 ! S U B R O U T I N E   G E T _ C O L U M N
01236 ! -----
01237 ! -----
01238 !
01239 !
01240 subroutine get_column(me, icol, r, status_ok)
01241
01242     implicit none
01243
01244     class(csv_file),intent(inout) :: me
01245     integer,intent(in)    :: icol
01246     class(*),dimension(:),intent(out) :: r
01247     logical,intent(out)   :: status_ok
01248
01249     integer :: i ! counter
01250 #if defined __GFORTRAN__
01251     character(len=:),allocatable :: tmp ! for gfortran workaround
01252 #endif
01253
01254     CALL setmessagesource("get_column")
01255
01256     ! we know the data is allocated, since that
01257     ! was checked by the calling routines.
01258
01259     if (me%n_cols>=icol .and. icol>0) then
01260
01261         do i=1,me%n_rows ! row loop
01262
01263 #if defined __GFORTRAN__
01264         ! the following is a workaround for gfortran bugs:
01265         select type (r)
01266         type is (character(len=*))
01267             tmp = repeat(' ',len(r)) ! size the string
01268             call me%csv_get_value(i,icol,tmp,status_ok)
01269             r(i) = tmp
01270         class default
01271             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 !-----
01330 ! S U B R O U T I N E   G E T _ R E A L _ C O L U M N
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 !-----
01374 ! 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
01375 !-----
01391 !-----
01392 subroutine get_integer_column(me,ic平,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 !-----SUBROUTINE TOKENIZE_CSV_LINE
01550 !-----SUBROUTINE TOKENIZE_CSV_LINE
01551 !-----
01552 !-----
01553 subroutine tokenize_csv_line(me, line, cells)
01554 implicit none
01555
01556 class(csv_file),intent(inout) :: me
01557 character(len=*),intent(in) :: line
01558 type(csv_string),dimension(:),allocatable,intent(out) :: cells
01559
01560 integer :: i ! counter
01561 character(len=:),allocatable :: tmp ! a temp string with whitespace removed
01562 integer :: n ! length of compressed string
01563
01564 call split(line,me%delimiter,me%chunk_size,cells)
01565
01566 ! remove quotes if present:
01567 do i = 1, size(cells)
01568
01569     ! remove whitespace from the string:
01570     tmp = trim(adjustl(cells(i)%str))
01571     n = len(tmp)
01572
01573     if (n>1) then
01574         ! if the first and last non-blank character is
01575         ! a quote, then remove them and replace with what
01576         ! is inside the quotes. Otherwise, leave it as is.
01577         if (tmp(1:1)==me%quote .and. tmp(n:n)==me%quote) then
01578             if (n>2) then
01579                 cells(i)%str = tmp(2:n-1) ! remove the quotes
01580             else
01581                 cells(i)%str = " ! empty string
01582             end if
01583         end if
01584     end if
01585
01586 end do
01587
01588 end subroutine tokenize_csv_line
01589 !=====
01590 !-----
01591 !-----FUNCTION NUMBER_OF_LINES_IN_FILE
01592 !-----
01593 function number_of_lines_in_file(iunit) result(n_lines)
01594 implicit none
01595
01596 integer,intent(in) :: iunit ! the file unit number
01597                           ! (assumed to be open)
01598 integer :: n_lines ! the number of lines in the file
01599
01600 character(len=1) :: tmp
01601 integer :: istat
01602
01603 rewind(iunit)
01604 n_lines = 0
01605 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

```

17.17 /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 = '\(I256\)'](#)

17.17.1 Detailed Description

Various parameters.

Author

Jacob Williams

Copyright

License BSD

Definition in file [csv_parameters.F90](#).

17.18 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

```

17.19 /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

17.19.1 Detailed Description

Utility routines.

Author

Jacob Williams

Copyright

License BSD

Definition in file [csv_utilities.F90](#).

17.19.2 Function/Subroutine Documentation

17.19.2.1 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:



17.19.2.2 quicksort()

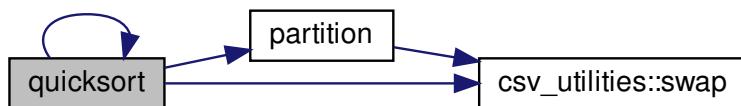
```
recursive subroutine sort_descending::quicksort (
    integer, intent(in) ilow,
    integer, intent(in) ihigh ) [private]
```

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

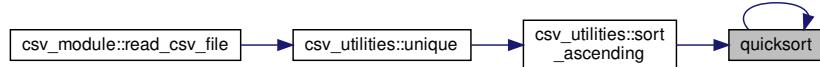
References [csv_utilities::max_size_for_insertion_sort](#), [partition\(\)](#), [quicksort\(\)](#), and [csv_utilities::swap\(\)](#).

Referenced by [quicksort\(\)](#), and [csv_utilities::sort_descending\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



17.20 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 !-----+
00014 !
00015
00016 MODULE csv_utilities
00017
00018 USE pahm_sizes, ONLY : wp, ip
00019 USE csv_parameters
00020
00021 PRIVATE
00022
00023 INTEGER, PARAMETER :: max_size_for_insertion_sort = 20 ! max size for using insertion sort.
00024
00025 PUBLIC :: unique
00026 PUBLIC :: expand_vector
00027 PUBLIC :: sort_descending
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 !
00054 !-----
00055 pure subroutine expand_vector(vec, n, chunk_size, val, finished)
00056
00057 implicit none
00058
00059 integer,dimension(:),allocatable,intent(inout) :: vec
00060 integer,intent(inout) :: n           ! counter for last element added to 'vec'.
00061                           ! must be initialized to 'size(vec)'
00062                           ! (or 0 if not allocated) before first call
00063 integer,intent(in)      :: chunk_size ! allocate 'vec' in blocks of this size (>0)
00064 integer,intent(in),optional :: val      ! the value to add to 'vec'
00065 logical,intent(in),optional :: finished ! set to true to return 'vec'
00066                           ! as its correct size ('n')
00067
00068 integer,dimension(:),allocatable :: tmp ! temporary array
00069
00070 if (present(val)) then
00071     if (allocated(vec)) then
00072         if (n==size(vec)) then
00073             ! have to add another chunk:
00074             allocate(tmp(size(vec)+chunk_size))
00075             tmp(1:size(vec)) = vec
00076             call move_alloc(tmp,vec)
00077         end if
00078         n = n + 1
00079     else
00080         ! the first element:
00081         allocate(vec(chunk_size))
00082         n = 1
00083     end if
00084     vec(n) = val
00085 end if
00086
00087 if (present(finished)) then
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 !-----
00101 ! F U N C T I O N   U N I Q U E
00102 !
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 !-----
00154 !-----
00167 subroutine sortAscending(ivec)
00168
00169 implicit none
00170 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 sort_descending
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(i1,i2)
00260
00261   implicit none
00262
00263   integer,intent(inout) :: i1
00264   integer,intent(inout) :: i2
00265
00266   integer :: tmp
00267
00268   tmp = i1
00269   i1 = i2
00270   i2 = tmp
00271
00272   end subroutine swap
00273 !=====
00274
00275 END MODULE csv_utilities

```

17.21 /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](#)

17.21.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [driver_mod.F90](#).

17.22 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(argcnt, argcmdline)  

00057  

00058         SELECT CASE(trim(argcmdline))  

00059             CASE("-V", "-v", "--V", "--v", "--version")  

00060                 CALL programversion  

00061                 stop  

00062  

00063             CASE("-H", "-h", "--H", "--h", "--help")  

00064                 CALL programhelp  

00065                 stop  

00066  

00067             CASE DEFAULT  

00068                 ! Do nothing  

00069             END SELECT  

00070         END DO  

00071         ! This is the first argument if not "-v/-h" were supplied.  

00072         ! It is assumed that this argument is the filename of the user control file.  

00073         CALL getarg(1, argcmdline)  

00074         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 !-----
00085 !-----
00086 !-----
00087 !----- SUBROUTINE pahm_init()
00088
00089     USE pahm_global, ONLY : noutdt
00090     USE pahm_mesh, ONLY : readmesh
00091     USE parwind, ONLY : readbesttrackfile, readcsvbesttrackfile
00092
00093     ! Initialize the logging system, needs to be called first
00094     CALL initlogging()
00095
00096     CALL setmessagesource("PaHM_Init")
00097
00098     ! Get possible command line arguments
00099     CALL getprogramcmdlargs()
00100
00101     ! Read the mesh/grid of the domain or the generic mesh/grid input file
00102     CALL readmesh()
00103
00104     ! Read all track files and save the data into the array of the best track structures
00105     ! for subsequent access by the P-W models in the program
00106     !CALL ReadBestTrackFile()
00107     CALL readcsvbesttrackfile()
00108
00109     cnttimebegin = 1
00110     cnttimeend   = noutdt
00111
00112     CALL unsetmessagesource()
00113
00114 END SUBROUTINE pahm_init
00115
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 !-----
00120 !----- SUBROUTINE pahm_run(nTimeSTP)
00121
00122     USE pahm_global, ONLY : modeltype
00123     USE parwind
00124     USE pahm_netcdfio
00125
00126     IMPLICIT NONE
00127
00128     INTEGER, INTENT(IN), OPTIONAL :: nTimeSTP
00129
00130     INTEGER :: iCnT
00131
00132     CALL setmessagesource("PaHM_Run")
00133
00134     IF (PRESENT(ntimestp)) THEN
00135         cnttimeend = cnttimebegin + ntimestp - 1
00136     END IF
00137
00138     SELECT CASE (modeltype)
00139     CASE (1)
00140         DO icnt = cnttimebegin, cnttimeend
00141             CALL gethollandfields(icnt)
00142
00143             IF (outfilenamespecified) THEN
00144                 ! Create the output NetCDF file and fill it with the static data only
00145                 ! Initialize all variables. This subroutine is called just once
00146                 CALL initadcircnetcdfoutfile(outfilename)
00147
00148                 CALL writenetcdfrecord(outfilename, icnt)
00149             END IF
00150         END DO
00151
00152     CASE (10)
00153         DO icnt = cnttimebegin, cnttimeend
00154             CALL getgahmfields(icnt)
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173

```

```

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(ntimestep)) THEN
00191     cntimebegin = cntimeend + 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

```

17.23 /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::logfile`name = 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_NCNAME_PRES
- character(len=20) pahm_global::ncvarnam_wndx = DEF_NCNAME_WNDX
- character(len=20) pahm_global::ncvarnam_wndy = DEF_NCNAME_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::wpres
- real(sz), dimension(:), allocatable pahm_global::times
- character(19), dimension(:), allocatable pahm_global::datestimes

17.23.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [global.F90](#).

17.24 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_out1 = 26         ! I/O unit for the output files
00034 !#####
00035 !###    END::: LUN NUMBERS FOR I/O OPERATIONS
00036 !#####
00037  

00038  

00039 !#####
00040 !###    BEG::: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
00041 !#####
00042 REAL(sz), PARAMETER :: defv_gravity = 9.80665_sz      ! Default (standard) gravitational acceleration
               (m/s^2)
00043 REAL(sz), PARAMETER :: defv_atmpress = 1013.25_sz     ! Default (standard) atmospheric pressure (mb)
00044  

00045 REAL(sz), PARAMETER :: defv_rhoair = 1.1478_sz       ! Default (standard) density of air at STP
               (kg/m^3)
00046  

00047  

00048 ! Water density is used in the code to convert the pressure to units of mH2O
00049 REAL(sz), PARAMETER :: defv_rhewater = 1000.0000      ! Default density of fresh water (kg/m^3)
00050  

00051 !--- FRESH WATER
00052 !      999.8900 ( 0 deg C)
00053 !      1000.0000 ( 4 deg C)
00054 !      999.7025 (10 deg C)
00055 !      999.1026 (15 deg C)
00056 !      998.2072 (20 deg C)
00057 !      997.0476 (25 deg C)
00058 !      995.6495 (30 deg C)
00059 !      994.0333 (35 deg C)
00060 !      992.2164 (40 deg C)
00061  

00062 !--- SEA WATER
00063 !      1028.0941 (35% S,  1 deg C)
00064 !      1027.8336 (35% S,  4 deg C)
00065 !      1027.0000 (35% S, 10 deg C)
00066 !      1026.0210 (35% S, 15 deg C)
00067 !      1024.8103 (35% S, 20 deg C)
00068 !      1023.3873 (35% S, 25 deg C)

```

```

00067                                         ! 1021.7694 (35% S, 30 deg C)
00068                                         ! 1019.9000 (35% S, 35 deg C)
00069                                         ! 1018.0000 (35% S, 40 deg C)
00070
00071 ! 1-min to 10-min wind conversion factors
00072 REAL(sz), PARAMETER :: one2ten = 0.8928_sz
00073 REAL(sz), PARAMETER :: ten2one = 1.0_sz / 0.8928_sz
00074
00075 REAL(sz), PARAMETER :: pi = 3.141592653589793_sz
00076 REAL(sz), PARAMETER :: deg2rad = pi / 180.0_sz          ! degrees to radians
00077 REAL(sz), PARAMETER :: rad2deg = 180.0_sz / pi        ! radians to degrees
00078 REAL(sz), PARAMETER :: basee = 2.718281828459045_sz ! mathematical constant e (natural logarithm base)
00079
00080 REAL(sz), PARAMETER :: rearth = 6378206.4_sz         ! radius of earth (m) (Clarke 1866 major spheroid
radius)
00081 REAL(sz), PARAMETER :: nm2m   = 1852.0_sz            ! nautical miles to meters
00082 REAL(sz), PARAMETER :: m2nm   = 1.0_sz / nm2m         ! meters to nautical miles
00083 REAL(sz), PARAMETER :: kt2ms  = nm2m / 3600.0_sz      ! knots to m/s
00084 REAL(sz), PARAMETER :: ms2kt  = 1.0_sz / kt2ms        ! m/s to knots
00085 REAL(sz), PARAMETER :: omega  = 2.0_sz * pi / 86164.2_sz
00086 REAL(sz), PARAMETER :: mb2pa  = 100.0_sz
00087 REAL(sz), PARAMETER :: mb2kpa = 0.1_sz
00088 !#####
00089 !### END:: GLOBAL PARAMETERS AND PHYSICAL CONSTANTS
00090 !#####
00091
00092
00093 !#####
00094 !### BEG :: VARIABLES RELATED TO THE CONTROL FILE
00095 !#####
00096 CHARACTER(LEN=FNAMELEN) :: logfilename = 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 (GAHM)
00205 LOGICAL :: writeparams = .false.
00206 ##### END :: VARIABLES RELATED TO THE CONTROL FILE
00207 #####
00208
00209
00210 ##### GLOBAL DATA ARRAYS
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 :: datetimes
00218 #####
00219 !### END :: GLOBAL DATA ARRAYS
00220 #####
00221

```

```

00222
00223     CONTAINS
00224
00225
00226 !-----
00227 !  F U N C T I O N   A I R   D E N S I T Y
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
00290 END MODULE pahm_global

```

17.25 /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::allocatenodalandelementalarrays \(\)](#)
Allocates memory to mesh arrays.

Variables

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

17.25.1 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](#).

17.26 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 varibile 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  

type: ' // &  

                           ' [ ' // trim(meshfiletype) // ' ]'  

00095                 CALL allmessage(error, scratchmessage)  

00096                 CALL terminate()  

00098  

00099             CASE DEFAULT  

00100                 WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the  

mesh file type: ' // &  

                           ' [ ' // trim(meshfiletype) // ' ]'

```

```

00102         CALL allmessage(error, scratchmessage)
00103         CALL terminate()
00104     END SELECT
00105
00106 !----- SCHISM case
00107 CASE('SCHISM')
00108     SELECT CASE(touppercase(meshfileform))
00109         CASE('ASCII')
00110             CALL readmeshasciifort14()
00111
00112         CASE('NETCDF')
00113             WRITE(scratchmessage, '(a)') 'ReadMesh: NetCDF format is not supported yet for the mesh file
type: ' // &
00114                                         '[' // trim(meshfiletype) // ']'
00115             CALL allmessage(error, scratchmessage)
00116             CALL terminate()
00117
00118         CASE DEFAULT
00119             WRITE(scratchmessage, '(a)') 'ReadMesh: Only ASCII and NetCDF formats are supported for the
mesh file type: ' // &
00120                                         '[' // trim(meshfiletype) // ']'
00121             CALL allmessage(error, scratchmessage)
00122             CALL terminate()
00123     END SELECT
00124
00125 !----- FVCOM case
00126 CASE('FVCOM')
00127     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00128                                         '[' // trim(meshfiletype) // ']'
00129     CALL allmessage(error, scratchmessage)
00130     CALL terminate()
00131
00132 !----- ROMS case
00133 CASE('ROMS')
00134     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00135                                         '[' // trim(meshfiletype) // ']'
00136     CALL allmessage(error, scratchmessage)
00137     CALL terminate()
00138
00139 !----- GENERIC case
00140 CASE('GENERIC')
00141     WRITE(scratchmessage, '(a)') 'ReadMesh: This file type is not yet imlemented: ' // &
00142                                         '[' // trim(meshfiletype) // ']'
00143     CALL allmessage(error, scratchmessage)
00144     CALL terminate()
00145
00146 CASE DEFAULT
00147     WRITE(scratchmessage, '(a)') 'ReadMesh: Invalid mesh file type specified: ' // &
00148                                         '[' // trim(meshfiletype) // ']'
00149     CALL allmessage(error, scratchmessage)
00150     CALL terminate()
00151 END SELECT
00152
00153     CALL unsetmessagesource()
00154
00155 END SUBROUTINE readmesh
00156
00157 !=====
00158 !----- 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
00159 !-----
00160 !----- SUBROUTINE readmeshasciifort14()
00161 !
00162 !----- IMPLICIT NONE
00163
00164 IMPLICIT NONE
00165
00166 INTEGER, PARAMETER :: iUnit = lun_inp          ! LUN for read operations
00167 INTEGER          :: ios                         ! I/O status
00168 CHARACTER(LEN=512) :: fmtStr                   ! String to hold formats for I/O
00169 INTEGER          :: lineNumber                ! Line number currently being read
00170
00171 INTEGER          :: labNodes, numFNodes      ! Label and number of nodal faces for that label
00172 INTEGER          :: iCnt                      ! Counters
00173
00174
00175 CALL setmessagesource("ReadMeshASCIIFort14")
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187

```

```

00188     CALL openfileforread(iunit, trim(meshfilename), ios)
00189
00190     linenum = 1
00191
00192     READ(unit=iunit, fmt='(a80)', err=10, END=20, IOSTAT=ios) agrid
00193     linenum = linenum + 1
00194
00195     CALL logmessage(info, "Reading the mesh file: " // trim(meshfilename))
00196     CALL logmessage(info, "Mesh file comment line: " // trim(agrid))
00197     CALL logmessage(info, "Reading mesh file dimensions and coordinates.")
00198
00199     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) ne, np
00200     linenum = linenum + 1
00201
00202     CALL allocatenodalandelementalarrays()
00203
00204 ! N O D E   T A B L E
00205 DO icnt = 1, np
00206     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, slam(icnt), sfea(icnt), dp(icnt)
00207
00208     ! Check for (invalid longitude, latitude) values.
00209     ! Currently only geographical coordinates are supported.
00210     IF (.NOT. ((slam(icnt) >= -180.0_sz) .AND. (slam(icnt) <= 180.0_sz)) .OR. &
00211         .NOT. ((sfea(icnt) >= -90.0_sz) .AND. (sfea(icnt) <= 90.0_sz))) THEN
00212
00213     fmtstr = "("Input file: " // trim(meshfilename) // '", ", line ", i0,
00214     fmtstr = trim(fmtstr) // ' " contains invalid (lon, lat) values: ", " [, f14.4, ", ", f14.4,
00215     "]" ,
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
00230
00231     ! Check if numFNodes in the line is beyond the value of parameter MAXFACENODES,
00232     ! to avoid out of bounds errors for the array "nm".
00233     IF (numfnodes > maxfacenodes) THEN
00234         fmtstr = "("Input file: " // trim(meshfilename) // '", ", reading line ", i0,
00235         fmtstr = trim(fmtstr) // ' " gave a number of face nodes equal to: ", i0,
00236         fmtstr = trim(fmtstr) // ' ", which is greater than MAXFACENODES")'
00237     WRITE(scratchmessage, trim(fmtstr)) linenum, numfnodes
00238
00239     CALL allmessage(error, scratchmessage)
00240     CLOSE(iunit)
00241     CALL terminate()
00242 ELSE
00243     backspace(unit=iunit)
00244 END IF
00245
00246     READ(unit=iunit, fmt=*, err=10, END=20, IOSTAT=ios) labNodes, nfn(icnt), nm(icnt, 1:nfn(icnt))
00247
00248     linenum = linenum + 1
00249 END DO
00250
00251     CLOSE(iunit)
00252
00253     !PV Need to also check if arrays contain any missing values
00254     IF ((comparereals(slam0, rmissv) == 0) .OR. &
00255         (comparereals(sfea0, rmissv) == 0)) THEN
00256         slam0 = sum(slam, 1) / np
00257         sfea0 = sum(sfea, 1) / np
00258     END IF
00259
00260     CALL geotocpp(sfea, slam, sfea0, slam0, xcslam, ycsfea)
00261
00262     CALL logmessage(info, 'Finished reading mesh file dimensions and coordinates.')
00263
00264     CALL unsetmessagesource()
00265
00266     ismeshok = .true.
00267
00268     RETURN

```

17.27 /home/takis/CSDL/parwinds-doc/src/messages.F90 File Reference

```
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
```

17.27 /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::loglevelname`
- 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.`

17.27.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Note

Adopted from the ADCIRC source code.

Definition in file [messages.F90](#).

17.28 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) :: loglevelname  

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 !-----
00103 !      S U B R O U T I N E      O P E N      L O G      F I L E
00104 !-----
00112 !-----
00113 SUBROUTINE openlogfile()
00114
00115     IMPLICIT NONE
00116
00117     INTEGER :: errorIO ! zero if the file opened successfully
00118
00119     logfileopened = .false.
00120
00121     OPEN(unit=lun_log, file=trim(adjustl(logfilename)), action='WRITE', status='REPLACE', iostat=errorio)
00122
00123     IF (errorio == 0) THEN
00124         logfileopened = .true.
00125     ELSE
00126         WRITE(scratchmessage, '(a, i0, a, i0)')
00127             'Could not open the log file = ' // trim(adjustl(logfilename)) // &
00128             ' on logical unit LUN_LOG = ', lun_log, &
00129             '. Error code was: errorIO = ', errorio
00130         CALL screenmessage(error, scratchmessage)
00131     END IF
00132
00133 END SUBROUTINE openlogfile
00134
00135 !=====
00136
00137 !-----
00138 !      S U B R O U T I N E      C L O S E      L O G      F I L E
00139 !-----
00147 !-----
00148 SUBROUTINE closelogfile()
00149
00150     IMPLICIT NONE
00151
00152     IF (logfileopened) CLOSE(unit=lun_log)
00153
00154 END SUBROUTINE closelogfile
00155
00156 !=====
00157
00158 !-----
00159 !      S U B R O U T I N E      S C R E E N      M E S S A G E
00160 !-----
00179 !-----
00180 SUBROUTINE screenmessage_1(message)
00181
00182     IMPLICIT NONE

```

```

00183      ! Global variables
00184      CHARACTER(LEN=*) , INTENT(IN) :: message
00185
00186      IF (nscreen > 0) THEN
00187          IF (loginitcalled) THEN
00188              WRITE(lun_screen, '(a)') '    --- ' // trim(adjustl(message))
00189          ELSE
00190              WRITE(lun_screen, '(a, :: ", a, :: ", a)') 'InitLogging not called',
00191                                         trim(adjustl(messagesources(sourcenumber))), &
00192                                         trim(adjustl(message))
00193          END IF
00194      !#ifdef FLUSH_MESSAGES
00195          ! In Fortran >=2003 the call is:
00196          ! FLUSH(LUN_LOG)
00197          CALL flush(lun_screen)
00198      !#endif
00199      END IF
00200
00201      END SUBROUTINE screenmessage_1
00202
00203
00204      SUBROUTINE screenmessage_2(level, message)
00205
00206      IMPLICIT NONE
00207
00208      ! Global variables
00209      INTEGER, INTENT(IN) :: level
00210      CHARACTER(LEN=*) , INTENT(IN) :: message
00211
00212      IF (nscreen > 0) THEN
00213          IF (loginitcalled) THEN
00214              WRITE(lun_screen, '(a, :: ", a, :: ", a)') trim(adjustl(loglevelnames(level + 2))), &
00215                                         trim(adjustl(messagesources(sourcenumber))), &
00216                                         trim(adjustl(message))
00217          ELSE
00218              WRITE(lun_screen, '(a, :: ", a, :: ", a)') 'InitLogging not called',
00219                                         trim(adjustl(messagesources(sourcenumber))), &
00220                                         trim(adjustl(message))
00221          END IF
00222      !#ifdef FLUSH_MESSAGES
00223          ! In Fortran >=2003 the call is:
00224          ! FLUSH(LUN_LOG)
00225          CALL flush(lun_screen)
00226      !#endif
00227      END IF
00228
00229      END SUBROUTINE screenmessage_2
00230
00231 !=====
00232
00233 !-----
00234 !     S   U   B   R   O   U   T   I   N   E   L   O   G   M   E   S   S   A   G   E
00235 !-----
00236
00237
00238      SUBROUTINE logmessage_1(message)
00239
00240      IMPLICIT NONE
00241
00242      ! Global variables
00243      CHARACTER(LEN=*) , INTENT(IN) :: message
00244
00245      IF (logfileopened) THEN
00246          IF (loginitcalled) THEN
00247              WRITE(lun_log, '(a)') '    --- ' // trim(adjustl(message))
00248          ELSE
00249              WRITE(lun_log, '(a, :: ", a, :: ", a)') 'InitLogging not called',
00250                                         trim(adjustl(messagesources(sourcenumber))), &
00251                                         trim(adjustl(message))
00252          END IF
00253      !#ifdef FLUSH_MESSAGES
00254          ! In Fortran >=2003 the call is:
00255          ! FLUSH(LUN_LOG)
00256          CALL flush(lun_log)
00257      !#endif
00258      END IF
00259
00260      END SUBROUTINE logmessage_1
00261
00262
00263      SUBROUTINE logmessage_2(level, message)
00264
00265      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 !-----
00305 !     S U B R O U T I N E   A L L   M E S S A G E
00306 !-----
00307 !-----
00308 SUBROUTINE allmessage_1(message)
00309
00310     IMPLICIT NONE
00311
00312     ! Global variables
00313     CHARACTER(LEN=*), INTENT(IN)  :: message
00314
00315     CALL screenmessage(message)
00316     CALL logmessage(message)
00317
00318 END SUBROUTINE allmessage_1
00319
00320 SUBROUTINE allmessage_2(level, message)
00321
00322     IMPLICIT NONE
00323
00324     ! Global variables
00325     INTEGER, INTENT(IN)          :: level
00326     CHARACTER(LEN=*), INTENT(IN)  :: message
00327
00328     CALL screenmessage(level, message)
00329     CALL logmessage(level, message)
00330
00331 END SUBROUTINE allmessage_2
00332
00333
00334 !-----
00335 !     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
00336 !-----
00337 !-----
00338 SUBROUTINE setmessagesource(source)
00339
00340     IMPLICIT NONE
00341
00342     ! Global variables
00343     CHARACTER(LEN=*), INTENT(IN)  :: source
00344
00345     sourcenumber = sourcenumber + 1
00346     messagesources(sourcenumber) = source
00347
00348 END SUBROUTINE setmessagesource
00349
00350 !-----
00351 !     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
00352 !-----
00353 !-----
00354 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
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
00466     USE version
00467
00468     IMPLICIT NONE
00469
00470     CALL setmessagesource("Terminate")
00471
00472     CALL allmessage(error, trim(adjustl(prog_name)) // " Terminating.")
00473
00474     CALL unsetmessagesource()
00475
00476     stop
00477
00478 END SUBROUTINE terminate
00479
00480 !=====
00481
00482 END MODULE pahm_messages
```

17.29 /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_netcdfio](#)

Macros

- #define [NetCDFCheckErr\(arg\)](#) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)

Functions/Subroutines

- subroutine [pahm_netcdfio::initadcircnetcdfoutfile](#) (adcircOutFile)
Initializes a new NetCDF data file and puts it in define mode.
- subroutine [pahm_netcdfio::newadcircnetcdfoutfile](#) (ncID, adcircOutFile)
Creates a new NetCDF data file and puts it in define mode.
- subroutine [pahm_netcdfio::base_netcdfcheckerr](#) (ierr, file, line)
Checks the return value from netCDF calls.
- subroutine [pahm_netcdfio::netcdfterminate](#) ()
Terminates the program on NetCDF error.
- subroutine [pahm_netcdfio::writenetcdfrecord](#) (adcircOutFile, timeLoc)
Writes data to the NetCDF file.
- subroutine [pahm_netcdfio::setrecordcounterandstoretime](#) (ncID, f, t)
Sets the record counter.

Variables

- integer, private `pahm_ncdfio::ncformat`
- integer, parameter, private `pahm_ncdfio::nc4form` = IOR(NF90_NETCDF4, NF90_CLASSIC_← MODEL)
- integer, parameter, private `pahm_ncdfio::nc3form` = IOR(NF90_CLOBBER, 0)
- integer, private `pahm_ncdfio::nodedimid`
- integer, private `pahm_ncdfio::vertdimid`
- integer, private `pahm_ncdfio::elemdimid`
- integer, private `pahm_ncdfio::meshdimid`
- integer, private `pahm_ncdfio::meshvarid`
- integer, private `pahm_ncdfio::projvarid`
- type(`filedata_t`), save `pahm_ncdfio::myfile`
- type(`timedata_t`), save `pahm_ncdfio::mytime`
- type(`adcirccoorddata_t`), save, private `pahm_ncdfio::crdtime`
- type(`adcirccoorddata_t`), save, private `pahm_ncdfio::crdlons`
- type(`adcirccoorddata_t`), save, private `pahm_ncdfio::crdlats`
- type(`adcirccoorddata_t`), save, private `pahm_ncdfio::crdxcs`
- type(`adcirccoorddata_t`), save, private `pahm_ncdfio::crdycs`
- type(`adcircvardata_t`), save, private `pahm_ncdfio::datelements`
- type(`adcircvardata_t`), save, private `pahm_ncdfio::datatmpres`
- type(`adcircvardata_t`), save, private `pahm_ncdfio::datwindx`
- type(`adcircvardata_t`), save, private `pahm_ncdfio::datwindy`

17.29.1 Detailed Description

Author

Panagiotis Velissariou `panagiotis.velissariou@noaa.gov`

Definition in file `netcdfio.F90`.

17.29.2 Macro Definition Documentation

17.29.2.1 NetCDFCheckErr

```
#define NetCDFCheckErr( arg ) BASE_NetCDFCheckErr(arg, __FILE__, __LINE__)
```

17.30 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 :: crdtimes
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 !-----
00122 !-----
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
00143 IF (firstcall) THEN
00144     firstcall = .false.
00145
00146     CALL setmessagesource("InitAdcircNetCDFOutFile")
00147
00148
00149     refdatetimestr = datetime2string(refyear, refmonth, refday, refhour, refmin, refsec, units =
00150     unittime)
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, crdtimes%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, jcnt)
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 = xcslam
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) = crdtime%dimID
00390      datatmpres%varDims(1)   = SIZE(wpress, 1)
00391      datatmpres%varDims(2)   = crdtime%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) = crdtime%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, ncdlevel)
00482     CALL netcdfcheckerr(ierr)
00483     ierr = nf90_def_var_deflate(ncid, crdlats%varID,      ncshuffle, ncdeflate, ncdlevel)
00484     CALL netcdfcheckerr(ierr)
00485     ierr = nf90_def_var_deflate(ncid, crdxcs%varID,      ncshuffle, ncdeflate, ncdlevel)
00486     CALL netcdfcheckerr(ierr)
00487     ierr = nf90_def_var_deflate(ncid, crdycs%varID,      ncshuffle, ncdeflate, ncdlevel)
00488     CALL netcdfcheckerr(ierr)
00489     ierr = nf90_def_var_deflate(ncid, datelements%varID, ncshuffle, ncdeflate, ncdlevel)
00490     CALL netcdfcheckerr(ierr)
00491     ierr = nf90_def_var_deflate(ncid, datatmpres%varID, ncshuffle, ncdeflate, ncdlevel)
00492     CALL netcdfcheckerr(ierr)
00493     ierr = nf90_def_var_deflate(ncid, datwindx%varID,      ncshuffle, ncdeflate, ncdlevel)
00494     CALL netcdfcheckerr(ierr)
00495     ierr = nf90_def_var_deflate(ncid, datwindy%varID,      ncshuffle, ncdeflate, ncdlevel)
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
00531     CALL date_and_time(values = tvals)
00532     WRITE(moddatetimestr, '(i3.2, ":00")') tvals(4) / 60 ! this is the timezone
00533     moddatetimestr = datetime2string(tvals(1), tvals(2), tvals(3), tvals(5), tvals(6), tvals(7), zone =
00534     moddatetimestr)
00535
00536     ierr = nf90_put_att(ncid, nf90_global, 'creation_date', trim(moddatetimestr))
00537     CALL netcdfcheckerr(ierr)
00538
00539 !----- Finalize the definitions in the NetCDF file
00540     ierr = nf90_enddef(ncid)
00541     CALL netcdfcheckerr(ierr)
00542
00543 !=====
00544 !===== (5) Put the static data into the NetCDF file and then close it
00545 !=====
00546     ierr = nf90_put_var(ncid, crdlons%varID, crdlons%var, crdlons%start, crdlons%count)
00547     CALL netcdfcheckerr(ierr)
00548
00549     ierr = nf90_put_var(ncid, crdlats%varID, crdlats%var, crdlats%start, crdlats%count)
00550     CALL netcdfcheckerr(ierr)
00551
00552     ierr = nf90_put_var(ncid, crdxcs%varID, crdxcs%var, crdxcs%start, crdxcs%count)
00553     CALL netcdfcheckerr(ierr)
00554
00555     ierr = nf90_put_var(ncid, crdycs%varID, crdycs%var, crdycs%start, crdycs%count)
00556     CALL netcdfcheckerr(ierr)
00557
00558     ierr = nf90_put_var(ncid, datelements%varID, datelements%var, datelements%start, datelements%count)
00559     CALL netcdfcheckerr(ierr)
00560
00561 !=====
00562 !===== (6) Set all the "initialized" flags to .TRUE.
00563 !=====
00564     crdlons%initialized = .true.
00565     crdlats%initialized = .true.
00566     crdxcs%initialized = .true.
00567     crdycs%initialized = .true.
00568     datelements%initialized = .true.
00569     datatmpres%initialized = .true.
00570     datwindx%initialized = .true.
00571     datwindy%initialized = .true.
00572
00573     myfile%fileName = adcircoutfile
00574     myfile%initialized = .true.
00575
00576 !---- Close the NetCDF file
00577     ierr = nf90_close(ncid)
00578     CALL netcdfcheckerr(ierr)
00579
00580     CALL unsetmessagesource()
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 !-----  

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 netcdffterminate
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)

```

```
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 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 writenetcdffrecord
00825
00826 !=====
00827
00828 !----- 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
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 !----- 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
00831
00832 SUBROUTINE setrecordcounterandstoretime(ncID, f, t)
00833
00834     IMPLICIT NONE
00835
00836     INTEGER, INTENT(IN) :: ncID
00837     TYPE(filedata_t), INTENT(INOUT) :: f
00838     TYPE(timedata_t), INTENT(INOUT) :: t
00839
00840     REAL(SZ), ALLOCATABLE :: storedTimes(:) ! array of time values in file
00841     LOGICAL :: timeFound ! true if current time is in array of stored times
00842
00843     INTEGER :: ndim ! number of dimensions in the netcdf file
00844     INTEGER :: nvar ! number of variables in the netcdf file
00845     INTEGER :: natt ! number of attributes in the netcdf file
00846
00847     INTEGER :: counti(1), starti(1)
00848     INTEGER :: ierr ! success or failure of netcdf call
00849     INTEGER :: i ! loop counter
00850
00851
00852     CALL setmessagesource("SetRecordCounterAndStoreTime")
00853
00854     ! Inquire the time variable
00855     ierr = nf90_inquire(ncid, ndim, nvar, natt, t%timeDimID)
00856     CALL netcdfcheckerr(ierr)
00857
00858     ierr = nf90_inquire_dimension(ncid, t%timeDimID, len = f%fileRecCounter)
00859     CALL netcdfcheckerr(ierr)
00860
00861     ierr = nf90_inq_varid(ncid, 'time', t%timeID)
00862     CALL netcdfcheckerr(ierr)
00863
00864     ! Determine the relationship between the current simulation time
00865     ! and the time array stored in the netcdf file. Set the record
00866     ! counter based on this relationship.
00867     IF (f%fileRecCounter /= 0) THEN
00868         ALLOCATE(storedtimes(f%fileRecCounter))
00869         ierr = nf90_get_var(ncid, t%timeID, storedtimes)
00870         CALL netcdfcheckerr(ierr)
00871
00872
00873
00874
00875
00876
00877
00878
00879
00880
00881
00882
00883
00884
00885
00886
00887
00888

```

```
00889     timefound = .false.
00890
00891     DO i = 1, f%fileRecCounter
00892       IF ((t%time(1) < storedtimes(i)) .OR. (abs(t%time(1) - storedtimes(i)) < 1.0d-10)) THEN
00893         timefound = .true.
00894         EXIT
00895       ENDIF
00896     END DO
00897
00898     IF (timefound .EQV. .false.) THEN
00899       ! Increment the record counter so that we can store data at the
00900       ! next location in the netcdf file (i.e., all of the times
00901       ! in the netcdf file were found to be earlier than the current
00902       ! adcirc simulation time).
00903       f%fileRecCounter = f%fileRecCounter + 1
00904     ELSE
00905       ! set the counter at the index that reflects the
00906       ! current time within the netcdf file (or is between two times
00907       ! found in the netcdf file).
00908       ! WARNING: all subsequent data will remain in the file, we
00909       ! are just overwriting it ... if we don't overwrite all of it,
00910       ! the pre-existing data will still be there, which is probably
00911       ! not what the user intended ... but apparently there is no
00912       ! way to delete data from netcdf files:
00913       ! http://www.unidata.ucar.edu/support/help/MailArchives/netcdf/msg02367.html
00914       scratchformat = ('Overwriting pre-existing data in netcdf file ",a,' // &
00915       '" for time=",f17.8,".' // 'Subsequent data in netcdf file remain unchanged.')
00916       WRITE(scratchmessage, scratchformat) trim(f%fileName), t%time(1)
00917       CALL allmessage(info, scratchmessage)
00918       f%fileRecCounter = i
00919     ENDIF
00920
00921     DEALLOCATE(storedtimes)
00922   ELSE
00923     ! set the counter at 1 so we can record our first time value
00924     f%fileRecCounter = 1
00925   ENDIF
00926
00927   ! Store simulation time in netcdf file
00928   starti(1) = f%fileRecCounter
00929   counti(1) = t%timelen
00930   ierr = nf90_put_var(ncid, t%timeID, t%time, starti, counti)
00931   CALL netcdfcheckerr(ierr)
00932
00933   CALL unsetmessagesource()
00934
00935 END SUBROUTINE setrecordcounterandstoretime
00936
00937 !=====
00938 END MODULE pahm_netcdfio
```

17.31 /home/takis/CSDL/parwinds-doc/src/pahm.F90 File Reference

Main PaHM program, calls Init, Run and Finalize procedures.

Functions/Subroutines

- program pahm

17.31.1 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](#).

17.31.2 Function/Subroutine Documentation

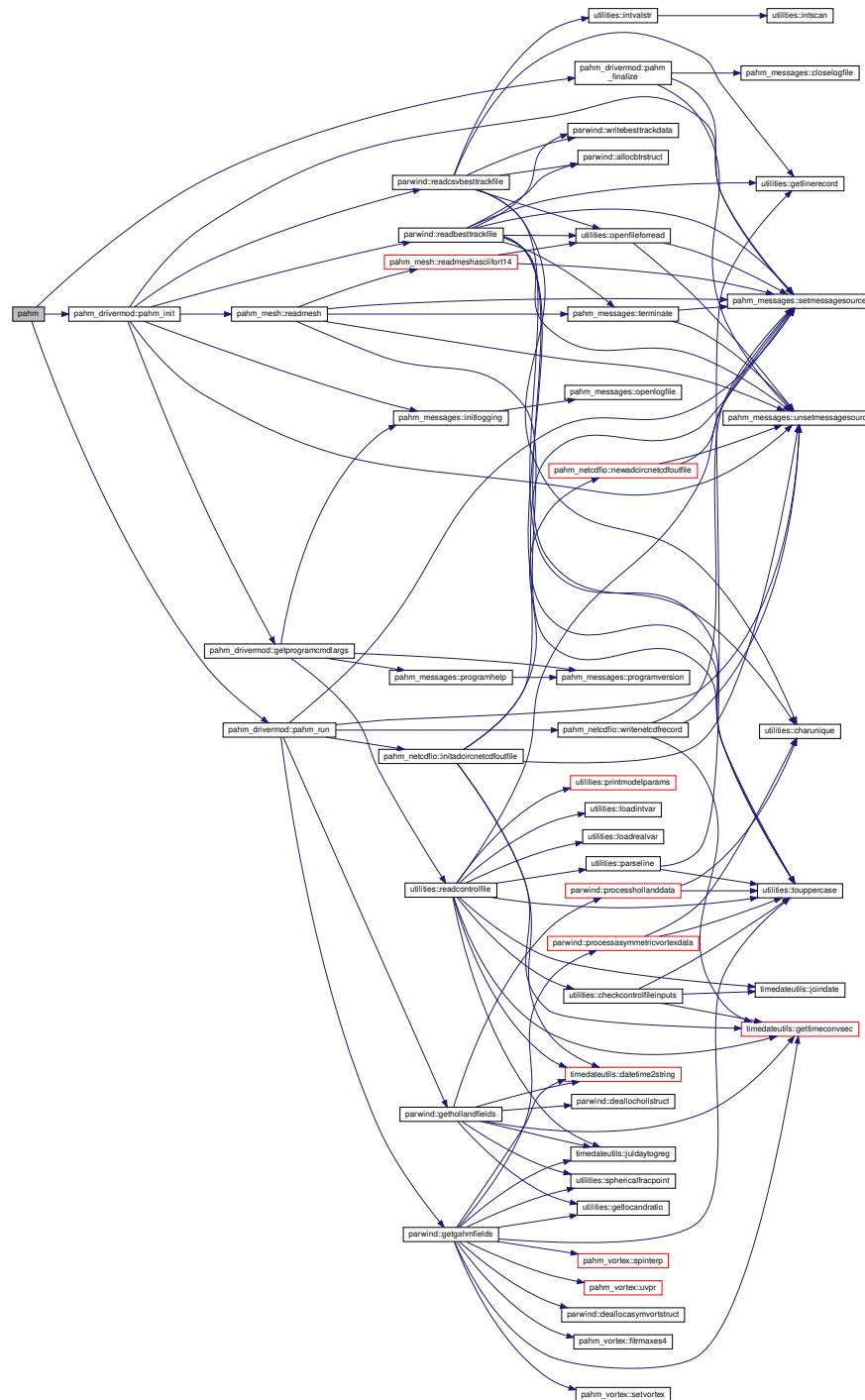
17.31.2.1 pahm()

```
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:



17.32 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 IMPLICIT NONE  

00031  

00032 CALL pahm_init()  

00033  

00034 CALL pahm_run()  

00035  

00036 CALL pahm_finalize()  

00037  

00038 END PROGRAM pahm

```

17.33 /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](#)

17.33.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [parwind.F90](#).

17.34 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 gostrophic balance  !PV shouldn't be a user
00026   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
00045     best track: 00 - 99
00046     CHARACTER(LEN=4), ALLOCATABLE :: tech(:)           ! acronym for each objective technique or CARQ or
00047     WRNG,
00048     INTEGER, ALLOCATABLE        :: tau(:)             ! BEST for best track, up to 4 chars.
00049     best-track,                :: tau(:)             ! forecast period: -24 through 240 hours, 0 for
00050     INTEGER, ALLOCATABLE        :: intlat(:)          ! negative taus used for CARQ and WRNG records.
00051     INTEGER, ALLOCATABLE        :: intlon(:)          ! latitude for the DTG: 0 - 900 tenths of degrees
00052     CHARACTER(LEN=1), ALLOCATABLE :: ew(:)              ! latitude for the DTG: 0 - 900 tenths of degrees
00053     CHARACTER(LEN=1), ALLOCATABLE :: ns(:)              ! E/W
00054     INTEGER, ALLOCATABLE        :: intvmax(:)          ! N/S
00055     kts
00056     INTEGER, ALLOCATABLE        :: intmslp(:)          ! maximum sustained wind speed in knots: 0 - 300
00057     CHARACTER(LEN=2), ALLOCATABLE :: ty(:)               ! minimum sea level pressure, 850 - 1050 mb
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075     INTEGER, ALLOCATABLE        :: rad(:)             ! highest level of tc development:
00076     record: 34, 50 or 64 kt
00077     CHARACTER(LEN=3), ALLOCATABLE :: windcode(:)        ! DB - disturbance,
00078
00079     INTEGER, ALLOCATABLE        :: intrad1(:)          ! TD - tropical depression,
00080     intensity, or radius of    :: intrad1(:)          ! TS - tropical storm,
00081     WINDCODE. 0 - 999 n mi    :: intrad1(:)          ! TY - typhoon,
00082     INTEGER, ALLOCATABLE        :: intrad2(:)          ! ST - super typhoon,
00083     of 2nd quadrant wind       :: intrad2(:)          ! TC - tropical cyclone,
00084     mi
00085     INTEGER, ALLOCATABLE        :: intrad3(:)          ! HU - hurricane,
00086     of 3rd quadrant wind       :: intrad3(:)          ! SD - subtropical depression,
00087     mi
00088     INTEGER, ALLOCATABLE        :: intrad4(:)          ! SS - subtropical storm,
00089     of 4th quadrant wind       :: intrad4(:)          ! EX - extratropical systems,
00090     mi
00091     INTEGER, ALLOCATABLE        :: intpouter(:)        ! PT - post tropical,
00092     isobar, 900 - 1050 mb      :: intpouter(:)        ! IN - inland,
00093     INTEGER, ALLOCATABLE        :: introuter(:)        ! DS - dissipating,
00094     INTEGER, ALLOCATABLE        :: intrmw(:)           ! LO - low,
00095     INTEGER, ALLOCATABLE        :: gusts(:)            ! WV - tropical wave,
00096     eye(:)                   ! ET - extrapolated,
00097     CHARACTER(LEN=3), ALLOCATABLE :: subregion(:)        ! MD - monsoon depression,
00098
00099     ! XX - unknown.
00100
00101     ! wind intensity for the radii defined in this
00102     ! radius code:
00103     ! AAA - full circle
00104     ! NEQ, SEQ, SWQ, NWQ - quadrant
00105     ! if full circle, radius of specified wind
00106
00107     ! first quadrant wind intensity as specified by
00108     ! if full circle this field not used, or radius
00109     ! intensity as specified by WINDCODE. 0 - 999 n
00110     ! mi
00111     ! if full circle this field not used, or radius
00112     ! intensity as specified by WINDCODE. 0 - 999 n
00113     ! mi
00114     ! if full circle this field not used, or radius
00115     ! intensity as specified by WINDCODE. 0 - 999 n
00116     ! mi
00117     ! pressure in millibars of the last closed
00118     ! isobar, 900 - 1050 mb
00119     ! radius of the last closed isobar, 0 - 999 n mi
00120     ! radius of max winds, 0 - 999 n mi
00121     ! gusts, 0 - 999 kt
00122     ! eye diameter, 0 - 120 n mi
00123     ! subregion code: W,A,B,S,P,C,E,L,Q
00124     ! A - Arabian Sea

```

```

00094 !      B - Bay of Bengal
00095 !      C - Central Pacific
00096 !      E - Eastern Pacific
00097 !      L - Atlantic
00098 !      P - South Pacific (135E - 120W)
00099 !      Q - South Atlantic
00100 !      S - South IO (20E - 135E)
00101 !      W - Western Pacific
00102      INTEGER, ALLOCATABLE      :: maxseas(:)      ! max seas: 0 - 999 ft
00103      CHARACTER(LEN=3), ALLOCATABLE :: initials(:)   ! forecaster's initials used for tau 0 WRNG or
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:          !           cy = Annual cyclone number 01 - 99
00109      !           x  = Subregion code: W,A,B,S,P,C,E,L,Q.
00110      INTEGER, ALLOCATABLE      :: cyclenum(:)    ! the cycle number
00111
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
00151      INTEGER,                  ALLOCATABLE :: irmw(:)      ! radius of max winds, 0 - 999 n mi
00152      REAL(sz),                  ALLOCATABLE :: rmw(:)      ! converted from nm to m
00153
00154      REAL(sz), DIMENSION(:),  ALLOCATABLE :: cprdt      ! central pressure intensity change (Pa / h)
00155      REAL(sz), DIMENSION(:),  ALLOCATABLE :: trvx, trvy  ! translational velocity components (x, y) of
00156      the
00157
00158 END TYPE hollanddata_t
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
00176      INTEGER, ALLOCATABLE       :: casttypenum(:)          ! objective technique sorting number, minutes
00177      for best track: 00 - 99
00178      CHARACTER(LEN=4),          ALLOCATABLE :: casttype(:)          ! BEST, OFCL, CALM, ...
00179      INTEGER, ALLOCATABLE       :: fcstinc(:)           ! forecast period: -24 through 240 hours, 0
00180
00181      INTEGER, DIMENSION(:),    ALLOCATABLE :: ilat, ilon           ! latitude, longitude for the GTD
00182      REAL(sz), DIMENSION(:),    ALLOCATABLE :: lat, lon           ! converted to decimal E/N (lon, lat)
00183      CHARACTER(LEN=1),          ALLOCATABLE :: ns(:), ew(:)           ! N/S and E/W character
00184      INTEGER,                  ALLOCATABLE :: ispeed(:)           ! maximum sustained wind speed in knots: 0 -
00185      300 kts
00186      REAL(sz),                ALLOCATABLE :: speed(:)           ! converted from kts to m/s
00187      INTEGER,                  ALLOCATABLE :: icpress(:)          ! minimum sea level pressure, 850 - 1050 mb
00188      REAL(sz),                ALLOCATABLE :: cpres(:)           ! converted to Pa
00189      CHARACTER(LEN=2),          ALLOCATABLE :: ty(:)           ! Highest level of tc development (see best
00190      track structure)
00191      INTEGER, ALLOCATABLE       :: ivr(:)           ! wind intensity for the radii defined in this
00192      record: 34, 50 or 64 kt
00193      CHARACTER(LEN=3),          ALLOCATABLE :: windcode(:)          ! radius code: AAA - full circle, NEQ, SEQ,
00194      SWQ, NWQ - quadrant
00195      INTEGER, ALLOCATABLE       :: ir(:, :)           ! if full circle, radius of specified wind
00196      intensity, or radius of
00197      quadrant, 4: 4th quadrant
00198
00199      INTEGER,                  ALLOCATABLE :: iprp(:)           ! pressure in millibars of the last closed
00200      isobar, 900 - 1050 mb
00201      REAL(sz),                ALLOCATABLE :: prp(:)           ! converted to Pa
00202
00203      INTEGER,                  ALLOCATABLE :: irrp(:)           ! radius of the last closed isobar, 0 - 999 n
00204      mi
00205      REAL(sz),                ALLOCATABLE :: rrp(:)           ! converted from nm to m
00206
00207      INTEGER,                  ALLOCATABLE :: irmw(:)           ! radius of max winds, 0 - 999 n mi
00208      REAL(sz),                ALLOCATABLE :: rmw(:)           ! converted from nm to m
00209
00210      INTEGER, ALLOCATABLE       :: gusts(:)           ! gusts, 0 - 999 kt
00211      INTEGER, ALLOCATABLE       :: eye(:)            ! eye diameter, 0 - 120 n mi
00212      CHARACTER(LEN=3),          ALLOCATABLE :: subregion(:)          ! subregion code: W,A,B,S,P,C,E,L,Q
00213
00214      ! A - Arabian Sea
00215      ! B - Bay of Bengal
00216      ! C - Central Pacific
00217      ! E - Eastern Pacific
00218      ! L - Atlantic
00219      ! P - South Pacific (135E - 120W)
00220      ! Q - South Atlantic
00221      ! S - South IO (20E - 135E)
00222      ! W - Western Pacific
00223      ! max seas: 0 - 999 ft
00224      ! forecaster's initials used for tau 0 WRNG or
00225
00226      INTEGER, ALLOCATABLE       :: maxseas(:)
00227      CHARACTER(LEN=3),          ALLOCATABLE :: initials(:)
00228      OFCL, up to 3 chars
00229
00230      REAL(sz), DIMENSION(:),   ALLOCATABLE :: trvx, trvy           ! translational velocity components (x, y) of
00231      the
00232
00233      INTEGER, ALLOCATABLE       :: idir(:)           ! moving hurricane (m/s)
00234      REAL(sz), ALLOCATABLE       :: dir(:)
00235      INTEGER, ALLOCATABLE       :: istormspeed(:)          ! storm direction, 0 - 359 degrees
00236      REAL(sz), ALLOCATABLE       :: stormspeed(:)          ! storm speed, 0 - 999 kts
00237      CHARACTER(LEN=STORMNAMELEN), ALLOCATABLE &
00238      stormname(:)           ! converted from kts to m/s
00239
00240      ! literal storm name, number, NONAME or
00241      ! cy = Annual cyclone number 01 - 99
00242      ! x = Subregion code: W,A,B,S,P,C,E,L,Q.
00243
00244      ! extended/asymmetric vortex data
00245      INTEGER,                  :: ncycles
00246      INTEGER, DIMENSION(:),    ALLOCATABLE :: numcycle
00247      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 :: hollbs
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 !-----
00265   SUBROUTINE readbesttrack()
00267
00268     USE pahm_global, ONLY : lun_btrk, lun_btrk1, nbtrfiles, besttrackfilename
00269     USE utilities, ONLY : getlinerecord, openfileforread, touppercase, charunique
00270     USE timedateutils, ONLY : timeconv
00271     USE sortutils, ONLY : arth, indexx, arrayequal
00272
00273     IMPLICIT NONE
00274
00275     CHARACTER(LEN=FNAMELEN)      :: inpfile
00276     CHARACTER(LEN=512)          :: inpline, line
00277     CHARACTER(LEN=512)          :: fmtStr
00278
00279     INTEGER                     :: lenLine
00280     INTEGER                     :: nLines           ! Number of lines counter
00281     INTEGER                     :: iFile, iCnt        ! loop counters
00282     INTEGER                     :: iUnit, errIO, ios, status
00283
00284     !CHARACTER(LEN=4)            :: castType
00285
00286     CHARACTER(LEN=10), ALLOCATABLE :: chkArrStr(:)
00287     INTEGER, ALLOCATABLE         :: idxArrStr(:)
00288     INTEGER                      :: nUnique, maxCnt, kCnt, kMax
00289
00290     INTEGER, ALLOCATABLE         :: idx0(:), idx1(:)
00291     REAL(SZ)                   :: tmpFcstTime, refFcstTime
00292
00293 !----- Initialize variables
00294     iunit = lun_btrk
00295     errio = 0
00296
00297     !fmtStr = '(a2, 2x, i2, 2x, a10, 2x, i2, 2x, a4, 2x, i3, 2x, i3, a1, 2x, i4, a1, 2x, i3, 2x, i4, 2x,
00298 a2,
00299     fmtstr = '(a2, 1x, i3, 2x, a10, 1x, i3, 2x, a4, 2x, i3, 2x, i3, a1, 1x, i5, a1, 2x, i3, 2x, i4, 2x,
00300 a2,
00301     fmtstr = trim(fmtstr) // ' 2x, i3, 2x, a3, 4(2x, i4), 2x, i4, 2x, i3, 2x, i3, 2x, i3, '
00302     fmtstr = trim(fmtstr) // ' 2x, a3, 2x, i3, 2x, a3, 1x, 2(i3, 2x), a11'
00303 !-----
00304
00305     CALL setmessagesource("ReadBestTrackFile")
00306
00307 ! Allocate the best track structure array. This structure holds all the
00308 ! input values for the storm track as read in from the track input file
00309 ! (a-deck, b-deck ATCF format) as well as the converted best track variables
00310 ! (as appropriate).
00311     ALLOCATE(besttrackdata(nbtrfiles))
00312
00313 ! This is the main loop. We loop through all the best track files
00314 ! (user input)
00315 DO ifile = 1, nbtrfiles
00316   inpfile = besttrackfilename(ifile)
00317
00318   CALL openfileforread(iunit, trim(adjustl(inpfile)), errio)
00319   IF (errio /= 0) THEN
00320     WRITE(scratchmessage, '(a)') 'Error opening the best track file: ' // trim(adjustl(inpfile))
00321     CALL allmessage(error, scratchmessage)
00322
00323     CALL unsetmessagesource()
00324
00325     CALL terminate()
00326   ELSE
00327     WRITE(scratchmessage, '(a)') 'Processing the best track file: ' // trim(adjustl(inpfile))
00328     CALL logmessage(info, scratchmessage)

```

```

00328     END IF
00329
00330     besttrackdata(ifile)%fileName = trim(adjustl(inpfile))
00331     besttrackdata(ifile)%thisStorm = ""
00332     besttrackdata(ifile)%loaded = .false.
00333     besttrackdata(ifile)%numRec = -1
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)

```

```

00408      besttrackdata(ifile)%month(icnt)
00409          IF (ios /= 0) besttrackdata(ifile)%month(icnt) = -1
00410          READ(besttrackdata(ifile)%month(icnt)(7:8), fmt='(i2.2)', iostat=ios)
00411          besttrackdata(ifile)%day(icnt)
00412              IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00413                  READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00414          besttrackdata(ifile)%hour(icnt)
00415              IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00416          !-----
00417      END IF
00418  END DO
00419
00420  10 IF (errio /= 0) THEN
00421      WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00422          ', while processing line: ' // trim(adjustl(inpline))
00423      CALL allmessage(error, scratchmessage)
00424
00425      CLOSE(iunit)
00426
00427      CALL unsetmessagesource()
00428
00429      CALL terminate()
00430  END IF
00431
00432  11 IF (ios /= 0) THEN
00433      WRITE(scratchmessage, '(a)') 'Error in file: ' // trim(adjustl(inpfile)) // &
00434          ', while processing line: ' // trim(adjustl(line))
00435      CALL allmessage(error, scratchmessage)
00436
00437      CLOSE(iunit)
00438
00439      CALL unsetmessagesource()
00440
00441  20 CLOSE(iunit)
00442
00443      besttrackdata(ifile)%thisStorm = ""
00444      besttrackdata(ifile)%loaded     = .true.
00445      besttrackdata(ifile)%numRec    = nlines
00446
00447
00448  !-----
00449  ! Get the unique storm name and store it in the thisStorm string
00450  ALLOCATE(chkarrstr(nlines))
00451  ALLOCATE(idxarrstr(nlines))
00452
00453  nunique = charunique(besttrackdata(ifile)%stormName, chkarrstr, idxarrstr)
00454
00455  maxcnt = -1
00456  DO kcnt = 1, nunique
00457      kmax = count(chkarrstr(kcnt) == besttrackdata(ifile)%stormName)
00458      IF (kmax > maxcnt) THEN
00459          maxcnt = kmax
00460          besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcnt)))
00461      END IF
00462  END DO
00463
00464  DEALLOCATE(chkarrstr)
00465  DEALLOCATE(idxarrstr)
00466
00467  !-----
00468  ! This is an extra step (paranoid) to ensure that the dates in the bestTrackData are
00469  ! stored in ascending order
00470  ALLOCATE(idx0(besttrackdata(ifile)%numRec))
00471  ALLOCATE(idx1(besttrackdata(ifile)%numRec))
00472
00473  CALL indexx(besttrackdata(ifile)%dtg, idx1, status, .true.)
00474
00475  IF (status /= 0) THEN
00476      CALL unsetmessagesource()
00477
00478      CALL terminate()
00479  END IF
00480
00481  ! Create the index array to be used in the comparison below
00482  idx0 = arth(1, 1, besttrackdata(ifile)%numRec)
00483
00484  IF (.NOT. arrayequal(idx0, idx1)) THEN

```

```

00486      besttrackdata(ifile)%basin      = besttrackdata(ifile)%basin(idx1)
00487      besttrackdata(ifile)%cyNum     = besttrackdata(ifile)%cyNum(idx1)
00488      besttrackdata(ifile)%dtg       = besttrackdata(ifile)%dtg(idx1)
00489      besttrackdata(ifile)%techNum   = besttrackdata(ifile)%techNum(idx1)
00490      besttrackdata(ifile)%tech      = besttrackdata(ifile)%tech(idx1)
00491      besttrackdata(ifile)%tau       = besttrackdata(ifile)%tau(idx1)
00492      besttrackdata(ifile)%intLat    = besttrackdata(ifile)%intLat(idx1)
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 DEALLOCATE(idx0)
00519 DEALLOCATE(idx1)
00520 !-----
00521 !----- This should be last after the fields are indexed in ascending order.
00522 !----- It set the cycle number array in the data structure
00523 DO icnt = 1, besttrackdata(ifile)%numRec
00524 ! This is for the cycleNum, the last column we consider
00525 IF (icnt == 1) THEN
00526     kcmt = icnt
00527     besttrackdata(ifile)%cycleNum(icnt) = kcmt
00528 ELSE
00529     kcmt = kcmt + 1
00530     IF (besttrackdata(ifile)%dtg(icnt) == besttrackdata(ifile)%dtg(icnt-1)) THEN
00531         besttrackdata(ifile)%cycleNum(icnt) = besttrackdata(ifile)%cycleNum(icnt-1)
00532     kcmt = kcmt - 1
00533     ELSE
00534         besttrackdata(ifile)%cycleNum(icnt) = kcmt
00535     END IF
00536 END IF
00537 END DO
00538 !----- This should be last after the fields are indexed in ascending order.
00539 !----- We generate arbitrarily the forecast increments for internal use only.
00540 !----- In the best track file, for the BEST track fields the forecast period
00541 !----- is always 0.
00542 !----- This is our reference time for the subsequent calculations
00543 CALL timeconv(besttrackdata(ifile)%year(1), besttrackdata(ifile)%month(1), &
00544             besttrackdata(ifile)%day(1), 0, 0, 0.0_sz, reffcsttime)
00545
00546 DO icnt = 1, besttrackdata(ifile)%numRec
00547     CALL timeconv(besttrackdata(ifile)%year(icnt), besttrackdata(ifile)%month(icnt), &
00548                   besttrackdata(ifile)%day(icnt), besttrackdata(ifile)%hour(icnt), &
00549                   0, 0.0_sz, tmpfcsttime)
00550     besttrackdata(ifile)%tau(icnt) = nint((tmpfcsttime - reffcsttime) / 3600.0_sz)
00551 END DO
00552
00553 CALL writebesttrackdata(besttrackfilename(ifile), besttrackdata(ifile))
00554 END DO ! End of "iFile" loop
00555
00556 CALL unsetmessagessource()
00557
00558 END SUBROUTINE readbesttrackfile
00559
00560
00561
00562
00563 !----- 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
00564
00565
00566

```

```

00567 !-----
00581 !-----
00582 SUBROUTINE readcsvbesttrackfile()
00583
00584   USE pahm_global, ONLY    : nbtrfiles, besttrackfilename
00585   USE utilities, ONLY     : getlinerecord, openfileforread, touppercase, charunique, &
00586           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     kcmt = 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(adjustl(sval2d(icnt, 6))))
00670      !--- col: 7
00671      tmpstr = trim(adjustl(sval2d(icnt, 7)))
00672      READ(tmpstr, '(i3, a1)') &
00673          besttrackdata(ifile)%intLat(icnt), besttrackdata(ifile)%ns(icnt)
00674      !--- col: 8
00675      tmpstr = trim(adjustl(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(adjustl(sval2d(icnt, 9))))
00680      !--- col: 10
00681      besttrackdata(ifile)%intMslp(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 10))))
00682      !--- col: 11
00683      WRITE(besttrackdata(ifile)%ty(icnt), '(a2)') trim(adjustl(sval2d(icnt, 11)))
00684      !--- col: 12
00685      besttrackdata(ifile)%rad(icnt)        = intvalstr(trim(adjustl(sval2d(icnt, 12))))
00686      !--- col: 13
00687      WRITE(besttrackdata(ifile)%windCode(icnt), '(a3)') trim(adjustl(sval2d(icnt, 13)))
00688      !--- col: 14
00689      besttrackdata(ifile)%intRad1(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 14))))
00690      !--- col: 15
00691      besttrackdata(ifile)%intRad2(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 15))))
00692      !--- col: 16
00693      besttrackdata(ifile)%intRad3(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 16))))
00694      !--- col: 17
00695      besttrackdata(ifile)%intRad4(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 17))))
00696      !--- col: 18
00697      besttrackdata(ifile)%intPOuter(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 18))))
00698      !--- col: 19
00699      besttrackdata(ifile)%intROuter(icnt)   = intvalstr(trim(adjustl(sval2d(icnt, 19))))
00700      !--- col: 20
00701      besttrackdata(ifile)%intRmw(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 20))))
00702      !--- col: 21
00703      besttrackdata(ifile)%gusts(icnt)       = intvalstr(trim(adjustl(sval2d(icnt, 21))))
00704      !--- col: 22
00705      besttrackdata(ifile)%eye(icnt)         = intvalstr(trim(adjustl(sval2d(icnt, 22))))
00706      !--- col: 23
00707      WRITE(besttrackdata(ifile)%subregion(icnt), '(a3)') trim(adjustl(sval2d(icnt, 23)))
00708      !--- col: 24
00709      besttrackdata(ifile)%maxseas(icnt)     = intvalstr(trim(adjustl(sval2d(icnt, 24))))
00710      !--- col: 25
00711      besttrackdata(ifile)%initials(icnt)    = trim(adjustl(sval2d(icnt, 25)))
00712      !--- col: 26
00713      besttrackdata(ifile)%dir(icnt)         = intvalstr(trim(adjustl(sval2d(icnt, 26))))
00714      !--- col: 27
00715      besttrackdata(ifile)%intSpeed(icnt)    = intvalstr(trim(adjustl(sval2d(icnt, 27))))
00716      !--- col: 28
00717      WRITE(besttrackdata(ifile)%stormName(icnt), '(a10)') trim(adjustl(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
00733      !----- Get the year, month, day, hour from the DGT string
00734      READ(besttrackdata(ifile)%dtg(icnt)(1:4), fmt='(i4.4)', iostat=ios)
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)

```

```

00739      IF (ios /= 0) besttrackdata(ifile)%day(icnt) = -1
00740      READ(besttrackdata(ifile)%dtg(icnt)(9:10), fmt='(i2.2)', iostat=ios)
00741      IF (ios /= 0) besttrackdata(ifile)%hour(icnt) = -1
00742      !-----
00743
00744      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 kcmt = 1, nunique
00760          kmax = count(chkarrstr(kcmt) == besttrackdata(ifile)%stormName)
00761          IF (kmax > maxcnt) THEN
00762              maxcnt = kmax
00763              besttrackdata(ifile)%thisStorm = trim(adjustl(chkarrstr(kcmt)))
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      END IF
00821
00822      DEALLOCATE(idx0)
00823      DEALLOCATE(idx1)
00824      !-----
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, reffcsttime)
00854
00855      DO icnt = 1, besttrackdata(ifile)%numRec
00856          CALL timeconv(besttrackdata(ifile)%year(icnt), besttrackdata(ifile)%month(icnt), &
00857                         besttrackdata(ifile)%day(icnt), besttrackdata(ifile)%hour(icnt), &
00858                         0, 0.0_sz, tmpfcsttime)
00859          besttrackdata(ifile)%tau(icnt) = nint((tmpfcsttime - reffcsttime) / 3600.0_sz)
00860      END DO
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
00899
00900
00901
00902
00903
00904
00905
00906
00907
00908
00909
00910
00911
00912
00913
00914
00915
00916
00917

```

```

00918     REAL(SZ)                      :: spdVal, pressVal, rrpVal, rmwVal
00919
00920     status = 0 ! no error
00921
00922
00923     CALL setmessagesource("ProcessHollandData")
00924
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: ', idtrfile
00930         CALL allmessage(error, scratchmessage)
00931
00932         CALL unsetmessagesource()
00933
00934         RETURN
00935     END IF
00936
00937     ELSE
00938         status = 1
00939
00940         WRITE(scratchmessage, '(a, i0, a, i0)') 'Wrong best track structure id (idTrFile): ', idtrfile, &
00940             ', it should be: (1<= idTrFile <= nBTrFiles); nBTrFiles = ', nbtrfiles
00941         CALL allmessage(error, scratchmessage)
00942
00943         CALL unsetmessagesource()
00944
00945         RETURN
00946     END IF
00947
00948     WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
00949     CALL logmessage(info, scratchmessage)
00950
00951     ! Most likely the array size will be larger if repeated times are found
00952     ! in the best track structure.
00953     ALLOCATE(outdtg(besttrackdata(idtrfile)%numRec))
00954     ALLOCATE(idxdtg(besttrackdata(idtrfile)%numRec))
00955
00956     ! Get unique lines that represent new points in time.
00957     ! Repeated time points occur in hindcasts for the purpose of
00958     ! describing winds in the quadrants of the storm. We don't use the
00959     ! quadrant-by-quadrant wind data. Repeated time data occur in the
00960     ! forecast because the time data is just the time that the forecast
00961     ! was made. The important parameter in the forecast file is the
00962     ! forecast increment.
00963     numuniqrec = charunique(besttrackdata(idtrfile)%dtg, outdtg, idxdtg)
00964
00965     !-----
00966     ! Populate the Holland structure
00967     !-----
00968     CALL allocholstruct(strout, numuniqrec)
00969
00970     ALLOCATE(casttime(numuniqrec))
00971
00972     strout%fileName = besttrackdata(idtrfile)%fileName
00973     strout>thisStorm = besttrackdata(idtrfile)%thisStorm
00974     strout%loaded = .true.
00975     strout%numRec = numuniqrec
00976
00977     WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
00978     CALL logmessage(info, scratchmessage)
00979
00980     DO icnt = 1, numuniqrec
00981         plidx = idxdtg(icnt)
00982
00983         casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))
00984
00985         ! Convert speeds from knots to m/s
00986         spdval = kt2ms * besttrackdata(idtrfile)%intVMax(plidx)
00987
00988         ! Convert pressure(s) from mbar to Pa
00989         pressval = 100.0_sz * besttrackdata(idtrfile)%intMslp(plidx)
00990
00991         ! Convert all distances from nm to km/m
00992         rrpval = nm2m * besttrackdata(idtrfile)%intROuter(plidx) ! in m
00993         rmwval = nm2m * besttrackdata(idtrfile)%intRmw(plidx) ! in m
00994
00995         strout%basin(icnt) = besttrackdata(idtrfile)%basin(plidx)
00996         strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(plidx)

```

```

00997     stout%dtg(icnt)          = besttrackdata(idtrfile)%dtg(plidx)
00998     stout%year(icnt)         = besttrackdata(idtrfile)%year(plidx)
00999     stout%month(icnt)        = besttrackdata(idtrfile)%month(plidx)
01000     stout%day(icnt)          = besttrackdata(idtrfile)%day(plidx)
01001     stout%hour(icnt)          = besttrackdata(idtrfile)%hour(plidx)
01002     stout%castType(icnt)      = besttrackdata(idtrfile)%tech(plidx)
01003     stout%fcstInc(icnt)       = besttrackdata(idtrfile)%tau(plidx)
01004     stout%iLat(icnt)          = besttrackdata(idtrfile)%intLat(plidx)
01005     stout%lat(icnt)           = besttrackdata(idtrfile)%lat(plidx)
01006     stout%lLon(icnt)          = besttrackdata(idtrfile)%intLon(plidx)
01007     stout%lon(icnt)           = besttrackdata(idtrfile)%lon(plidx)
01008
01009     stout%iSpeed(icnt)        = besttrackdata(idtrfile)%intVMax(plidx)
01010     stout%speed(icnt)          = spdval
01011     stout%icPress(icnt)        = besttrackdata(idtrfile)%intMslp(plidx)
01012     stout%cPress(icnt)          = pressval
01013     stout%iRrp(icnt)           = besttrackdata(idtrfile)%intROuter(plidx)
01014     stout%rrp(icnt)            = rrpval
01015     stout%iRmw(icnt)           = besttrackdata(idtrfile)%intRmw(plidx)
01016     stout%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(stout%year(icnt), stout%month(icnt), stout%day(icnt), stout%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 ( (stout%fcstInc(icnt) /= 0) .AND. (stout%fcstInc(icnt) == stout%fcstInc(icnt - 1)))
01029       cycle
01030       END IF
01031       IF (stout%fcstInc(icnt) == 0) THEN
01032         CALL timeconv(stout%year(icnt), stout%month(icnt), stout%day(icnt), &
01033                     stout%hour(icnt), 0, 0.0_sz, casttime(icnt))
01034       ELSE
01035         casttime(icnt) = casttime(icnt - 1) + (stout%fcstInc(icnt) - stout%fcstInc(icnt - 1)) *
01036             3600.0_sz
01037       END IF
01038       IF ((stout%icPress(icnt) == 0) .OR. (stout%iRmw(icnt) == 0)) THEN
01039         CALL allmessage(error,
01040                         'The storm hindcast/forecast input file ' // trim(stout%fileName) // '&
01041                         ' contains invalid data for central pressure or rMax.')
01042         CALL terminate()
01043       END IF
01044
01045 ! Adding a new type to allow the analyst to add lines
01046 ! that do nothing but produce zero winds and background barometric
01047 ! pressure. These lines can have a date/time like a BEST line or
01048 ! a date/time and forecast period like an OFCL line.
01049 CASE("CALM")
01050   ! PV check if this is needed
01051   WRITE(scratchmessage, '(a)') 'The file: ' // trim(stout%fileName) // ' contains at least one
01052   "CALM" line.'
01053   CALL logmessage.echo, scratchmessage)
01054
01055   IF (icnt > 1) THEN
01056     IF ( (stout%fcstInc(icnt) /= 0) .AND. (stout%fcstInc(icnt) == stout%fcstInc(icnt - 1)))
01057     cycle
01058     END IF
01059     IF (stout%fcstInc(icnt) == 0) THEN
01060       CALL timeconv(stout%year(icnt), stout%month(icnt), stout%day(icnt), &
01061                     stout%hour(icnt), 0, 0.0_sz, casttime(icnt))
01062     ELSE
01063       casttime(icnt) = casttime(icnt - 1) + (stout%fcstInc(icnt) - stout%fcstInc(icnt - 1)) *
01064             3600.0_sz
01065     END IF
01066     CASE DEFAULT      ! unrecognized
01067       WRITE(scratchmessage, '(a)') 'Only "BEST", "OFCL", or "CALM" are allowed in the 5th column of '
01068       // & trim(adjustl(stout%fileName))
01069       CALL allmessage(error, scratchmessage)
01070       CALL terminate()

```

```

01071      END SELECT
01072
01073      strout%castTime(icnt) = casttime(icnt)
01074  END DO    ! numUniqRec
01075
01076      ! Calculate the cPress intensity change (dP/dt)
01077      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 !-----
01098 !-----
01099 SUBROUTINE processasymmetricvortexdata(idTrFile, strOut, status)
01100
01101     USE pahm_global, ONLY : rad2deg, deg2rad, nm2m, kt2ms, ms2kt, &
01102                     backgroundatmpress, windreduction, besttrackfilename, nbtrfiles
01103     USE utilities, ONLY : touppercase, charunique, sphericaldistance
01104     USE timedateutils, ONLY : timeconv
01105     USE pahm_vortex
01106
01107
01108     IMPLICIT NONE
01109
01110     INTEGER, INTENT(IN)          :: idTrFile
01111     TYPE(asymmetricvortexdata_t), INTENT(OUT) :: strOut
01112     INTEGER, INTENT(OUT)          :: status ! error status
01113
01114     ! ----- Local variables
01115     INTEGER                      :: numRec
01116
01117     INTEGER                      :: iCnt, iCyc, i, k ! loop counters
01118
01119     CHARACTER(LEN=4)              :: castType      !hindcast,forecast
01120     REAL(SZ), ALLOCATABLE         :: castTime(:)   ! seconds since start of year
01121
01122     INTEGER                      :: nCycles
01123     REAL(SZ), DIMENSION(:), ALLOCATABLE :: cycleTime
01124     INTEGER, DIMENSION(:), ALLOCATABLE :: totRecPerCycle
01125
01126     INTEGER                      :: radiiSum ! record radius values for filling in missing vals
01127     INTEGER                      :: numNonZero ! number of nonzero isotach radii
01128     INTEGER                      :: firstEntry   ! first entry in the cycle
01129     INTEGER                      :: lastEntry    ! last entry in the cycle
01130     REAL(sz)                     :: stormMotion  ! portion of Vmax attributable to storm motion
01131     REAL(sz)                     :: stormMotionU ! U portion of Vmax attributable to storm motion
01132     REAL(sz)                     :: stormMotionV ! V portion of Vmax attributable to storm motion
01133     REAL(sz)                     :: U_Vr, V_Vr
01134     REAL(SZ), DIMENSION(4)        :: vmwBL
01135     INTEGER, DIMENSION(4)        :: vmwBLflag
01136     REAL(SZ)                     :: vMaxPseudo
01137     REAL(SZ), DIMENSION(:,:,), ALLOCATABLE :: phiFactors
01138     REAL(SZ), DIMENSION(4)        :: vMaxesBLTemp
01139     INTEGER, DIMENSION(:, :, ), ALLOCATABLE :: irad ! working isotach radii
01140     REAL(SZ), DIMENSION(4)        :: rMaxWTemp
01141     INTEGER                      :: iQuadRot, nQuadRot
01142
01143
01144     INTEGER, DIMENSION(0:5)      :: lookupRadii ! periodic interpolation
01145     REAL(SZ), DIMENSION(4)        :: r
01146     REAL(SZ)                     :: pn      ! Ambient surface pressure (mb)
01147     REAL(SZ)                     :: pc      ! Surface pressure at center of storm (mb)
01148     REAL(SZ)                     :: cLat, cLon ! Current eye location (degrees north, degrees east)
01149     REAL(SZ)                     :: vMax   ! Current Max sustained wind velocity in storm (knots)
01150     REAL(SZ), DIMENSION(4)        :: gamma   ! factor applied to the StormMotion
01151     REAL(SZ), DIMENSION(4)        :: quadRotateAngle, quadRotateAngle_new
01152     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
01200          CALL allmessage(error, scratchmessage)
01201
01202          CALL unsetmessagesource()
01203
01204          RETURN
01205      END IF
01206
01207      WRITE(scratchmessage, '(a, i0)') 'Processing the best track structure with id: ', idtrfile
01208      CALL logmessage(info, scratchmessage)
01209
01210      ! This is the number of all records in the processed best track data structure
01211      numrec = besttrackdata(idtrfile)%numRec
01212
01213      !-----
01214      ! Populate the asymmetric vortex structure
01215      !-----
01216      CALL allocasymvortstruct(strout, numrec)
01217
01218      ALLOCATE(casttime(numrec))
01219      ALLOCATE(cycletime(numrec))
01220      ALLOCATE(totrecpercycle(numrec))
01221      ALLOCATE(phifactors(numrec, 4))
01222      ALLOCATE(irad(numrec, 4))
01223
01224      strout%fileName = besttrackdata(idtrfile)%fileName
01225      strout%thisStorm = besttrackdata(idtrfile)%thisStorm
01226      strout%loaded = .true.
01227      strout%numRec = numrec
01228
01229      totrecpercycle = 0
01230
01231      WRITE(scratchmessage, '(a)') 'Starting the population of the best track structure variables ...'
01232      CALL logmessage(info, scratchmessage)
01233
01234      ncycles = 1
01235      totrecpercycle(ncycles) = 1
01236
01237      DO icnt = 1, numrec
01238
01239          casttype = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))
01240
01241          strout%basin(icnt) = besttrackdata(idtrfile)%basin(icnt)
01242          strout%stormNumber(icnt) = besttrackdata(idtrfile)%cyNum(icnt)
01243          strout%dtg(icnt) = besttrackdata(idtrfile)%dtg(icnt)
01244          strout%year(icnt) = besttrackdata(idtrfile)%year(icnt)
01245          strout%month(icnt) = besttrackdata(idtrfile)%month(icnt)
01246          strout%day(icnt) = besttrackdata(idtrfile)%day(icnt)
01247          strout%hour(icnt) = besttrackdata(idtrfile)%hour(icnt)
01248          strout%castTypeNum(icnt) = besttrackdata(idtrfile)%techNum(icnt)
01249          strout%castType(icnt) = touppercase(trim(adjustl(besttrackdata(idtrfile)%tech(icnt))))

```

```

01250     stout%fcstInc(icnt)      = besttrackdata(idtrfile)%tau(icnt)
01251     stout%ilat(icnt)        = besttrackdata(idtrfile)%intLat(icnt)
01252     stout%lat(icnt)         = besttrackdata(idtrfile)%lat(icnt)
01253     stout%ilon(icnt)        = besttrackdata(idtrfile)%intLon(icnt)
01254     stout%lon(icnt)         = besttrackdata(idtrfile)%lon(icnt)
01255     stout%ew(icnt)          = besttrackdata(idtrfile)%ew(icnt)
01256     stout%ns(icnt)          = besttrackdata(idtrfile)%ns(icnt)
01257     stout%ispeed(icnt)       = besttrackdata(idtrfile)%intVMax(icnt)
01258     stout%ispeed(icnt)       = kt2ms * stout%ispeed(icnt)           ! Convert speeds from knots to m/s
01259     stout%icPress(icnt)      = besttrackdata(idtrfile)%intMslp(icnt)
01260     stout%cPress(icnt)       = 100.0_sz * stout%icPress(icnt)
01261     stout%ty(icnt)          = besttrackdata(idtrfile)%ty(icnt)
01262     stout%ivr(icnt)          = besttrackdata(idtrfile)%rad(icnt)
01263     stout%windCode(icnt)     = besttrackdata(idtrfile)%windCode(icnt)
01264     stout%ir(icnt, 1)         = besttrackdata(idtrfile)%intRad1(icnt)
01265     stout%ir(icnt, 2)         = besttrackdata(idtrfile)%intRad2(icnt)
01266     stout%ir(icnt, 3)         = besttrackdata(idtrfile)%intRad3(icnt)
01267     stout%ir(icnt, 4)         = besttrackdata(idtrfile)%intRad4(icnt)
01268     stout%iprp(icnt)         = besttrackdata(idtrfile)%intPOuter(icnt)
01269     stout%prp(icnt)          = 100.0_sz * stout%iprp(icnt)           ! Convert pressure(s) from mbar to Pa
01270     stout%irRp(icnt)         = besttrackdata(idtrfile)%intROuter(icnt)
01271     stout%rrp(icnt)          = nm2m * stout%irRp(icnt)           ! Convert all distances from nm to m
01272     stout%irMw(icnt)         = besttrackdata(idtrfile)%intRmw(icnt)
01273     stout%rmw(icnt)          = nm2m * stout%irMw(icnt)
01274     stout%gusts(icnt)        = besttrackdata(idtrfile)%gusts(icnt)
01275     stout%eye(icnt)          = besttrackdata(idtrfile)%eye(icnt)
01276     stout%subregion(icnt)     = besttrackdata(idtrfile)%subregion(icnt)
01277     stout%maxseas(icnt)      = besttrackdata(idtrfile)%maxseas(icnt)
01278     stout%initials(icnt)     = besttrackdata(idtrfile)%initials(icnt)
01279     stout%idir(icnt)          = besttrackdata(idtrfile)%dir(icnt)
01280     stout%dir(icnt)          = real(stout%dir(icnt))
01281     stout%istormSpeed(icnt)   = besttrackdata(idtrfile)%intSpeed(icnt)
01282     stout%stormSpeed(icnt)    = kt2ms * stout%istormSpeed(icnt)           ! Convert speeds from knots to m/s
01283     stout%stormName(icnt)     = besttrackdata(idtrfile)%stormName(icnt)
01284
01285
01286 !PV DO WE NEED TO INCLUDE THE SAME CODE FOR CASTTIME AS IN HOLLAND?
01287     CALL timeconv(stout%year(icnt), stout%month(icnt), stout%day(icnt), stout%hour(icnt), 0, 0.0_sz,
01288     casttime(icnt))
01289     stout%castTime(icnt) = casttime(icnt)
01290
01291 !----- Check for a new cycle
01292 IF (icnt /= 1) THEN
01293     IF (stout%fcstInc(icnt) /= stout%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 stout%nCycles = ncycles
01303 !strOut%numCycle(iCnt) = nCycles
01304 stout%numCycle(icnt) = besttrackdata(idtrfile)%cycleNum(icnt)
01305 cycletime(icnt) = stout%fcstInc(icnt) * 3600.0_sz
01306 END DO ! numRec
01307
01308 ! Calculate the translation velocity in m/s and knots,
01309 ! Set background pressure
01310 DO icnt = 1, numrec
01311     ! Set iprp to background pressure
01312     stout%iprp(icnt) = nint(backgroundatmpress)      ! in mbar
01313     stout%prp(icnt) = 100.0_sz * stout%iprp(icnt) ! in Pa
01314
01315     ! Check/set central pressure
01316     IF (stout%icPress(icnt) == 0) THEN
01317         IF (icnt == 1) THEN
01318             WRITE(scratchmessage, '(a)') &
01319                 'Central pressure set to zero on first line/record when processing the best track file: ' // &
01320                 trim(adjustl(besttrackfilename(idtrfile)))
01321             CALL allmessage(error, scratchmessage)
01322             CALL terminate()
01323         ELSE
01324             CALL allmessage(warning, "Central pressure persisted from previous value.")
01325             stout%icPress(icnt) = stout%icPress(icnt - 1)
01326             stout%cPress(icnt) = stout%cPress(icnt - 1)
01327         END IF
01328     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%iSpeed(icnt), sz)
01427         pn   = real(strout%iPrp(icnt), sz)
01428         pc   = real(strout%icPress(icnt), sz)
01429         clat = strout%lat(icnt)
01430         clon = strout%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) = strout%ir(icnt, 4)
01442         lookupradii(5) = strout%ir(icnt, 1)
01443         radiisum = 0
01444         numnonzero = 0
01445
01446         DO i=1, 4
01447             lookupradii(i) = strout%ir(icnt,i)
01448             radiisum = radiisum + strout%ir(icnt,i)
01449             IF (strout%ir(icnt, i) > 0) THEN
01450                 numnonzero = numnonzero + 1
01451                 strout%quadFlag(icnt, i) = 1 ! use the Rmax resulting from this
01452             ELSE
01453                 strout%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(numnonzero)
01459             CASE(0) ! no isotachs reported, use overall Rmax; set isotach to vMax
01460                 strout%quadFlag(icnt, :) = 1
01461                 IF (strout%irRmw(icnt) /= 0) THEN
01462                     irad(icnt, :) = strout%irRmw(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(strout%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(strout%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 (strout%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      strout%hollB(icnt)      = 1.0_sz
01492      strout%hollBs(icnt, 1:4) = 1.0_sz
01493      phifactors(icnt, 1:4)   = 1.0_sz
01494      irr                     = strout%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 thta 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                  vmaxpseudo = vr
01545                  vmwb1(i) = sqrt((vmaxpseudo * cos(quadrantangles(i) + (deg2rad * 90.0_sz)) - &
01546                      stormmotionu)**2.0_sz + &
01547                      (vmaxpseudo * 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*vmwb1(i)**2.0_sz * windreduction**2.0_sz) * &
01553                              vr**2.0_sz)) / (2.0_sz * (stormmotion**2.0_sz - &
01554                                  vmwb1(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
01560              ELSE
01561                  vmwb1(i) = vmaxbl
01562                  ! gamma = |quadrantVr| / |vMaxBL|
01563                  gamma(i) = ((2.0_sz * u_vr * stormmotionu+2.0_sz * v_vr * stormmotionv) - &
01564                      sqrt((2.0_sz * u_vr * stormmotionu+2.0_sz * v_vr * &
01565                          stormmotionv)**2.0_sz - &
01566                          4.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * &
01567                              windreduction**2.0_sz) * vr**2.0_sz)) / &
01568                      (2.0_sz * (stormmotion**2.0_sz-vmaxbl**2.0_sz * windreduction**2.0_sz))
01569                  gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)

```

```

01570     quadrantvr(i) = (vr - gamma(i) * stormmotion) / windreduction !scalar cal.
01571     END IF
01572   ELSE
01573     vmwbl(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(rmaxwttemp)
01589 strout%rMaxW(icnt, :) = rmaxwttemp(:)
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 vmwblflag = 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. (vmwblflag(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   &
01639   &
01640   &
01641   &
01642   &
01643   &
01644   &
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. (vmwblflag(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) vmwblflag(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
01674                  ELSE
01675                      vmwbl(i) = vmaxbl
01676                      u_vr = vr * sin(strout%dir(icnt) * deg2rad)
01677                      v_vr = vr * sin(strout%dir(icnt) * deg2rad)
01678
01679                      ! gamma = |quadrantVr| / |vMaxBL|
01680                      gamma(i) = ((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr * stormmotionv) -
01681
01682                          &                                         sqrt((2.0_sz * u_vr * stormmotionu + 2.0_sz * v_vr *
01683                                         stormmotionv)**2.0_sz - &
01684                                         4.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz *
01685                                         &
01686                                         windreduction**2.0_sz) * vr**2.0_sz)) /
01687
01688                      &
01689                      (2.0_sz * (stormmotion**2.0_sz - vmaxbl**2.0_sz * windreduction**2.0_sz))
01690                      gamma(i) = max(min(gamma(i), 1.0_sz), 0.0_sz)
01691
01692                      quadrantvr(i) = (vr - gamma(i) * stormmotion) / windreduction !scalar cal.
01693
01694                  END IF
01695                  ELSE
01696                      vmwbl(i) = vmaxbl
01697                  END IF
01698                  END DO
01699
01700                  CALL setusequadrantvr(.true.)
01701                  CALL newvortexfull(pn, pc, clat, clon, vmaxbl)
01702                  strout%hollB(icnt) = getshapeparameter()
01703                  CALL setisotachwindspeeds(quadrantvr)
01704                  CALL setisotachradii(r)
01705                  CALL setvmaxesbl(vmwbl)
01706
01707                  IF (geostrophicswitch .EQV. .true.) THEN
01708                      CALL calcrmaxesfull()
01709
01710                  ELSE
01711                      CALL calcrmaxes()
01712                  END IF
01713                  CALL getrmaxes(rmaxwtemp)
01714                  strout%rMaxW(icnt, :) = rmaxwtemp(:)
01715
01716                  ! add deterministic statement to exit the loop when conditions met
01717                  !IF (k == 1) rMaxWHighIso(:) = strOut%rMaxW(iCnT, :) !PV This should be at the beginning of
01718                  ! the loop
01719
01720                  DO i = 1, 4
01721                      quadrotateangle_new(i) = fang(r(i), rmaxwhighiso(i))
01722                      IF (abs(quadrotateangle_new(i) - quadrotateangle(i)) > 0.2_sz) THEN
01723                          vioflag(i) = .true.
01724                      END IF
01725                  END DO
01726
01727                  IF ((count(vioflag) >= 1) .AND. (iquadrot < nquadrot)) THEN
01728                      quadrotateangle(:) = quadrotateangle_new(:)
01729
01730                  ELSE
01731                      EXIT
01732                  END IF
01733
01734                  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%hollBs(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(firstentry, :) = 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+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(iCn,:)>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
01889         quadrants.'
01890         CALL allmessage(error, scratchmessage)
01891         CALL allmessage(error, 'Execution terminated.')
01892     END SELECT
01893
01894     ! Persist last good 34kt Rmax values if all radii are missing
01895     DO icyc = 1, ncycles
01896         IF (totrecpercycle(icyc) == 1) THEN
01897             icnt = sum(totrecpercycle(1:icyc))
01898             IF ((all(strout%ir(icnt, :) == 0)) .AND. (strout%iRmw(icnt) == 0)) THEN
01899                 IF ((icyc - 1) >= 1) THEN
01900                     strout%rMaxW(icnt, :) = strout%rMaxW(icnt - totrecpercycle(icyc - 1), :)
01901                 ELSE
01902                     strout%rMaxW(icnt, :) = 25 ! default value when all else fails
01903                 END IF
01904             END IF
01905         END IF
01906     END DO
01907
01908     !-----!
01909     ! Here convert the hurricane speeds to be stored in the data structure
01910     strout%iStormSpeed = nint(strout%stormSpeed)
01911     strout%stormSpeed = kt2ms * strout%stormSpeed
01912     strout%idir = nint(strout%dir)
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                             nbtrfiles, besttrackfilename,
01941                             noutdt, mdbegsimtime, mdoutdt,
01942                             refyear, refmonth, refday, refhour, refmin, refsec,
01943                             times, datetimes,
01944                             wvelx, wvely, wpress
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     (Pa)
01960     REAL(SZ) :: trVX, trVY, trSPD ! storm translation velocities (m/s)
01961     REAL(SZ) :: trSpdX, trSpdy ! adjusted translation velocities (m/s)
01962     REAL(SZ) :: lon, lat ! current eye location
01963
01964     REAL(SZ), ALLOCATABLE :: rad(:) ! distance of nodal points from the eye
01965

```

```

01984      INTEGER, ALLOCATABLE          :: radIDX(:)           ! indices of nodal points duch that rad <=
01985      rrp
01985      INTEGER                      :: maxRadIDX        ! total number of radIDX elements
01986      REAL(SZ)                     :: windMultiplier    ! for storm 2 in lpfs ensemble DO WE NEED
01987      THIS?
01987      REAL(SZ)                     :: dx, dy, theta
01988      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
01992      of ABL)
01993      REAL(SZ)                     :: sfVelX, sfVelY   ! calculated surface (10-m above ground)
01993      wind velocities (m/s)
01994
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,

```

```

refmin, refsec)
02061     CALL juldaytoggreg(jday, iyear, imonth, iday, ihour, imin, isec)
02062     datestimes(icnt) = trim(adjustl(datetime2string(iyear, imonth, iday, ihour, imin, isec, 0)))
02063 END DO
02064 !-----
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 deallochollstruct(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 deallochollstruct(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 firstcall = .false.
02123
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(tmpimestr, '(a)') datestimes(icnt)
02151   WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(trimstr1)) // " " //
trim(adjustl(trimimestr))
02152 CALL allmessage(scratchmessage)
02153
02154 DO stcnt = 1, nbtrfiles
02155   ! Get the bin interval where Times(icnt) is bounded and the corresponding ratio
02156   ! factor for the subsequent linear interpolation in time. In order for this to
02157   ! work, the array holStru%castTime should be ordered in ascending order.
02158   CALL getlocandratio(times(icnt), holstru(stcnt)%castTime, j11, j12, wtratio)
02159
02160   ! Skip the subsequent calculations if Times(icnt) is outside the castTime range
02161   ! by exiting this loop
02162   IF ((j11 <= 0) .OR. (j12 <= 0)) THEN
02163     WRITE(scratchmessage, '(a)') 'Requested output time: ' // trim(adjustl(trimimestr)) // &
02164           ', skipping generating data for this time'
02165   CALL logmessage(info, scratchmessage)
02166
02167   cycle
02168 END IF
02169
02170   ! Perform linear interpolation in time
02171   stormnumber = holstru(stcnt)%stormNumber(j11)
02172
02173   CALL sphericalfracpoint(holstru(stcnt)%lat(j11), holstru(stcnt)%lon(j11), &
02174                 holstru(stcnt)%lat(j12), holstru(stcnt)%lon(j12), &
02175                 wtratio, lat, lon)
02176   !lat = holStru(stCntr)%lat(j11) + &
02177   !      wtRatio * (holStru(stCntr)%lat(j12) - holStru(stCntr)%lat(j11))
02178   !lon = holStru(stCntr)%lon(j11) + &
02179   !      wtRatio * (holStru(stCntr)%lon(j12) - holStru(stCntr)%lon(j11))
02180
02181   ! Radius of the last closed isobar
02182   rrp = holstru(stcnt)%rrp(j11) + &
02183         wtratio * (holstru(stcnt)%rrp(j12) - holstru(stcnt)%rrp(j11))
02184
02185   ! Radius of maximum winds
02186   rmw = holstru(stcnt)%rmw(j11) + &
02187         wtratio * (holstru(stcnt)%rmw(j12) - holstru(stcnt)%rmw(j11))
02188
02189   ! Get all the distances of the mesh nodes from (lat, lon)
02190   rad = sphericaldistance(sfua, slam, lat, lon)
02191   ! ... and the indices of the nodal points where rad <= rrp
02192   radidx = pack([(i, i = 1, np)], rad <= rrp)
02193   maxradidx = SIZE(radidx)
02194
02195   ! If the condition rad <= rrp is not satisfied anywhere then exit this loop
02196   IF (maxradidx == 0) THEN
02197     WRITE(tmpstr1, '(f20.3)') rrp
02198     tmpstr1 = '(rrp = ' // trim(adjustl(tmpstr1)) // ' m)'
02199     WRITE(scratchmessage, '(a)') 'No nodal points found inside the radius of the last closed isobar '
02200
02201   trim(adjustl(tmpstr1)) // ' for storm: ' // &
02202   trim(adjustl(holstru(stcnt)%thisStorm))
02203   CALL logmessage(info, scratchmessage)
02204
02205   EXIT
02206 END IF
02207
02208   speed = holstru(stcnt)%speed(j11) + &
02209             wtratio * (holstru(stcnt)%speed(j12) - holstru(stcnt)%speed(j11))
02210
02211   cpress = holstru(stcnt)%cPress(j11) + &
02212             wtratio * (holstru(stcnt)%cPress(j12) - holstru(stcnt)%cPress(j11))
02213
02214   trvx = holstru(stcnt)%trVx(j11) + &
02215             wtratio * (holstru(stcnt)%trVx(j12) - holstru(stcnt)%trVx(j11))
02216   trvy = holstru(stcnt)%trVy(j11) + &
02217             wtratio * (holstru(stcnt)%trVy(j12) - holstru(stcnt)%trVy(j11))

```

```

02217
02218 ! If this is a "CALM" period, set winds to zero velocity and pressure equal to the
02219 ! background pressure and return. PV: check if this is actually needed
02220 IF (cpress < 0.0_sz) THEN
02221   wvelx = 0.0_sz
02222   wvely = wvelx
02223   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 wind tapers
02280   ! to zero toward the eye of the storm and at long distances from the storm.
02281   trspd = (abs(grvel) / speed) * trvx
02282   trspd = (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 + trspdxd
02302      sfvely = sfvely + trspdy
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
02356      SUBROUTINE getgahmfields(timeIDX)
02357
02358      USE pahm_mesh, ONLY      : slam, sfea, np, ismeshok
02359      USE pahm_global, ONLY    : backgroundatmpress, one2ten,
02360                                deg2rad, rad2deg, basee, omega, kt2ms, mb2pa, mb2kpa, m2nm, nm2m, rearth, &
02361                                nbtrfiles, besttrackfilename,
02362                                noutdt, mdbegsimtime, mdoutdt,
02363                                refyear, refmonth, refday, refhour, refmin, refsec,
02364                                times, datestimes,
02365                                wvelx, wvely, wpress
02366      USE utilities, ONLY     : touppercase, sphericaldistance, sphericalfracpoint, getlocandratio
02367      USE timedateutils, ONLY  : juldaytogram, gregtojulday, gettimeconvsec, datetimetostring, timeconv
02368      USE pahm_vortex, ONLY    : spinterp, fitrmaxes4, rmaxes4, quadflag4, quadir4, bs4, vmb14, &
02369                                setvortex, uvpr
02370
02371      IMPLICIT NONE
02372
02373      INTEGER, INTENT(IN)          :: timeIDX
02374
02375      REAL(SZ)                      :: coriolis ! Coriolis force (1/s)
02376
02377      INTEGER                         :: iCnt, kCnt, stCnt
02378      INTEGER                         :: i, j11, j12
02379      INTEGER                         :: status
02380
02381      REAL(SZ)                      :: jday
02382      INTEGER                         :: iYear, iMonth, iDay, iHour, iMin, iSec
02383
02384      CHARACTER(LEN=64)              :: tmpTimeStr, tmpStr1, tmpStr2
02385
02386
02387      INTEGER                         :: maxTrackRecords
02388      INTEGER, SAVE                  :: iCyc, iSot ! do we need to save this?
02389      INTEGER, DIMENSION(:, :, :, :, :), ALLOCATABLE, SAVE :: ir, quadFlag
02390      REAL(SZ), DIMENSION(:, :, :, :, :), ALLOCATABLE, SAVE :: rMaxW, hollBs, vMaxesBL
02391
02392      REAL(SZ)                      :: stormMotion ! portion of Vmax attributable to storm motion
02393      REAL(SZ)                      :: stormMotionU ! U portion of Vmax attributable to storm motion
02394      REAL(SZ)                      :: stormMotionV ! V portion of Vmax attributable to storm motion
02395
02396      CHARACTER(LEN = 10), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: stormName
02397      CHARACTER(LEN = 4), DIMENSION(:, :, :), ALLOCATABLE, SAVE :: castType ! hindcast/nowcast or forecast?
02398      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: stormNumber ! storm identification number
02399      INTEGER, DIMENSION(:, :, :), ALLOCATABLE, SAVE :: year, month, day, hour

```

```

02400    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE :: iRmw
02401    REAL(SZ), DIMENSION(:, :, ), ALLOCATABLE, SAVE :: lat, lon
02402    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE :: iSpeed, iCPress
02403    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE :: fcstInc ! hours between forecasts
02404    REAL(SZ), DIMENSION(:, :, ), ALLOCATABLE, SAVE :: hollB
02405    INTEGER, DIMENSION(:, :, ), ALLOCATABLE, SAVE :: nCycles
02406    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        ELSE
02470            IF ((np <= 0) .OR. (noutdt <= 0)) THEN
02471                WRITE(tmpstr1, '(a, i0)') 'np = ', np
02472                WRITE(tmpstr2, '(a, i0)') 'nOutDT = ', noutdt
02473                WRITE(scratchmessage, '(a)') 'Variables "np" or "nOutDT" are not defined properly: '
02474                trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
02475                CALL allmessage(error, scratchmessage)
02476
02477            CALL unsetmessagesource()
02478
02479            CALL terminate()

```

```

02480      END IF
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(cvmwbl1(np), cvmwbl2(np))
02512 ALLOCATE(crmmaxwl(np), crmaxw2(np))
02513 ALLOCATE(crmaxwtrue1(np), crmaxwtrue2(np))
02514 ALLOCATE(chollbs(np))
02515 ALLOCATE(cphifactor(np))
02516 ALLOCATE(cvmwbl(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 allmessage(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 deallocasymvortstruct(asyvortstru(stcnt))
02554         DEALLOCATE(asyvortstru)
02555

```

```

02556      CALL unsetmessagesource()
02557
02558      CALL terminate()
02559      ELSE
02560      WRITE(scratchmessage, '(a)') 'Processing the asymmetric vortex data structure for the best track
file: ' // &
02561                  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(irnw(nbtrfiles,                maxtrackrecords))
02577      ALLOCATE(ispeed(nbtrfiles,               maxtrackrecords))
02578      ALLOCATE(icpress(nbtrfiles,              maxtrackrecords))
02579      ALLOCATE(fcstinc(nbtrfiles,              maxtrackrecords))
02580      ALLOCATE(hollbs(nbtrfiles,               maxtrackrecords))
02581      ALLOCATE(ncycles(nbtrfiles),            maxtrackrecords)
02582      ALLOCATE(numcycle(nbtrfiles,             maxtrackrecords))
02583      ALLOCATE(isotachpercycle(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 ! Initialize the variables
02601 !-----
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 kcnt = 1, asyvortstru(stcnt)%numRec
02615          ! pick out the cycle data that the user wants to use
02616          ! by looping through quads
02617          IF (kcnt == 1) THEN
02618              icyc = 1
02619              isot = 1
02620
02621              stormnumber(stcnt, icyc) = asyvortstru(stcnt)%stormNumber(kcnt)
02622              year(stcnt, icyc)        = asyvortstru(stcnt)%year(kcnt)
02623              month(stcnt, icyc)       = asyvortstru(stcnt)%month(kcnt)
02624              day(stcnt, icyc)         = asyvortstru(stcnt)%day(kcnt)
02625              hour(stcnt, icyc)        = asyvortstru(stcnt)%hour(kcnt)
02626              casttype(stcnt, icyc)    = asyvortstru(stcnt)%castType(kcnt)
02627              fcstinc(stcnt, icyc)     = asyvortstru(stcnt)%fcstInc(kcnt)
02628              lat(stcnt, icyc)         = asyvortstru(stcnt)%lat(kcnt)
02629              lon(stcnt, icyc)         = asyvortstru(stcnt)%lon(kcnt)
02630              ispeed(stcnt, icyc)       = asyvortstru(stcnt)%iSpeed(kcnt)
02631              icpress(stcnt, icyc)      = asyvortstru(stcnt)%iCPress(kcnt)
02632              ivr(stcnt, icyc)          = asyvortstru(stcnt)%ivr(kcnt)
02633              ipn(stcnt, icyc)          = asyvortstru(stcnt)%iPrp(kcnt)
02634              irmw(stcnt, icyc)         = asyvortstru(stcnt)%iRmw(kcnt)
02635

```

```

02636      hdir(stcnt, icyc)      = asyvortstru(stcnt)%iDir(kcnt)
02637      hspeed(stcnt, icyc)    = asyvortstru(stcnt)%iStormSpeed(kcnt)
02638      stormname(stcnt, icyc) = asyvortstru(stcnt)%stormName(kcnt)
02639      numcycle(stcnt, icyc)   = asyvortstru(stcnt)%numCycle(kcnt)
02640      hollb(stcnt, icyc)     = asyvortstru(stcnt)%hollB(kcnt)
02641
02642      utrans(stcnt, icyc)    = asyvortstru(stcnt)%trVx(kcnt)
02643      vtrans(stcnt, icyc)    = asyvortstru(stcnt)%trVy(kcnt)
02644
02645      CALL timeconv(year(stcnt, icyc), month(stcnt, icyc), day(stcnt, icyc), hour(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          END IF
02658      END DO
02659
02660      ELSE ! kCnT == 1
02661          IF (asyvortstru(stcnt)%numCycle(kcnt) == asyvortstru(stcnt)%numCycle(kcnt-1)) THEN
02662              IF (isot > 4) THEN
02663                  WRITE(scratchmessage,'(a, i0, a)') &
02664                      'The GAHM asymmetric data structure has more than 4 iSotachs in cycle ', &
02665                      asyvortstru(stcnt)%numCycle(kcnt), '.'
02666                  CALL allmessage(error, scratchmessage)
02667                  CALL terminate()
02668              END IF
02669              isot = isot + 1 ! same iCyc, next iSotach
02670
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      hour(stcnt, icyc)        = asyvortstru(stcnt)%hour(kcnt)
02685      casttype(stcnt, icyc)    = asyvortstru(stcnt)%castType(kcnt)
02686      fcstinc(stcnt, icyc)     = asyvortstru(stcnt)%fcstInc(kcnt)
02687      lat(stcnt, icyc)         = asyvortstru(stcnt)%lat(kcnt)
02688      lon(stcnt, icyc)         = asyvortstru(stcnt)%lon(kcnt)
02689      ispeed(stcnt, icyc)       = asyvortstru(stcnt)%iSpeed(kcnt)
02690      icpress(stcnt, icyc)      = asyvortstru(stcnt)%iCPress(kcnt)
02691      ivr(stcnt, icyc)         = asyvortstru(stcnt)%ivr(kcnt)
02692      ipn(stcnt, icyc)         = asyvortstru(stcnt)%iPrp(kcnt)
02693      irmw(stcnt, icyc)        = asyvortstru(stcnt)%iRmw(kcnt)
02694      hdir(stcnt, icyc)        = asyvortstru(stcnt)%iDir(kcnt)
02695      hspeed(stcnt, icyc)      = asyvortstru(stcnt)%iStormSpeed(kcnt)
02696      stormname(stcnt, icyc)    = asyvortstru(stcnt)%stormName(kcnt)
02697      numcycle(stcnt, icyc)     = asyvortstru(stcnt)%numCycle(kcnt)
02698      hollb(stcnt, icyc)       = asyvortstru(stcnt)%hollB(kcnt)
02699
02700      utrans(stcnt, icyc)      = asyvortstru(stcnt)%trVx(kcnt)
02701      vtrans(stcnt, icyc)      = asyvortstru(stcnt)%trVy(kcnt)
02702
02703      CALL timeconv(year(stcnt, icyc), month(stcnt, icyc), day(stcnt, icyc), hour(stcnt, icyc), &
02704          0, 0, cycletime(stcnt, icyc))
02705
02706      isotachspercycle(stcnt, icyc) = asyvortstru(stcnt)%isotachsPerCycle(kcnt)
02707
02708      DO i = 1, 4
02709          IF (asyvortstru(stcnt)%quadFlag(kcnt, i) == 1) THEN
02710              ir(stcnt, icyc, i, isot)      = asyvortstru(stcnt)%ir(kcnt, i)
02711              rmaxw(stcnt, icyc, i, isot)    = asyvortstru(stcnt)%rMaxW(kcnt, i)
02712              quadflag(stcnt, icyc, i, isot) = asyvortstru(stcnt)%quadFlag(kcnt, i)
02713              hollbs(stcnt, icyc, i, isot)  = asyvortstru(stcnt)%hollBs(kcnt, i)
02714              vmaxesbl(stcnt, icyc, i, isot) = asyvortstru(stcnt)%vMaxesBL(kcnt, i)
02715          END IF
02716      END DO
02716      END IF ! kCnT /= 1

```

```

02717      END DO ! kCnt = 1, asyVortStru(stCnt)%numRec
02718
02719      ncycles(stcnt) = asyvortstru(stcnt)%nCycles
02720  END DO ! nBTrFiles
02721
02722      firstcall = .false.
02723  END IF
02724 ! ##### FIRSTCALL BLOCK
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 ! THIS IS THE MAIN TIME LOOP
02740 ! IT USES "timeIDX" TO ADVANCE THE CALCULATIONS IN TIME
02741 !-----
02742 icnt = timeidx
02743     WRITE(tmpstr1, '(i5)') icnt
02744     WRITE(tmpstr2, '(i5)') noutdt
02745 tmpstr1 = '(' // trim(tmpstr1) // '/' // trim(adjustl(trimstr2)) // ')'
02746     WRITE(tmptimestr, '(a)') datestimes(icnt)
02747 WRITE(scratchmessage, '(a)') 'Working on time frame: ' // trim(adjustl(trimstr1)) // " " //
02748 trim(adjustl(tmptimestr))
02749     CALL allmessage(scratchmessage)
02750
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(tmptimestr)) // &
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 firmaxes4()

```

```

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)      = holls(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     cholls  = 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      !-----
02894      ! 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
02895      !-----
02911      !-----
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_btrkl
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%intMslp(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(ix, i4, ""), 1x, i4, "", 1x, i4, "", 1x, i3, "", 1x, i3,
",",
03045 fmtstr = trim(fmtstr) // ' 1x, a3, "", 1x, i3, "", 1x, a3, "", 1x, i3, "", 1x, i3, "",'
03046 fmtstr = trim(fmtstr) // ' 1x, a10, "", 1x, i4, "",'
03047 fmtstr = trim(fmtstr) // ' 1x, i4, "", 4(ix, i1, ""), 2x, 4(ix, f6.1, ""), 9(ix, f8.4, "")'
03048 !
03049
03050 fsuf = '_asymvort'
03051 IF (PRESENT(suffix)) fsuf = adjustl(suffix)
03052
03053 CALL setmessagesource("WriteAsymmetricVortexData")
03054
03055 IF (.NOT. trackstruc%loaded) THEN
03056   WRITE(scratchmessage, '(a)') "The input best track structure is empty. Best track data won't be
written."
03057   CALL allmessage(info, scratchmessage)
03058
03059   RETURN
03060 END IF
03061
03062 outfile = trim(adjustl(inpfile)) // trim(fsuf)
03063
03064 WRITE(scratchmessage, '(a)') 'Writting the "extended" best track data to: ' // trim(adjustl(outfile))

```

```

03065     CALL logmessage(info, scratchmessage)
03066
03067     OPEN(unit=iunit, file=trim(outfile), status='REPLACE', action='WRITE', iostat=errio)
03068
03069     IF (errio /= 0) THEN
03070         WRITE(scratchmessage, '(a)') 'Error opening the outFile: ' // trim(outfile) // &
03071             ', skip writing 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%hol1B(icnt),    (trackstruc%hol1Bs(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 !-----+
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))

```

```

03158   IF (.NOT. ALLOCATED(str%eye))      ALLOCATE(str%eye(nrec))
03159   IF (.NOT. ALLOCATED(str%subregion)) ALLOCATE(str%subregion(nrec))
03160   IF (.NOT. ALLOCATED(str%maxseas))   ALLOCATE(str%maxseas(nrec))
03161   IF (.NOT. ALLOCATED(str%initials))  ALLOCATE(str%initials(nrec))
03162   IF (.NOT. ALLOCATED(str%dir))       ALLOCATE(str%dir(nrec))
03163   IF (.NOT. ALLOCATED(str%intSpeed))  ALLOCATE(str%intSpeed(nrec))
03164   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 !-----
03180 ! 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
03181 !
03192 !
03193 SUBROUTINE deallocbtrstruct(str)
03194
03195 IMPLICIT NONE
03196
03197 TYPE(besttrackdata_t), INTENT(INOUT) :: str
03198
03199 str%numRec = -1
03200 str%loaded = .false.
03201
03202 !----- Input parameters
03203 IF (ALLOCATED(str%basin))      DEALLOCATE(str%basin)
03204 IF (ALLOCATED(str%cyNum))      DEALLOCATE(str%cyNum)
03205 IF (ALLOCATED(str%dtg))       DEALLOCATE(str%dtg)
03206 IF (ALLOCATED(str%techNum))   DEALLOCATE(str%techNum)
03207 IF (ALLOCATED(str%tech))     DEALLOCATE(str%tech)
03208 IF (ALLOCATED(str%tau))      DEALLOCATE(str%tau)
03209 IF (ALLOCATED(str%intLat))   DEALLOCATE(str%intLat)
03210 IF (ALLOCATED(str%intLon))   DEALLOCATE(str%intLon)
03211 IF (ALLOCATED(str%ew))       DEALLOCATE(str%ew)
03212 IF (ALLOCATED(str%ns))       DEALLOCATE(str%ns)
03213 IF (ALLOCATED(str%intVMax))  DEALLOCATE(str%intVMax)
03214 IF (ALLOCATED(str%intMspl))  DEALLOCATE(str%intMspl)
03215 IF (ALLOCATED(str%ty))      DEALLOCATE(str%ty)
03216 IF (ALLOCATED(str%rad))     DEALLOCATE(str%rad)
03217 IF (ALLOCATED(str%windCode)) DEALLOCATE(str%windCode)
03218 IF (ALLOCATED(str%intRad1))  DEALLOCATE(str%intRad1)
03219 IF (ALLOCATED(str%intRad2))  DEALLOCATE(str%intRad2)
03220 IF (ALLOCATED(str%intRad3))  DEALLOCATE(str%intRad3)
03221 IF (ALLOCATED(str%intRad4))  DEALLOCATE(str%intRad4)
03222 IF (ALLOCATED(str%intPOuter)) DEALLOCATE(str%intPOuter)
03223 IF (ALLOCATED(str%intROuter)) DEALLOCATE(str%intROuter)
03224 IF (ALLOCATED(str%intRmw))   DEALLOCATE(str%intRmw)
03225 IF (ALLOCATED(str%gusts))   DEALLOCATE(str%gusts)
03226 IF (ALLOCATED(str%eye))    DEALLOCATE(str%eye)
03227 IF (ALLOCATED(str%subregion)) DEALLOCATE(str%subregion)
03228 IF (ALLOCATED(str%maxseas)) DEALLOCATE(str%maxseas)
03229 IF (ALLOCATED(str%initials)) DEALLOCATE(str%initials)
03230 IF (ALLOCATED(str%dir))    DEALLOCATE(str%dir)
03231 IF (ALLOCATED(str%intSpeed)) DEALLOCATE(str%intSpeed)
03232 IF (ALLOCATED(str%stormName)) DEALLOCATE(str%stormName)
03233 IF (ALLOCATED(str%cycleNum)) DEALLOCATE(str%cycleNum)
03234
03235 !----- Converted parameters
03236 IF (ALLOCATED(str%year))      DEALLOCATE(str%year)
03237 IF (ALLOCATED(str%month))    DEALLOCATE(str%month)
03238 IF (ALLOCATED(str%day))      DEALLOCATE(str%day)
03239 IF (ALLOCATED(str%hour))    DEALLOCATE(str%hour)
03240 IF (ALLOCATED(str%lat))     DEALLOCATE(str%lat)
03241 IF (ALLOCATED(str%lon))     DEALLOCATE(str%lon)
03242
03243 END SUBROUTINE deallocbtrstruct
03244
03245 !-----
03246 ! 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
03248

```

```

03249 ! -----
03262 ! -----
03263 SUBROUTINE allochollstruct(str, nRec)
03264
03265 IMPLICIT NONE
03266
03267 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%ilLon)) ALLOCATE(str%ilLon(nrec))
03290 IF (.NOT. ALLOCATED(str%lon)) ALLOCATE(str%lon(nrec))
03291
03292 IF (.NOT. ALLOCATED(str%isSpeed)) ALLOCATE(str%isSpeed(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)

```

```

03352   IF (ALLOCATED(str%iLon))           DEALLOCATE(str%iLon)
03353   IF (ALLOCATED(str%lon))           DEALLOCATE(str%lon)
03354
03355   IF (ALLOCATED(str%iSpeed))          DEALLOCATE(str%iSpeed)
03356   IF (ALLOCATED(str%speed))          DEALLOCATE(str%speed)
03357
03358   IF (ALLOCATED(str%icPress))         DEALLOCATE(str%icPress)
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 !-----
03379 !
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%iLon))              ALLOCATE(str%iLon(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%iSpeed))            ALLOCATE(str%iSpeed(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 allocasyvmortstruct
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%ccPress))        DEALLOCATE(str%ccPress)
03511
03512 IF (ALLOCATED(str%ty))             DEALLOCATE(str%ty)
03513
03514 IF (ALLOCATED(str%ivr))           DEALLOCATE(str%ivr)
03515 IF (ALLOCATED(str%windCode))       DEALLOCATE(str%windCode)
03516 IF (ALLOCATED(str%ir))             DEALLOCATE(str%ir)
03517
03518 IF (ALLOCATED(str%iprp))          DEALLOCATE(str%iprp)
03519 IF (ALLOCATED(str%prp))            DEALLOCATE(str%prp)
03520 IF (ALLOCATED(str%irrp))           DEALLOCATE(str%irrp)
03521 IF (ALLOCATED(str%rrp))            DEALLOCATE(str%rrp)
03522
03523 IF (ALLOCATED(str%irmw))          DEALLOCATE(str%irmw)
03524 IF (ALLOCATED(str%rmw))            DEALLOCATE(str%rmw)

```

```

03536
03537     IF (ALLOCATED(str%gusts))           DEALLOCATE(str%gusts)
03538     IF (ALLOCATED(str%eye))             DEALLOCATE(str%eye)
03539     IF (ALLOCATED(str%subregion))        DEALLOCATE(str%subregion)
03540     IF (ALLOCATED(str%maxseas))          DEALLOCATE(str%maxseas)
03541     IF (ALLOCATED(str%initials))         DEALLOCATE(str%initials)
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

```

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

Variables

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

17.35.1 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`.

17.36 sizes.F90

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

```

00001 !-----  

00002 !      M O D U L E   S I Z E S  

00003 !-----  

00013 !-----  

00014  

00015 MODULE pahm_sizes  

00016  

00017 IMPLICIT NONE  

00018  

00019 !-----  

00020 ! I N T E R F A C E S  

00021 !-----  

00022 INTERFACE comparereals  

00023     MODULE PROCEDURE comparesinglereals  

00024     MODULE PROCEDURE comparedoublereals  

00025 END INTERFACE comparereals  

00026  

00027 INTERFACE fixnearwholereal  

00028     MODULE PROCEDURE fixnearwholesinglereal  

00029     MODULE PROCEDURE fixnearwholedoublereal  

00030 END INTERFACE fixnearwholereal

```

```

00031 !-----
00032
00033 ! SP = single precision, HP = high (double) precision
00034 INTEGER, PARAMETER :: sp = selected_real_kind(6, 37) ! 6 digits, range \([10^{-37} , 10^{+37} - 1]\), 32 bits
00035 INTEGER, PARAMETER :: hp = selected_real_kind(15, 307) ! 15 digits, range \([10^{-307} , 10^{+307} - 1]\), 64 bits
00036
00037 ! Precision of integers:
00038 INTEGER, PARAMETER :: int16 = selected_int_kind(38) ! Range \([-2^{127}, +2^{127} - 1]\), 39 digits
00039 plus sign; 128 bits
00040 INTEGER, PARAMETER :: int8 = selected_int_kind(18) ! Range \([-2^{63}, +2^{63} - 1]\), 19 digits
00041 plus sign; 64 bits
00042 INTEGER, PARAMETER :: int4 = selected_int_kind(9) ! Range \([-2^{31}, +2^{31} - 1]\), 10 digits
00043 plus sign; 32 bits
00044 INTEGER, PARAMETER :: int2 = selected_int_kind(4) ! Range \([-2^{15}, +2^{15} - 1]\), 5 digits
00045 plus sign; 16 bits
00046 INTEGER, PARAMETER :: int1 = selected_int_kind(2) ! Range \([-2^7 , +2^7 - 1]\), 3 digits
00047 plus sign; 8 bits
00048
00049 INTEGER, PARAMETER :: wp = hp ! default real kind (for csv_module)
00050 INTEGER, PARAMETER :: ip = int8 ! default integer kind (for csv_module)
00051
00052 ! By default we perform all calculations in double precision
00053 ! SET NUMBER OF BYTES "SZ" IN REAL(SZ) DECLARATIONS
00054 ! SET "NBYTE" FOR PROCESSING INPUT DATA RECORD LENGTH
00055 #ifdef REAL4
00056 INTEGER, PARAMETER :: sz = sp
00057 INTEGER, PARAMETER :: nbytes = 4
00058 #endif
00059
00060 ! Used to initialize the mesh arrays and in NetCDF output files for missing values.
00061 ! Also used to initialize some input variables to check if these variables
00062 ! were supplied user defined values.
00063 REAL(sz), PARAMETER :: rmissv = -999999.0_sz
00064 INTEGER, PARAMETER :: imissv = -999999
00065
00066 CHARACTER(LEN=1), PARAMETER :: blank = ' '
00067
00068 ! Filename length (considers the presence of the full path in the filename)
00069 INTEGER, PARAMETER :: fnamelen = 1024
00070
00071
00072 CONTAINS
00073
00074
00075 !-----
00076 ! F U N C T I O N CompareDoubleReals
00077 !-----
00078
00079 !-----
00101 INTEGER FUNCTION comparedoublerels(rVal1, rVal2, eps) RESULT(myValOut)
00102
00103 IMPLICIT NONE
00104
00105 ! Global variables
00106 REAL(hp), INTENT(IN) :: rval1, rval2
00107 REAL(hp), OPTIONAL, INTENT(IN) :: eps
00108
00109 ! Local variables
00110 REAL(hp) :: epssys, epsusr, value
00111
00112
00113 epssys = 2.0_hp * epsilon(rval1)
00114
00115 IF (PRESENT(eps)) THEN
00116   epsusr = abs(eps)
00117 ELSE
00118   epsusr = epssys
00119 ENDIF
00120
00121 IF ((abs(rval1) < 1.0_hp) .OR. (abs(rval2) < 1.0_hp)) THEN
00122   value = rval1 - rval2
00123 ELSE
00124   value = (rval1 - rval2) / max(rval1, rval2)
00125   IF (abs(value) < 1.0_hp) value = rval1 - rval2
00126 END IF

```

```

00127
00128      IF (abs(value) < epsusr) THEN
00129          myvalout = 0
00130      ELSE IF (rval1 < rval2) THEN
00131          myvalout = -1
00132      ELSE
00133          myvalout = 1
00134      END IF
00135
00136      RETURN
00137
00138  END FUNCTION comparedoublereals
00139
00140 !=====
00141
00142 !-----
00143 !  F U N C T I O N   C O M P A R E   S I N G L E   R E A L S
00144 !-----
00167 !
00168 INTEGER FUNCTION comparesinglereals(rVal1, rVal2, eps) RESULT(myValOut)
00169
00170     IMPLICIT NONE
00171
00172     ! Global variables
00173     REAL(sp), INTENT(IN) :: rval1, rval2
00174     REAL(sp), OPTIONAL, INTENT(IN) :: eps
00175
00176     ! Local variables
00177     REAL(sp) :: epssys, epsusr, value
00178
00179
00180     epssys = 2.0_sp * epsilon(rval1)
00181
00182     IF (PRESENT(eps)) THEN
00183         epsusr = abs(eps)
00184     ELSE
00185         epsusr = epssys
00186     ENDIF
00187
00188     IF ((abs(rval1) < 1.0_sp) .OR. (abs(rval2) < 1.0_sp)) THEN
00189         value = rval1 - rval2
00190     ELSE
00191         value = (rval1 - rval2) / max(rval1, rval2)
00192     IF (abs(value) < 1.0_sp) value = rval1 - rval2
00193     END IF
00194
00195     IF (abs(value) < epsusr) THEN
00196         myvalout = 0
00197     ELSE IF (rval1 < rval2) THEN
00198         myvalout = -1
00199     ELSE
00200         myvalout = 1
00201     END IF
00202
00203     RETURN
00204
00205  END FUNCTION comparesinglereals
00206
00207 !=====
00208
00209 !-----
00210 !  F U N C T I O N   F I X   N E A R   W H O L E   D O U B L E   R E A L
00211 !-----
00234 !
00235 REAL(hp) function fixnearwholedoublereal(rval, eps) result(myvalout)
00236
00237     IMPLICIT NONE
00238
00239     ! Global Variables
00240     REAL(hp), INTENT(IN) :: rval
00241     REAL(hp), OPTIONAL, INTENT(IN) :: eps
00242
00243     ! Local Variables
00244     REAL(hp) :: epssys, epsusr, value
00245
00246
00247     epssys = 2.0_hp * epsilon(rval)
00248
00249     IF (PRESENT(eps)) THEN
00250         epsusr = abs(eps)
00251     ELSE

```

```

00252     epsusr = epssys
00253     ENDIF
00254
00255     myvalout = rval
00256     value    = anint(myvalout)
00257     IF (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 !-----+
00271 REAL(sp) function fixnearwholesinglereal(rval, eps) result(myvalout)
00272
00273     IMPLICIT NONE
00274
00275     ! Global Variables
00276     REAL(sp), INTENT(IN)      :: rval
00277     REAL(sp), OPTIONAL, INTENT(IN) :: eps
00278
00279     ! Local Variables
00280     REAL(sp)                  :: epssys, epsusr, value
00281
00282
00283     epssys = 2.0_sp * epsilon(rval)
00284
00285     IF (PRESENT(eps)) THEN
00286         epsusr = abs(eps)
00287     ELSE
00288         epsusr = epssys
00289     ENDIF
00290
00291     myvalout = rval
00292     value    = anint(myvalout)
00293     IF (comparereals(myvalout, value, epsusr) == 0) myvalout = value
00294
00295     RETURN
00296
00297 END FUNCTION fixnearwholesinglereal
00298
00299 !=====
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312
00313
00314
00315
00316
00317
00318
00319 !=====
00320
00321 END MODULE pahm_sizes

```

17.37 /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)
- 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").

17.37.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [sortutils.F90](#).

17.37.2 Function/Subroutine Documentation

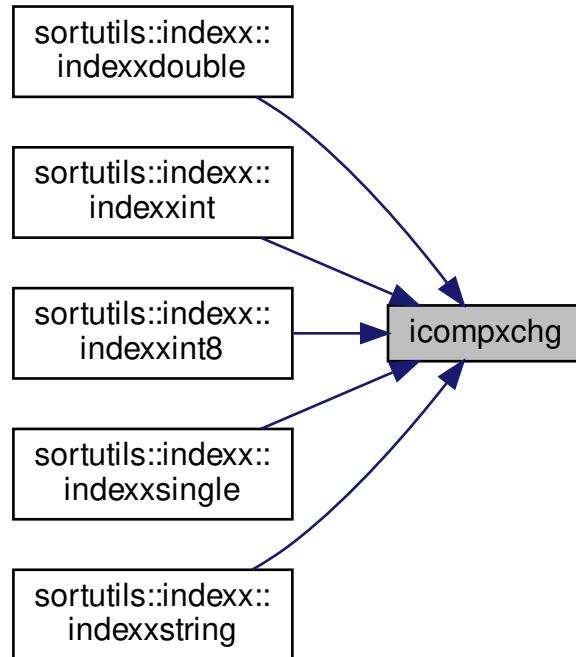
17.37.2.1 **icompxchg()**

```
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:



17.38 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 arrayeqaldouble
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 !-----
00067 !-----
00068 !-----
00069 SUBROUTINE indexxint(arr1D, idx1D, status)
00070
00071     IMPLICIT NONE
00072
00073     ! Global variables
00074     INTEGER, DIMENSION(:), INTENT(IN)    :: arr1D
00075     INTEGER, DIMENSION(:), INTENT(OUT)   :: idx1D
00076     INTEGER, OPTIONAL, INTENT(OUT)       :: status
00077
00078     ! Local variables
00079     INTEGER, PARAMETER                   :: NN = 15, nstack = 50
00080     INTEGER                               :: a
00081     INTEGER                               :: nARR, nIDX, tmpIDX
00082     INTEGER                               :: k, i, j, l, r
00083     INTEGER                               :: ist, stack(NSTACK)
00084     CHARACTER(LEN=64)                     :: tmpStr1, tmpStr2
00085
00086
00087     CALL setmessagesource("IndexxInt")
00088
00089     IF (PRESENT(status)) status = 0
00090
00091     narr = SIZE(arr1D, 1)
00092     nidx = SIZE(idx1D, 1)
00093
00094     IF (narr /= nidx) THEN
00095         WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00096         WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00097         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00098             trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00099
00100     CALL allmessage(error, scratchmessage)
00101     CALL unsetmessagesource()
00102
00103     IF (PRESENT(status)) status = 1
00104
00105     RETURN
00106 END IF
00107
00108     idx1D = arth(1, 1, narr)
00109
00110     ist = 0
00111     l   = 1
00112     r   = narr
00113
00114     DO
00115         IF (r - l < nn) THEN
00116             DO j = l + 1, r
00117                 tmpidx = idx1D(j)
00118
00119             CALL swap(idx1D, arr1D, l, j)
00120
00121             l = l + 1
00122
00123         END IF
00124
00125         r = r - 1
00126
00127     END DO
00128
00129     CALL swap(idx1D, arr1D, l, r)
00130
00131     IF (nn > 1) THEN
00132         CALL indexxint(arr1D, idx1D, status)
00133     END IF
00134
00135 END SUBROUTINE indexxint

```

```

00134      a = arrld(tmpidx)
00135      DO i = j - 1, 1, -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 icompxchg(idxld(l), idxld(r))
00156     CALL icompxchg(idxld(l + 1), idxld(r))
00157     CALL icompxchg(idxld(l), idxld(l + 1))
00158
00159  i = l + 1
00160  j = r
00161  tmpidx = idxld(l + 1)
00162  a = arrld(tmpidx)
00163
00164  DO
00165      DO
00166          i = i + 1
00167          IF (arrld(idxld(i)) > a) EXIT
00168      END DO
00169
00170      DO
00171          j = j - 1
00172          IF (arrld(idxld(j)) < a) EXIT
00173      END DO
00174
00175      IF (j < i) EXIT
00176      CALL swap(idxld(i), idxld(j))
00177  END DO
00178
00179  idxld(l + 1) = idxld(j)
00180  idxld(j) = tmpidx
00181  ist = ist + 2
00182
00183  IF (ist > nstack) THEN
00184      WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
00185      WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
00186      trim(adjustl(tmpstr1))
00187
00188      CALL logmessage(error, scratchmessage)
00189      CALL unsetmessagesource()
00190
00191      IF (PRESENT(status)) status = 2
00192
00193      RETURN
00194
00195  END IF
00196
00197  IF (r - i + 1 >= j - 1) THEN
00198      stack(ist) = r
00199      stack(ist - 1) = i
00200      r = j - 1
00201  ELSE
00202      stack(ist) = j - 1
00203      stack(ist - 1) = l
00204      l = i
00205  END IF
00206  END IF
00207 END DO
00208
00209  CALL unsetmessagesource()
00210
00211  CONTAINS
00212
00213  SUBROUTINE icompxchg(i, j)

```

```

00215      IMPLICIT NONE
00216
00217
00218 ! Global variables
00219      INTEGER, INTENT(INOUT) :: i, j
00220
00221 ! Local variables
00222      INTEGER :: swp
00223
00224      IF (arr1d(j) < arr1d(i)) THEN
00225          swp = i
00226          i    = j
00227          j    = swp
00228      END IF
00229
00230      END SUBROUTINE icompxchg
00231
00232      END SUBROUTINE indexxint
00233
00234 !=====
00235
00236 !-----+
00237 ! S U B R O U T I N E   I N D E X X   I N T   8
00238 !-----+
00239
00240 !-----
00241      SUBROUTINE indexxint8(arr1D, idx1D, status)
00242
00243      IMPLICIT NONE
00244
00245 ! Global variables
00246      INTEGER(INT8), DIMENSION(:), INTENT(IN)  :: arr1D
00247      INTEGER, DIMENSION(:), INTENT(OUT)       :: idx1D
00248      INTEGER, OPTIONAL, INTENT(OUT)           :: status
00249
00250 ! Local variables
00251      INTEGER, PARAMETER                      :: NN = 15, nstack = 50
00252      INTEGER(INT8)                           :: a
00253      INTEGER                                 :: nARR, nIDX, tmpIDX
00254      INTEGER                                 :: k, i, j, l, r
00255      INTEGER                                 :: ist, stack(NSTACK)
00256      CHARACTER(LEN=64)                        :: tmpStr1, tmpStr2
00257
00258
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
00267          WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00268          WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00269          WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00270                                      trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00271
00272      CALL allmessage(error, scratchmessage)
00273      CALL unsetmessagesource()
00274
00275      IF (PRESENT(status)) status = 1
00276
00277      RETURN
00278  END IF
00279
00280      idx1D = arth(1, 1, narr)
00281
00282      ist = 0
00283      l   = 1
00284      r   = narr
00285
00286      DO
00287          IF (r - l < nn) THEN
00288              DO j = l + 1, r
00289                  tmpidx = idx1d(j)
00290                  a = arr1d(tmpidx)
00291                  DO i = j - 1, l, -1
00292                      IF (arr1d(idx1d(i)) <= a) EXIT
00293                      idx1d(i + 1) = idx1d(i)
00294                  END DO
00295                  idx1d(i + 1) = tmpidx
00296              END DO
00297
00298
00299
00300
00301
00302
00303
00304
00305
00306
00307
00308
00309
00310
00311
00312

```

```

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(idxld(k), idxld(l + 1))
00327     CALL icompxchg(idxld(l), idxld(r))
00328     CALL icompxchg(idxld(l + 1), idxld(r))
00329     CALL icompxchg(idxld(l), idxld(l + 1))
00330
00331     i = l + 1
00332     j = r
00333     tmpidx = idxld(l + 1)
00334     a = arrld(tmpidx)
00335
00336     DO
00337       DO
00338         i = i + 1
00339         IF (arrld(idxld(i)) > a) EXIT
00340     END DO
00341
00342     DO
00343       j = j - 1
00344       IF (arrld(idxld(j)) < a) EXIT
00345     END DO
00346
00347     IF (j < i) EXIT
00348     CALL swap(idxld(i), idxld(j))
00349   END DO
00350
00351   idxld(l + 1) = idxld(j)
00352   idxld(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(trimstr1))
00359
00360     CALL logmessage(error, scratchmessage)
00361     CALL unsetmessagesource()
00362
00363     IF (PRESENT(status)) status = 2
00364
00365     RETURN
00366
00367   END IF
00368
00369   IF (r - i + 1 >= j - 1) THEN
00370     stack(ist) = r
00371     stack(ist - 1) = i
00372     r = j - 1
00373   ELSE
00374     stack(ist) = j - 1
00375     stack(ist - 1) = l
00376     l = i
00377   END IF
00378 END IF
00379 END DO
00380
00381 CALL unsetmessagesource()
00382
00383
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 !-----  

00409 ! S U B R O U T I N E   I N D E X X   S T R I N G
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(l, 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         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(idxld(k), idxld(l + 1))
00505     CALL icompxchg(idxld(l), idxld(r))
00506     CALL icompxchg(idxld(l + 1), idxld(r))
00507     CALL icompxchg(idxld(l), idxld(l + 1))
00508
00509     i = l + 1
00510     j = r
00511     tmpidx = idxld(l + 1)
00512     a = arrld(tmpidx)
00513
00514     DO
00515         DO
00516             i = i + 1
00517             IF (stringlexcomp(arrld(idxld(i)), a, sflag) > 0) EXIT
00518         END DO
00519
00520         DO
00521             j = j - 1
00522             IF (stringlexcomp(arrld(idxld(j)), a, sflag) < 0) EXIT
00523         END DO
00524
00525             IF (j < i) EXIT
00526             CALL swap(idxld(i), idxld(j))
00527         END DO
00528
00529     idxld(l + 1) = idxld(j)
00530     idxld(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     CONTAINS
00562
00563     SUBROUTINE icompxchg(i, j)
00564
00565         IMPLICIT NONE
00566
00567         ! Global variables
00568         INTEGER, INTENT(INOUT) :: i, j
00569
00570         ! Local variables
00571         INTEGER :: swp
00572
00573

```

```

00574     IF (stringlexcomp(arrld(j), arrld(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 !-----+
00606 !-----+
00607 SUBROUTINE indexxsingle(arr1D, idx1D, status)
00608
00609     IMPLICIT NONE
00610
00611     ! Global variables
00612     REAL(SP), DIMENSION(:), INTENT(IN) :: arr1D
00613     INTEGER, DIMENSION(:), INTENT(OUT) :: idx1D
00614     INTEGER, OPTIONAL, INTENT(OUT)      :: status
00615
00616     ! Local variables
00617     INTEGER, PARAMETER                  :: NN = 15, nstack = 50
00618     REAL(SP)                           :: a
00619     INTEGER                            :: nARR, nIDX, tmpIDX
00620     INTEGER                            :: k, i, j, l, r
00621     INTEGER                            :: ist, stack(NSTACK)
00622     CHARACTER(LEN=64)                   :: tmpStr1, tmpStr2
00623
00624
00625     CALL setmessagesource("IndexxSingle")
00626
00627     IF (PRESENT(status)) status = 0
00628
00629     narr = SIZE(arr1D, 1)
00630     nidx = SIZE(idx1D, 1)
00631
00632     IF (narr /= nidx) THEN
00633         WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00634         WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nidx
00635         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00636                                         trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00637
00638     CALL logmessage(error, scratchmessage)
00639     CALL unsetmessagesource()
00640
00641     IF (PRESENT(status)) status = 1
00642
00643     RETURN
00644 END IF
00645
00646     idx1D = arth(1, 1, narr)
00647
00648     ist = 0
00649     l   = 1
00650     r   = narr
00651
00652     DO
00653         IF (r - l < nn) THEN
00654             DO j = l + 1, r
00655                 tmpidx = idx1D(j)
00656                 a = arr1D(tmpidx)
00657                 DO i = j - 1, l, -1
00658                     IF (arr1D(idx1D(i)) <= a) EXIT
00659                     idx1D(i + 1) = idx1D(i)
00660                 END DO
00661                 idx1D(i + 1) = tmpidx
00662             END DO
00663
00664         IF (ist == 0) THEN
00665             CALL unsetmessagesource()
00666
00667             RETURN
00668         END IF
00669
00670         r   = stack(ist)
00671         l   = stack(ist - 1)

```

```

00672     ist = ist - 2
00673   ELSE
00674     k = (l + r) / 2
00675
00676     CALL swap(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) = 1
00726       l = i
00727     END IF
00728   END IF
00729 END DO
00730
00731 CALL unsetmessagesource()
00732
00733 CONTAINS
00734
00735 SUBROUTINE icompxchg(i, j)
00736
00737   IMPLICIT NONE
00738
00739   ! Global variables
00740   INTEGER, INTENT(INOUT) :: i, j
00741
00742   ! Local variables
00743   INTEGER :: swp
00744
00745   IF (arrld(j) < arrld(i)) THEN
00746     swp = i
00747     i = j
00748     j = swp
00749   END IF
00750
00751 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 !-----
00778 !-----
00779 SUBROUTINE indexxdouble(arr1D, idx1D, status)
00780
00781 IMPLICIT NONE
00782
00783 ! Global variables
00784 REAL(HP), DIMENSION(:, INTENT(IN) :: arr1D
00785 INTEGER, DIMENSION(:, INTENT(OUT) :: idx1D
00786 INTEGER, OPTIONAL, INTENT(OUT) :: status
00787
00788 ! Local variables
00789 INTEGER, PARAMETER :: NN = 15, nstack = 50
00790 REAL(HP) :: a
00791 INTEGER :: nARR, nIndex, tmpIDX
00792 INTEGER :: k, i, j, l, r
00793 INTEGER :: ist, stack(NSTACK)
00794 CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00795
00796
00797 CALL setmessagesource("IndexxDouble")
00798
00799 IF (PRESENT(status)) status = 0
00800
00801 narr = SIZE(arr1D, 1)
00802 nIndex = SIZE(idx1D, 1)
00803
00804 IF (narr /= nIndex) THEN
00805   WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
00806   WRITE(tmpstr2, '(a, i0)') 'nIDX = ', nIndex
00807   WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and idx1D is not the same: ' // &
00808   trim(adjustl(tmpstr1)) // ', ' // trim(adjustl(tmpstr2))
00809
00810 CALL logmessage(error, scratchmessage)
00811 CALL unsetmessagesource()
00812
00813 IF (PRESENT(status)) status = 1
00814
00815 RETURN
00816 END IF
00817
00818 idx1D = arth(1, 1, narr)
00819
00820 ist = 0
00821 l = 1
00822 r = narr
00823
00824 DO
00825   IF (r - l < nn) THEN
00826     DO j = l + 1, r
00827       tmpidx = idx1D(j)
00828       a = arr1D(tmpidx)
00829       DO i = j - 1, l, -1
00830         IF (arr1D(idx1D(i)) <= a) EXIT
00831         idx1D(i + 1) = idx1D(i)
00832       END DO
00833       idx1D(i + 1) = tmpidx
00834     END DO
00835
00836   IF (ist == 0) THEN
00837     CALL unsetmessagesource()
00838
00839   RETURN
00840 END IF
00841
00842   r = stack(ist)
00843   l = stack(ist - 1)
00844   ist = ist - 2
00845 ELSE
00846   k = (l + r) / 2
00847
00848   CALL swap(idx1D(k), idx1D(l + 1))
00849   CALL icompxchg(idx1D(l), idx1D(r))
00850   CALL icompxchg(idx1D(l + 1), idx1D(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 (arrld(j) < arrld(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 !----- SUBROUTINE QUICK SORT
00931 !-----
```

```

00932 !-----
00950 !-----
00951 SUBROUTINE quicksort(arr1D, status)
00952
00953 IMPLICIT NONE
00954
00955 ! Global variables
00956 REAL(SZ), DIMENSION(:, INTENT(INOUT) :: arr1D
00957 INTEGER, OPTIONAL, INTENT(OUT) :: status
00958
00959 ! Local variables
00960 INTEGER, PARAMETER :: NN = 15, nstack = 50
00961 REAL(SZ) :: a
00962 INTEGER :: nARR
00963 INTEGER :: k, i, j, l, r
00964 INTEGER :: ist, stack(NSTACK)
00965 CHARACTER(LEN=64) :: tmpStr1
00966
00967
00968 CALL setmessagesource("QuickSort")
00969
00970 IF (PRESENT(status)) status = 0
00971
00972 narr = size(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 ELSE
01004   k = (l + r) / 2
01005
01006   CALL swap(arr1d(k), arr1d(l + 1))
01007   CALL swap(arr1d(l), arr1d(r), arr1d(l) > arr1d(r))
01008   CALL swap(arr1d(l + 1), arr1d(r), arr1d(l + 1) > arr1d(r))
01009   CALL swap(arr1d(l), arr1d(l + 1), arr1d(l) > arr1d(l + 1))
01010
01011 ! Initialize pointers for partitioning
01012 i = l + 1
01013 j = r
01014 a = arr1d(l + 1) ! Partitioning element.
01015
01016 DO ! Here is the meat.
01017   ! Scan up to find element >= a
01018   DO
01019     i = i + 1
01020     IF (arr1d(i) > a) EXIT
01021   END DO
01022
01023   ! Scan down to find element <= a
01024   DO
01025     j = j - 1
01026     IF (arr1d(j) < a) EXIT
01027   END DO
01028
01029 ! Pointers crossed. Exit with partitioning complete.

```

```

01030      IF (j < i) EXIT
01031
01032      CALL swap(arr1d(i), arr1d(j)) !Exchange elements.
01033      END DO
01034
01035      ! Insert partitioning element
01036      arr1d(l + 1) = arr1d(j)
01037      arr1d(j) = a
01038      ist = ist + 2
01039
01040      ! Push pointers to larger subarray on stack; process smaller subarray immediately.
01041      IF (ist > nstack) THEN
01042          WRITE(tmpstr1, '(a, i0)') 'NSTACK = ', nstack
01043          WRITE(scratchmessage, '(a)') 'The value of the NSTACK parameter is too small: ' // &
01044              trim(adjustl(trimstr1))
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
01079 SUBROUTINE sort2(arr1D, slv1D, status)
01080
01081     IMPLICIT NONE
01082
01083     ! Global variables
01084     REAL(SZ), DIMENSION(:, ), INTENT(INOUT) :: arr1D, slv1D
01085     INTEGER, OPTIONAL, INTENT(OUT) :: status
01086
01087     ! Local variables
01088     INTEGER :: nARR, nSLV
01089     INTEGER, DIMENSION(SIZE(arr1D)) :: idx1D
01090     CHARACTER(LEN=64) :: tmpStr1, tmpStr2
01091
01092
01093     CALL setmessagesource("Sort2")
01094
01095     narr = SIZE(arr1D, 1)
01096     nsdv = SIZE(slv1D, 1)
01097
01098     IF (narr /= nsdv) THEN
01099         WRITE(tmpstr1, '(a, i0)') 'nARR = ', narr
01100         WRITE(tmpstr2, '(a, i0)') 'nSLV = ', nsdv
01101         WRITE(scratchmessage, '(a)') 'The size of the 1D arrays arr1D and slv1D is not the same: ' // &
01102             trim(adjustl(trimstr1)) // ', ' // trim(adjustl(trimstr2))
01103
01104     CALL logmessage(error, scratchmessage)
01105     CALL unsetmessagesource()
01106
01107     IF (PRESENT(status)) status = 1
01108
01109     RETURN
01110 END IF
01111
01112 ! Make the index array

```

```
01129     CALL indexx(arrld, idxld, status)
01130
01131     ! Sort the array
01132     arrld = arrld(idxld)
01133
01134     ! Rearrange slave
01135     slvld = slvld(idxld)
01136
01137     CALL unsetmessagessource()
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 SUBROUTINE arraycopyint(src, dest, nCP, nNCP)
01149
01150     IMPLICIT NONE
01151
01152     ! Global variables
01153     INTEGER, DIMENSION(:), INTENT(IN) :: src
01154     INTEGER, DIMENSION(:), INTENT(OUT) :: dest
01155     INTEGER, INTENT(OUT) :: nCP, nNCP
01156
01157     ncp = min(SIZE(src), SIZE(dest))
01158     nncp = SIZE(src) - ncp
01159     dest(1:ncp) = src(1:ncp)
01160
01161 END SUBROUTINE arraycopyint
01162
01163 !=====
01164 ! 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
01165 !-----
01166
01167 SUBROUTINE arraycopsingle(src, dest, nCP, nNCP)
01168
01169     IMPLICIT NONE
01170
01171     ! Global variables
01172     REAL(SP), DIMENSION(:), INTENT(IN) :: src
01173     REAL(SP), DIMENSION(:), INTENT(OUT) :: dest
01174     INTEGER, INTENT(OUT) :: nCP, nNCP
01175
01176     ncp = min(SIZE(src), SIZE(dest))
01177     nncp = SIZE(src) - ncp
01178     dest(1:ncp) = src(1:ncp)
01179
01180 END SUBROUTINE arraycopsingle
01181
01182 !=====
01183 ! 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
01184 !-----
01185
01186 SUBROUTINE arraycopydouble(src, dest, nCP, nNCP)
01187
01188     IMPLICIT NONE
01189
01190     ! Global variables
01191     REAL(HP), DIMENSION(:), INTENT(IN) :: src
01192     REAL(HP), DIMENSION(:), INTENT(OUT) :: dest
01193     INTEGER, INTENT(OUT) :: nCP, nNCP
01194
01195     ncp = min(SIZE(src), SIZE(dest))
01196     nncp = SIZE(src) - ncp
01197     dest(1:ncp) = src(1:ncp)
01198
01199 END SUBROUTINE arraycopydouble
01200
01201 !=====
01202 ! F U N C T I O N   A R R A Y   E Q U A L   I N T
01203 !-----
```

```

01280 !-----+
01281 LOGICAL FUNCTION arrayequalint(arr1, arr2) RESULT(myValOut)
01282
01283 IMPLICIT NONE
01284
01285 ! Global variables
01286 INTEGER, DIMENSION(:, ), INTENT(IN) :: arr1, arr2
01287
01288
01289 IF (SIZE(arr1) /= SIZE(arr2)) THEN
01290   myvalout = .false.
01291
01292   RETURN
01293 END IF
01294
01295 myvalout = .true.
01296 IF (any(arr1 - arr2 /= 0)) myvalout = .false.
01297
01298 RETURN
01299
01300 END FUNCTION arrayequalint
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 !-----+
01325
01326 LOGICAL FUNCTION arrayequalsingle(arr1, arr2) RESULT(myValOut)
01327
01328 IMPLICIT NONE
01329
01330 ! Global variables
01331 REAL(sp), DIMENSION(:, ), INTENT(IN) :: arr1, arr2
01332
01333 ! Local variables
01334 INTEGER :: i
01335
01336
01337 IF (SIZE(arr1) /= SIZE(arr2)) THEN
01338   myvalout = .false.
01339
01340   RETURN
01341 END IF
01342
01343 myvalout = .true.
01344
01345 DO i = 1, SIZE(arr1, 1)
01346   IF (comparereals(arr1(i), arr2(i), 0.0000001_sp) /= 0) THEN
01347     myvalout = .false.
01348
01349   EXIT
01350 END IF
01351 END DO
01352
01353 RETURN
01354
01355 END FUNCTION arrayequalsingle
01356
01357 !=====
01358
01359 !-----+
01360 ! F U N C T I O N   A R R A Y   E Q U A L   S I N G L E
01361 !-----+
01380
01381 LOGICAL FUNCTION arrayequaldouble(arr1, arr2) RESULT(myValOut)
01382
01383 IMPLICIT NONE
01384
01385 ! Global variables
01386 REAL(hp), DIMENSION(:, ), INTENT(IN) :: arr1, arr2
01387
01388 ! Local variables
01389 INTEGER :: i
01390
01391
01392 IF (SIZE(arr1) /= SIZE(arr2)) THEN
01393   myvalout = .false.
01394
01395   RETURN
01396 END IF

```

```

01397
01398     myvalout = .true.
01399
01400     DO i = 1, SIZE(arrl, 1)
01401       IF (comparereals(arrl(i), arr2(i), 0.000000000001_hp) /= 0) THEN
01402         myvalout = .false.
01403
01404       EXIT
01405     END IF
01406   END DO
01407
01408   RETURN
01409
01410 END FUNCTION arrayequaldouble
01411
01412 !=====
01413 !
01414 !-----+
01415 ! F U N C T I O N   S T R I N G   L E X   C O M P
01416 !-----+
01417 !
01418 !
01419 !-----+
01420 INTEGER FUNCTION stringlexcomp(str1, str2, msensitive) RESULT(myValOut)
01421
01422   USE utilities, ONLY : touppercase
01423
01424   IMPLICIT NONE
01425
01426   ! Global variables
01427   CHARACTER(LEN=*) , INTENT(IN) :: str1, str2
01428   LOGICAL, OPTIONAL, INTENT(IN) :: msensitive
01429
01430   ! Local variables
01431   LOGICAL :: sflag
01432
01433   sflag = .true.
01434   IF (PRESENT(msensitive)) sflag = msensitive
01435
01436   IF (sflag) THEN
01437     IF (trim(str1) == trim(str2)) THEN
01438       myvalout = 0
01439     ELSE IF (trim(str1) < trim(str2)) THEN
01440       myvalout = -1
01441     ELSE
01442       myvalout = 1
01443     END IF
01444   ELSE
01445     IF (touppercase(trim(str1)) == touppercase(trim(str2))) THEN
01446       myvalout = 0
01447     ELSE IF (touppercase(trim(str1)) < touppercase(trim(str2))) THEN
01448       myvalout = -1
01449     ELSE
01450       myvalout = 1
01451     END IF
01452   END IF
01453
01454   RETURN
01455
01456 END FUNCTION stringlexcomp
01457
01458 !=====
01459 !
01460 !-----+
01461 ! S U B R O U T I N E   S W A P   I N T
01462 !-----+
01463 !
01464 !
01465 SUBROUTINE swapint(a, b, mask)
01466
01467   IMPLICIT NONE
01468
01469   ! Global variables
01470   INTEGER, INTENT(INOUT) :: a, b
01471   LOGICAL, OPTIONAL, INTENT(IN) :: mask
01472
01473   ! Local variables
01474   INTEGER :: dum
01475   LOGICAL :: mFlag
01476
01477   mflag = .true.
01478   IF (PRESENT(mask)) mflag = mask
01479
01480 !=====
01481 !
01482 !-----+
01483 !
01484 !
01485 
```

```
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 ! SUBROUTINE SWAP SINGLE
01533 !-----
01534 !-----
01535 !-----
01536 SUBROUTINE swapsingle(a, b, mask)
01537
01538     IMPLICIT NONE
01539
01540     ! Global variables
01541     REAL(SP), INTENT(INOUT)      :: a, b
01542     LOGICAL, OPTIONAL, INTENT(IN) :: mask
01543
01544     ! Local variables
01545     REAL(SP) :: dum
01546     LOGICAL  :: mFlag
01547
01548
01549     mflag = .true.
01550     IF (PRESENT(mask)) mflag = mask
01551
01552     IF (mflag) THEN
01553         dum = a
01554         a   = b
01555         b   = dum
01556     END IF
01557
01558 END SUBROUTINE swapsingle
01559
01560 !=====
01561
01562 !-----
01563 ! SUBROUTINE SWAP DOUBLE
01564 !-----
01565 !-----
01566 !-----
01567 SUBROUTINE swapdouble(a, b, mask)
01568
01569     IMPLICIT NONE
01570
01571     ! Global variables
01572     REAL(HP), INTENT(INOUT)      :: a, b
01573     LOGICAL, OPTIONAL, INTENT(IN) :: mask
01574
01575     ! Local variables
01576     REAL(HP) :: dum
01577     LOGICAL  :: mFlag
01578
01579
01580     mflag = .true.
01581     IF (PRESENT(mask)) mflag = mask
01582
01583     IF (mflag) THEN
01584         dum = a
01585         a   = b
01586         b   = dum
01587     END IF
01588
01589 END SUBROUTINE swapdouble
01590
01591 !=====
01592
01593 !-----
01594 ! SUBROUTINE SWAP INT VEC
01595 !-----
01596 !-----
01597 !-----
01598 SUBROUTINE swapintvec(a, b, mask)
01599
01600     IMPLICIT NONE
01601
01602     ! Global variables
01603     INTEGER, DIMENSION(:), INTENT(INOUT) :: a, b
01604     LOGICAL, OPTIONAL, INTENT(IN)       :: mask
```

```

01665      ! Local variables
01666      INTEGER, DIMENSION(SIZE(a)) :: dum
01667      LOGICAL                      :: mFlag
01668
01669
01670      mflag = .true.
01671      IF (PRESENT(mask)) mflag = mask
01672
01673      IF (mflag) THEN
01674          dum = a
01675          a   = b
01676          b   = dum
01677      END IF
01678
01679
01680  END SUBROUTINE swapintvec
01681
01682 !=====
01683
01684 !-----
01685 ! S U B R O U T I N E   S W A P   S I N G L E   V E C
01686 !-----
01708 !-----
01709 SUBROUTINE swapsinglevec(a, b, mask)
01710
01711     IMPLICIT NONE
01712
01713     ! Global variables
01714     REAL(SP), DIMENSION(:, ), INTENT(INOUT) :: a, b
01715     LOGICAL, OPTIONAL, INTENT(IN)           :: mask
01716
01717     ! Local variables
01718     REAL(SP), DIMENSION(SIZE(a)) :: dum
01719     LOGICAL                      :: mFlag
01720
01721
01722     mflag = .true.
01723     IF (PRESENT(mask)) mflag = mask
01724
01725     IF (mflag) THEN
01726         dum = a
01727         a   = b
01728         b   = dum
01729     END IF
01730
01731  END SUBROUTINE swapsinglevec
01732
01733 !=====
01734
01735 !-----
01736 ! S U B R O U T I N E   S W A P   D O U B L E   V E C
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 !-----
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 !-----
01856 !-----
01857
01858 pure FUNCTION arthsingle(first, increment, n) RESULT(arthOut)
01859
01860 IMPLICIT NONE
01861
01862 ! Global variables
01863 REAL(sp), INTENT(IN) :: first, increment
01864 INTEGER, INTENT(IN) :: n
01865 REAL(sp), DIMENSION(n) :: arthout
01866
01867 ! Local variables
01868 INTEGER, PARAMETER :: npARTH = 16, npARTH2 = 8
01869 INTEGER :: k, k2
01870 REAL(sp) :: temp
01871
01872
01873 IF (n > 0) arthout(1) = first
01874
01875 IF (n <= npARTH) THEN
01876   DO k = 2, n
01877     arthout(k) = arthout(k - 1) + increment
01878   END DO
01879 ELSE
01880   DO k = 2, npARTH2
01881     arthout(k) = arthout(k - 1) + increment
01882   END DO
01883
01884   temp = increment * npARTH2
01885   k = npARTH2
01886
01887   DO
01888     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 !----- F U N C T I O N   A R T H   D O U B L E
01921 !-----
01922 !-----
01923 !-----
01924 pure FUNCTION arthdouble(first, increment, n) RESULT(arthOut)
01925
01926      IMPLICIT NONE
01927
01928      ! Global variables
01929      REAL(hp), INTENT(IN)    :: first, increment
01930      INTEGER, INTENT(IN)    :: n
01931      REAL(hp), DIMENSION(n) :: arthout
01932
01933      ! Local variables
01934      INTEGER, PARAMETER :: nparth = 16, nparth2 = 8
01935      INTEGER :: k, k2
01936      REAL(hp) :: temp
01937
01938      IF (n > 0) arthout(1) = first
01939
01940      IF (n <= nparth) THEN
01941          DO k = 2, n
01942              arthout(k) = arthout(k - 1) + increment
01943          END DO
01944      ELSE
01945          DO k = 2, nparth2
01946              arthout(k) = arthout(k - 1) + increment
01947          END DO
01948
01949          temp = increment * nparth2
01950          k = nparth2
01951
01952          DO
01953              IF (k >= n) EXIT
01954              k2 = k + k
01955              arthout(k + 1:min(k2, n)) = temp + arthout(1:min(k, n - k))
01956              temp = temp + temp
01957              k = k2
01958          END DO
01959      END IF
01960
01961      RETURN
01962
01963  END FUNCTION arthdouble
01964
01965 !=====
01966 END MODULE sortutils

```

17.39 /home/takis/CSDL/parwinds-doc/src/timedateutils.F90 File Reference

Data Types

- interface `timedateutils::timeconv`
- interface `timedateutils::gregtojulday`
- interface `timedateutils::splittatetimestring`

Modules

- module `timedateutils`

Functions/Subroutines

- subroutine `timedateutils::timeconviseC` (iYear, iMonth, iDay, iHour, iMin, iSec, timeSec)
Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- subroutine `timedateutils::timeconvrsec` (iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)
Convert time from year, month, day, hour, min, sec into seconds since the reference date of the simulation.
- logical function `timedateutils::leapyear` (iYear)
Checks for a leap year.
- integer function `timedateutils::yeardays` (iYear)
Determines the days of the year.
- integer function `timedateutils::monthdays` (iYear, iMonth)
Determines the days in the month of the year.
- integer function `timedateutils::dayofyear` (iYear, iMonth, iDay)
Determines the day of the year.
- real(sz) function `timedateutils::gregtojuldaysec` (iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
Determines the Julian date from a Gregorian date.
- real(sz) function `timedateutils::gregtojuldaysec` (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::splittdatetimestring` (inDateTime, iYear, iMonth, iDay, iHour, iMin, iSec)
Splits a date string into components.
- subroutine `timedateutils::splittdatetimestring2` (inDateTime, iDate, iTime)
Splits a date string into two components.
- character(len=len(indatetime)) function `timedateutils::preprocessdatetimestring` (inDateTime)
Pre-processes an arbitrary date string.
- integer function `timedateutils::joindate` (iYear, iMonth, iDay)
Pre-processes an arbitrary date string.
- subroutine `timedateutils::splittdate` (inDate, iYear, iMonth, iDay)
Pre-processes an arbitrary date string.
- character(len=64) function `timedateutils::datetime2string` (year, month, day, hour, min, sec, sep, units, zone, err)
Constructs a NetCDF time string.
- real(sz) function `timedateutils::gettimeconvsec` (units, invert)
Calculates the conversion factor between time units and seconds.
- real(sz) function `timedateutils::elapsedsecs` (inTime1, inTime2, inUnits)
Calculates the elapsed time in seconds.
- 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

17.39.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [timedateutils.F90](#).

17.40 **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 CONTAINS
00092
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 timeconvise(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, 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 !=====
00169 !
00170 !-----S U B R O U T I N E   T I M E   C O N V   R S E C
00171 !
00172 !
00173 !
00174 SUBROUTINE timeconvrsec(iYear, iMonth, iDay, iHour, iMin, rSec, timeSec)
00175
00176 USE pahm_global, ONLY : refyear, refmonth, refday, refhour, refmin, refsec
00177
00178 IMPLICIT NONE
00179
00180 ! Global variables
00181 INTEGER, INTENT(IN) :: iYear, iMonth, iDay, iHour, iMin
00182 REAL(SZ), INTENT(IN) :: rSec
00183 REAL(SZ), INTENT(OUT) :: timeSec
00184
00185 ! Local variables
00186 REAL(SZ) :: jd0, jd1
00187 CHARACTER(LEN=64) :: tmpStr1, tmpStr2
00188
00189 !---- START CALCULATIONS ----
00190
00191 CALL setmessagesource("TimeConv")
00192
00193 jd0 = gregtojulday(refyear, refmonth, refday, refhour, refmin, refsec)
00194 jd1 = gregtojulday(iyear, imonth, iday, ihour, imin, rsec)
00195
00196 IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
00197   timesec = rmissv
00198
00199   WRITE(tmpstr1, '(f20.3)') jd0
00200   WRITE(tmpstr2, '(f20.3)') jd1
00201   WRITE(scratchmessage, '(a)') 'Invalid julian dates calculated: refJD = ' // &
00202           trim(adjustl(tmpstr1)) // ', inpJD = ' // trim(adjustl(tmpstr2))
00203
00204   CALL allmessage(error, scratchmessage)
00205   CALL unsetmessagesource()
00206
00207   CALL terminate()
00208 END IF
00209
00210   timesec = elapsedsecs(jd0, jd1, 'days')
00211
00212   CALL unsetmessagesource()
00213
00214   RETURN
00215
00216 END SUBROUTINE timeconvrsec
00217
00218 !=====
00219 !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
00220 !DEL !-----
00221 !DEL !-----
00222 !DEL !-----
00223 !DEL !-----
00224 !DEL !-----SUBROUTINE TimeConvADCIRC(year, month, day, hour, minute, sec, timeSec)
00225
00226
00227
00228
00229
00230
00231
00232
00233
00234
00235
00236
00237
00238
00239
00240
00241
00242
00243
00244
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 ! FUNCTION LEAP YEAR
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 !-----
00365 !
00366 INTEGER FUNCTION yeardays(iYear) RESULT(myVal)
00367 IMPLICIT NONE
00369
00370 INTEGER, INTENT(IN) :: iyear
00371
00372 !---- START CALCULATIONS ----
00373
00374 myval = 365
00375 IF (leapyear(iyear)) myval = 366
00376
00377 RETURN
00378 END FUNCTION yeardays
00379
00380 !=====
00381 !
00382 !-----
00383 ! F U N C T I O N   M O N T H   D A Y S
00384 !-----
00402 !
00403 INTEGER FUNCTION monthdays(iYear, iMonth) RESULT(myVal)
00404 IMPLICIT NONE
00406
00407 ! Global variables
00408 INTEGER, INTENT(IN) :: iyear, imonth
00409
00410 ! Local variables
00411 INTEGER :: leap, monlen(12, 2)
00412
00413 !---- START CALCULATIONS ----
00414
00415 IF ((iyear < 1582) .OR. (imonth < 1) .OR. (imonth > 12)) THEN
00416 myval = imissv
00417
00418 RETURN
00419 END IF
00420
00421 ! Initialize lenghts of months:
00422 monlen = reshape(/ 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31, &
00423 31, 29, 31, 30, 31, 31, 31, 30, 31, 30, 31 /), &
00424 (/ 12, 2 /))
00425
00426 leap = 1
00427 IF (leapyear(iyear)) leap = 2
00428
00429 myval = monlen(imonth, leap)
00430
00431 RETURN
00432 END FUNCTION monthdays
00433
00434 !=====
00435 !
00436 !-----
00437 ! F U N C T I O N   D A Y   O F   Y E A R
00438 !-----
00459 !
00460 INTEGER FUNCTION dayofyear(iYear, iMonth, iDay) RESULT(myVal)
00461 IMPLICIT NONE
00463
00464 ! Global variables
00465 INTEGER, INTENT(IN) :: iyear, imonth, iday
00466
00467 ! Local variables
00468 REAL(sz) :: jd0, jd1
00469
00470 !---- START CALCULATIONS ----
00471
00472 jd0 = gregtojulday(iyear, 1, 1, 0, 0, 0)
00473 jd1 = gregtojulday(iyear, imonth, iday, 0, 0, 0)
00474
00475 IF ((comparereals(jd0, rmissv) <= 0) .OR. (comparereals(jd1, rmissv) <= 0)) THEN
00476 myval = imissv
00477
00478 RETURN
00479 END IF
00480

```

```

00481     myval = int(jd1 - jd0 + 1.0_sz)
00482
00483     RETURN
00484 END FUNCTION dayofyear
00485
00486 !=====
00487 !-----
00488 !----- F U N C T I O N   G R E G   T O   J U L   D A Y   I S E C
00489 !-----
00490 !-----
00491 !-----
00492 !----- REAL(sz) function gregtojuldaysec(iyear, imonth, iday, ihour, imin, isec, mjd) result(myval)
00493
00494     IMPLICIT NONE
00495
00496     ! Global variables
00497     INTEGER, INTENT(IN)          :: iyear, imonth, iday, ihour, imin, isec
00498     INTEGER, OPTIONAL, INTENT(IN) :: mjd
00499
00500     ! Local variables
00501     INTEGER :: leap, monlen(12, 2)
00502     LOGICAL :: modjul
00503     REAL(hp) :: templ, temp2
00504
00505     !---- START CALCULATIONS ----
00506
00507     modjul = .false.
00508     IF (PRESENT(mjd)) THEN
00509         modjul = (mjd > 0)
00510     ELSE
00511         modjul = (usemodjulday > 0)
00512     END IF
00513
00514     ! Initialize lengths of months:
00515     monlen = reshape(/( 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, &
00516                      31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31, 31 /), &
00517                      (/ 12, 2 /))
00518
00519     ! This function intentionally works on Gregorian dates only. For modeling
00520     ! purposes the min date supported 1582/10/05 is sufficient. Most likely,
00521     ! it is not necessary to go beyond that date.
00522
00523     ! Is this a LEAP year?
00524     leap = 1
00525     IF (leapyear(iyear)) leap = 2
00526
00527     IF (joindate(iyear, imonth, iday) < firstgregdate) THEN
00528         myval = rmissv
00529
00530         RETURN
00531     ELSE IF (((imonth < 1) .OR. (imonth > 12) .OR. &
00532               (iday < 1) .OR. (iday > monlen(imonth, leap)) .OR. &
00533               (ihour < 0) .OR. (ihour > 23) .OR. &
00534               (imin < 0) .OR. (imin > 59) .OR. &
00535               (isec < 0) .OR. (isec > 60)) THEN
00536         myval = rmissv
00537
00538         RETURN
00539     ELSE
00540         templ = int((imonth - 14.0_hp) / 12.0_hp)
00541         temp2 = iday - 32075.0_hp
00542             + int(1461.0_hp * (iyear + 4800.0_hp + templ) / 4.0_hp) &
00543             + int(367.0_hp * (imonth - 2.0_hp - templ * 12.0_hp) / 12.0_hp) &
00544             - int(3.0_hp * int((iyear + 4900.0_hp + templ) / 100.0_hp) / 4.0_hp)
00545         templ = real(ihour, hp) * 3600.0_hp &
00546             + real(imin, hp) * 60.0_hp &
00547             + real(isec, hp) - 43200.0_hp
00548
00549         IF (modjul) THEN
00550             myval = temp2 + (templ / 86400.0_hp) - mdjoffset
00551         ELSE
00552             myval = temp2 + (templ / 86400.0_hp)
00553         END IF
00554     END IF
00555
00556         RETURN
00557 END FUNCTION gregtojuldaysec
00558 !=====
00559 !-----
00560 !-----
00561 !-----

```

```

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 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) :: templ, 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, 31, 31, &
00679           31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 31, 31, 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 = rmssv
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 = rmssv
00700
00701   RETURN
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 gregtojuldayrsec
00721
00722 !=====
00723
00724 !----- 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
00725 !-----
00726 !
00774 !
00775 REAL(sz) function gregtojulday2(idate, itime, mjd) result(myval)
00776 IMPLICIT NONE
00777
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, 31, 30 /), &
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 = rmssv
00820
00821   RETURN
00822 ELSE
00823   IF (idate < firstgregdate) THEN
00824     myval = rmssv
00825
00826     RETURN
00827   ELSE
00828     templ = int((imonth - 14.0_hp) / 12.0_hp)
00829     temp2 = iday - 32075.0_hp &
00830           + int(1461.0_hp * (iyear + 4800.0_hp + templ) / 4.0_hp) &
00831           + int(367.0_hp * (imonth - 2.0_hp - templ * 12.0_hp) / 12.0_hp) &
00832           - int(3.0_hp * int((iyear + 4900.0_hp + templ) / 100.0_hp) / 4.0_hp)
00833     templ = 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 + (templ / 86400.0_hp) - mdjoffset
00839     ELSE
00840       myval = temp2 + (templ / 86400.0_hp)
00841     END IF
00842   END IF
00843 END IF
00844
00845 RETURN
00846 END FUNCTION gregtojulday2
00847 !=====
00848 !-----
00849 ! S U B R O U T I N E   J U L   D A Y   T O   G R E G
00850 !-----
00851 !-----
00852 !-----
00853 !-----
00854 SUBROUTINE juldaytogreg(julDay, iYear, iMonth, iDay, iHour, iMin, iSec, mJD)
00855
00856 IMPLICIT NONE
00857
00858 ! Global Variables
00859 REAL(SZ), INTENT(IN) :: julDay

```

```

00904     INTEGER, OPTIONAL, INTENT(IN) :: mJD
00905     INTEGER, INTENT(OUT)          :: iYear, iMonth, iDay, iHour, iMin, iSec
00906
00907     ! Local Variables
00908     REAL(HP) :: temp1 , temp2 , temp3 , temp4 , temp5
00909     REAL(HP) :: thisJulDay, myJulDay, delta
00910     INTEGER   :: nTry
00911     LOGICAL   :: modJul
00912
00913     !----- START CALCULATIONS -----
00914
00915     modjul = .false.
00916     IF (PRESENT(mjd)) THEN
00917       modjul = (mjd > 0)
00918     ELSE
00919       modjul = (usemodjulday > 0)
00920     END IF
00921
00922     IF (modjul) THEN
00923       thisJulday = julday + mdjoffset
00924     ELSE
00925       thisJulday = julday
00926     END IF
00927
00928     ! Check for valid Julian day (Gregorian calendar only)
00929     IF (thisJulday < offfirstgregday) THEN
00930       iyear = imissv
00931       imonth = imissv
00932       iday = imissv
00933       ihour = imissv
00934       imin = imissv
00935       isec = imissv
00936
00937     RETURN
00938   END IF
00939
00940   delta = 0.0_hp
00941   ntry = 1
00942   DO WHILE (ntry <= 2)
00943     myJulday= thisJulday + delta
00944     temp4 = myJulday
00945     temp5 = dmod(myJulday, 1.0_hp)
00946
00947     IF (temp5 < 0.5) THEN
00948       temp3 = 0.5_hp + temp5
00949       temp4 = aint(temp4)
00950     ELSE
00951       temp3 = temp5 - 0.5_hp
00952       temp4 = aint(temp4) + 1.0_hp
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 juldaytogg
00983 !=====

```

```

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 juldaytoreg(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 !----- 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
01046 !-----
01047 !-----
01071 !-----
01072 SUBROUTINE splittimetimestring(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 SUBROUTINE splitdatetimestring2(inDateTime, iDate, iTime)
01127
01128 IMPLICIT NONE
01129
01130 ! Global Variables
01131 CHARACTER(LEN=*) , INTENT(IN)    :: inDateTime
01132 INTEGER, INTENT(OUT)           :: iDate, iTime
01133
01134 ! Local Variables
01135 INTEGER                  :: iYear, iMonth, iDay, iHour, iMin, iSec
01136
01137 !----- START CALCULATIONS -----
01138
01139 CALL splitdatetimestring(indatetime, iyear, imonth, iday, ihour, imin, isec)
01140
01141 IF ((iyear == imissv) .OR. (imonth <= 0) .OR. (iday <= 0)) THEN
01142     idate = imissv
01143 ELSE
01144     idate = joindate(iyear, imonth, iday)
01145 END IF
01146
01147 itime = joindate(ihour, imin, isec)
01148
01149 END SUBROUTINE splitdatetimestring2
01150
01151 !=====
01152
01153 !-----+
01154 ! 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
01155 !-----+
01156 !-----+
01157
01158 FUNCTION preprocessdatetimestring(inDateTime) Result(myValOut)
01159
01160 IMPLICIT NONE
01161
01162 ! Global Variables
01163 CHARACTER(LEN=*) , INTENT(IN)    :: indatetime
01164 CHARACTER(LEN=LEN(inDateTime)) :: myvalout
01165
01166 ! Local Variables
01167 CHARACTER(LEN=1)          :: c
01168 INTEGER                   :: i, ipos
01169
01170 !----- START CALCULATIONS -----
01171
01172 myvalout = blank
01173 ipos = 1
01174
01175 DO i = 1, len(indatetime)
01176     c = indatetime(i:i)
01177     IF ((48 <= ichar(c)) .AND. (ichar(c) <= 57)) THEN
01178         myvalout(ipos:ipos) = c
01179         ipos = ipos + 1
01180     ENDIF
01181 END DO
01182
01183 RETURN
01184
01185 END FUNCTION preprocessdatetimestring
01186
01187 !=====
01188
01189 !-----+
01190 ! F U N C T I O N   J O I N   D A T E
01191 !-----+

```

```

01239 !-----+
01240 INTEGER FUNCTION joindate(iYear, iMonth, iDay) RESULT(myVal)
01241
01242     IMPLICIT NONE
01243
01244     ! Global Variables
01245     INTEGER, INTENT(IN) :: iyear, imonth, iday
01246
01247     !---- START CALCULATIONS ----
01248
01249     myval = iyear * 10000 + imonth * 100 + iday
01250
01251 END FUNCTION joindate
01252
01253 !=====
01254
01255 !-----+
01256 ! S U B R O U T I N E   S P L I T   D A T E
01257 !-----+
01279 !
01280 SUBROUTINE splitdate(indate, iYear, iMonth, iDay)
01281
01282     IMPLICIT NONE
01283
01284     ! Global Variables
01285     INTEGER, INTENT(IN) :: inDate
01286     INTEGER, INTENT(OUT) :: iYear, iMonth, iDay
01287
01288     !---- START CALCULATIONS ----
01289
01290     iyear = indate / 10000
01291     imonth = indate / 100 - iyear * 100
01292     iday = indate - imonth * 100 - iyear * 10000
01293
01294 END SUBROUTINE splitdate
01295
01296 !=====
01297
01298 !-----+
01299 ! F U N C T I O N   D A T E   T I M E 2   S T R I N G
01300 !
01322 !           (optional - for sep <= 0 use ' ', for sep > 0 use 'T')
01325 !           (optional - units = [S(seconds), M(minutes), H(hours), D(days), W(weeks)])
01334 !
01335 FUNCTION datetime2string(year, month, day, hour, min, sec, sep, units, zone, err) result(myValOut)
01336
01337     IMPLICIT NONE
01338
01339     INTEGER,          INTENT(IN)      :: year, month, day
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
01345     ! The resulting date time string. Considering using trim() on it.
01346     CHARACTER(LEN=64) :: myvalout
01347     CHARACTER(LEN=20) :: myunits, myzone
01348     CHARACTER(LEN=1) :: mytimesep
01349     INTEGER          :: myhour, mymin, mysec, myerr
01350
01351     myhour = 0
01352     IF (PRESENT(hour)) myhour = hour
01353     mymin = 0
01354     IF (PRESENT(min))  mymin = min
01355     mysec = 0
01356     IF (PRESENT(sec))  mysec = sec
01357
01358     mytimesep = ''
01359     IF (PRESENT(sep)) THEN
01360         IF (sep > 0) mytimesep = 'T'
01361         IF (sep <= 0) mytimesep = ' '
01362     END IF
01363
01364     IF (PRESENT(units)) THEN
01365         SELECT CASE(trim(adjustl(upp(units))))
01366             CASE('SECONDS', 'SECOND', 'SE', 'SC', 'S')
01367                 myunits = 'seconds since'
01368             CASE('MINUTES', 'MINUTE', 'MIN', 'M')
01369                 myunits = 'minutes since'
01370             CASE('HOURS', 'HOUR', 'HOU', 'HO', 'H')
01371                 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 !----- F U N C T I O N   G E T   T I M E   C O N V   S E C
01410 !-----
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 = adjustl(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 !-----+ F U N C T I O N   E L A P S E D   S E C S
01491 !-----+
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
01534     ELSE
01535         ALLOCATE(CHARACTER(LEN=1) :: unitsval)
01536         unitsval = 'S'
01537     END IF
01538
01539     uconfac = gettimeconvsec(unitsval)
01540
01541     myval = (intime2 - intime1) * uconfac
01542     myval = fixnearwholereal(myval, 0.001_sz)
01543
01544     DEALLOCATE(unitsval)
01545
01546     RETURN
01547
01548 END FUNCTION elapsedsecs
01549 !=====
01550 !-----+
01551 !-----+ F U N C T I O N   U P P
01552 !-----+
01553 !-----+
01565 !
01566 FUNCTION upp(inpString) RESULT(outString)
01567
01568     CHARACTER(*), INTENT(IN) :: inpstring
01569
01570     INTEGER, PARAMETER :: duc = ichar('A') - ichar('a')
01571     CHARACTER(LEN(inpString)) :: outstring
01572     CHARACTER :: ch
01573     INTEGER :: i
01574
01575     DO i = 1, len(inpstring)
01576         ch = inpstring(i:i)
01577         IF ((ch >= 'a') .AND. (ch <= 'z')) ch = char(ichar(ch) + duc)
01578         outstring(i:i) = ch
01579     END DO
01580
01581     RETURN
01582

```

```
01583     END FUNCTION upp
01584
01585 !=====
01586
01587 END MODULE timedateutils
```

17.41 /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::cptogeo_scalar](#) (x, y, lat0, lon0, lat, lon)
Transform from CPP (x, y) coordinates into geographical (lon, lat) coordinates.
- subroutine [utilities::cptogeo_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::sphericaldistanceharr](#) (lat1, lon1, lat2, lon2)
Calculates the distance of two points along the great circle using the Haversine formula.
- subroutine [utilities::sphericalfracpoint](#) (lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)
Calculates the coordinates of an intermediate point between two points along the great circle.
- subroutine [utilities::getlocandratio](#) (val, arrVal, idx1, idx2, wtRatio)
Calculates the location of a value in an 1D array of values.
- integer function [utilities::charunique](#) (inpVec, outVec, idxVec)
Find the unique non-blank elements in 1D character array.
- real(sp) function [utilities::valstr](#) (String)
Returns the value of the leading double precision real numeric string.
- real(hp) function [utilities::dvalstr](#) (String)
Returns the value of the leading double precision real numeric string.
- integer function [utilities::intvalstr](#) (String)
Returns the value of the leading integer numeric string.
- integer function [utilities::realscan](#) (String, Pos, Value)
Scans string looking for the leading single precision real numeric string.
- integer function [utilities::drealscan](#) (String, Pos, Value)
Scans string looking for the leading double precision real numeric string.
- integer function [utilities::intscan](#) (String, Pos, Signed, Value)
Scans string looking for the leading integer numeric string.

Variables

- integer, private [utilities::numbtfiles](#) = 0
- real(sz), parameter [utilities::closetol](#) = 0.001_SZ

17.41.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Definition in file [utilities.F90](#).

17.42 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 cpptogeo  

00035     MODULE PROCEDURE cpptogeo_scalar  

00036     MODULE PROCEDURE cpptogeo_1d  

00037 END INTERFACE cpptogeo  

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      RETURN ! file not found
00102
00103      ELSE
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 SUBROUTINE readcontrolfile(inpFile)
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196 !----- Initialize variables

```

```

00197 ! Global variables
00198 numbtfiles = 0
00199
00200 ! Local variables
00201 inpline = 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, lx, io, 2x, '
00221 fmtdimparinvalid = trim(fmtdimparinvalid) // trim(cntlfmtstr) // ', lx, a)'
00222
00223 fmtparnotfound = '(" Could not find input parameter: ", a, lx, '
00224 fmtparnotfound = trim(fmtparnotfound) // trim(cntlfmtstr) // ', lx, 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   stop ! file not found
00234 ELSE
00235   WRITE(lun_screen, '("The contol file : ", a, " was found and will be opened for reading.")')
00236   trim(tmpfilename)
00237 END IF
00238
00239 ! Open existing file
00240 OPEN(unit=iunit, file=trim(tmpfilename), status='OLD', action='READ', iostat=errio)
00241 IF (errio /= 0) THEN
00242   WRITE(lun_screen, '("Could not open the contol file: ", a, ".")') trim(tmpfilename)
00243   stop ! file found but could not be opened
00244 END IF
00245
00246 DO WHILE (.true.)
00247   READ(unit=iunit, fmt='(a)', err=10, END=20) inpline
00248   status = parseline(inpline, outline, keyword, nval, charval, realval)
00249
00250 IF (status > 0) THEN
00251   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, lx, 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                      above
00303                      numbtfiles = numbtfiles + 1
00304                      besttrackfilename(numbtfiles) = trim(tmpstr)
00305                      besttrackfilenamespecified = .true.
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 splitdatetimestring(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. begsimspecified .OR.
00388              enddatespecified .OR. endsimspecified) 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(adjustl(charval(nval)))
00397                  unittime = tmpcharval(1:1)
00398              ELSE
00399                  IF (trim(adjustl(unittime)) == "") THEN
00400                      tmpcharval = touppercase(adjustl(charval(nval)))
00401                      unittime = tmpcharval(1:1)
00402                  END IF
00403              END IF
00404          END IF
00405
00406      !----- CASE
00407      CASE ('BEGDATETIME')
00408          IF (begdatespecified .OR. begsimspecified) THEN
00409              scratchmessage = 'Only one of "begDateTime" or "begSimTime" can be specified'
00410          CALL allmessage(error, scratchmessage)
00411
00412          begdatespecified = .false.
00413
00414          ELSE
00415              IF (.NOT. refdatespecified) THEN
00416                  scratchmessage = 'Add the "refDateTime" keyword before "begDateTime".'
00417                  CALL allmessage(error, scratchmessage)
00418
00419              ELSE
00420                  IF (nval == 1) THEN
00421                      begdatetime = trim(adjustl(charval(nval)))
00422                  ELSE
00423                      IF (trim(adjustl(begdatetime)) == "") THEN
00424                          begdatetime = trim(adjustl(charval(nval)))
00425                      END IF
00426
00427                  CALL splitdatetimestring(begdatetime, begyear, begmonth, begday, beghour, begmin, begsec)
00428
00429                  IF ((begyear == imissv) .OR. (begmonth <= 0) .OR. (begday <= 0)) THEN
00430                      begdate = imissv
00431                  ELSE
00432                      begdate = joindate(begyear, begmonth, begday)
00433                  END IF
00434                  begtime = joindate(beghour, begmin, begsec)
00435
00436                  CALL timeconv(begyear, begmonth, begday, beghour, begmin, begsec, mdbegsimtime)
00437                  begsimtime = mdbegsimtime * gettimeconvsec(unittime, 1)

```

```

00436      begdatetime = datetime2string(begyear, begmonth, begday, beghour, begmin, begsec)
00437
00438      begdatespecified = .true.
00439      begsimspecified = .true.
00440      END IF
00441      END IF
00442
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 !----- CASE

```

```

00516      CASE ('ENDSIMTIME')
00517          IF (.enddatespecified .OR. endsimspecified) THEN
00518              scratchmessage = 'Only one of "endDateTime" and "endSimTime" can be specified'
00519              CALL allmessage(error, scratchmessage)
00520
00521          endsimspecified = .false.
00522
00523          ELSE
00524              IF (.NOT. refdatespecified) THEN
00525                  scratchmessage = 'Add the "refDateTime" keyword before "endSimTime").'
00526                  CALL allmessage(error, scratchmessage)
00527
00528              ELSE
00529                  npnts = loadrealvar(nval, realval, 1, rvalout)
00530                  endsimtime = rvalout(1)
00531
00532                  mdendsimtime = endsimtime * gettimeconvsec(unittime)
00533
00534                  jday = (mdendsimtime * gettimeconvsec('D', 1)) + gregtojulday(refyear, refmonth, refday,
00535                  refhour, refmin, refsec)
00536                  CALL juldaytoreg(jday, endyear, endmonth, endday, endhour, endmin, endsec)
00537                  begdatetime = datetime2string(endyear, endmonth, endday, endhour, endmin, endsec)
00538
00539                  endsdatespecified = .true.
00540                  endsimspecified = .true.
00541                  END IF
00542          END IF
00543
00544          !----- CASE
00545          CASE ('OUTFILENAME')
00546              IF (nval == 1) THEN
00547                  outfilename = trim(adjustl(charval(nval)))
00548              ELSE
00549                  IF (trim(adjustl(outfilename)) == "") THEN
00550                      outfilename = trim(adjustl(charval(nval)))
00551                  END IF
00552                  IF (trim(adjustl(outfilename)) /= "") outfilenamespecified = .true.
00553
00554          !----- CASE
00555          CASE ('NCVARNAM_PRES')
00556              IF (nval == 1) THEN
00557                  ncvarnam_pres = trim(adjustl(charval(nval)))
00558              ELSE
00559                  IF (trim(adjustl(ncvarnam_pres)) == "") THEN
00560                      ncvarnam_pres = trim(adjustl(charval(nval)))
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
00572          !----- CASE
00573          CASE ('NCVARNAM_WNDY')
00574              IF (nval == 1) THEN
00575                  ncvarnam_wndy = trim(adjustl(charval(nval)))
00576              ELSE
00577                  IF (trim(adjustl(ncvarnam_wndy)) == "") THEN
00578                      ncvarnam_wndy = trim(adjustl(charval(nval)))
00579                  END IF
00580
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 ('NCDLEVEL')
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(inline)), 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 !----- 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
00641 !-----
00642 !-----
00643 !-----
00644 !----- SUBROUTINE printmodelparams()
00645
00646 USE pahm_global
00647
00648 IMPLICIT NONE
00649
00650 CHARACTER(LEN=128) :: tmpStr
00651 INTEGER :: i
00652
00653 IF (writeparams) THEN
00654     print *, "
00655     WRITE(*, '(a)' )      '----- MODEL PARAMETERS -----'
00656
00657     WRITE(*, '(a, a)' )    ' title           = ', trim(adjustl(title))
00658
00659     DO i = 1, nbtrfiles
00660         WRITE(*, '(a, "", i1, "", a)' )   ' bestTrackFileName', i, " = " //
00661         trim(adjustl(besttrackfilename(i)))
00662     END DO
00663     WRITE(*, '(a, a)' )      ' meshFileType      = ', trim(adjustl(meshfiletype))
00664     WRITE(*, '(a, a)' )      ' meshFileName      = ', trim(adjustl(meshfilename))
00665     WRITE(*, '(a, a)' )      ' meshFileForm      = ', trim(adjustl(meshfileform))
00666
00667     print *, "
00668     WRITE(tmpstr, '(f20.5)' ) gravity
00669     WRITE(*, '(a, a)' )      ' gravity           = ', trim(adjustl(tmpstr)) // " m/s^2"
00670     WRITE(tmpstr, '(f20.5)' ) rhoWater
00671     WRITE(*, '(a, a)' )      ' rhoWater          = ', trim(adjustl(tmpstr)) // " kg/m^3"
00672     WRITE(tmpstr, '(f20.5)' ) rhoAir
00673     WRITE(*, '(a, a)' )      ' rhoAir            = ', trim(adjustl(tmpstr)) // " kg/m^3"
00674     WRITE(tmpstr, '(f20.5)' ) backgroundAtmPress
00675     WRITE(*, '(a, a)' )      ' backgroundAtmPress = ', trim(adjustl(tmpstr)) // " mbar"
00676     WRITE(tmpstr, '(f20.2)' ) windreduction

```

```

00683     WRITE(*, '(a, a)')      ' windReduction      = ', trim(adjustl(tmpstr))
00684
00685     print *, "
00686     WRITE(*, '(a, a)')      ' refDateTime      = ', trim(adjustl(refdatetime))
00687     WRITE(*, '(a, i4.4)')    ' refYear          = ', refyear
00688     WRITE(*, '(a, i2.2)')    ' refMonth         = ', refmonth
00689     WRITE(*, '(a, i2.2)')    ' refDay           = ', refday
00690     WRITE(*, '(a, i2.2)')    ' refHour          = ', refhour
00691     WRITE(*, '(a, i2.2)')    ' refMin           = ', refmin
00692     WRITE(*, '(a, i2.2)')    ' refSec           = ', refsec
00693     WRITE(*, '(a, 11)')     ' refDateSpecified = ', refdatespecified
00694
00695     print *, "
00696     WRITE(*, '(a, a)')      ' begDateTime      = ', trim(adjustl(begdatetime))
00697     WRITE(*, '(a, i4.4)')    ' begYear          = ', begyear
00698     WRITE(*, '(a, i2.2)')    ' begMonth         = ', begmonth
00699     WRITE(*, '(a, i2.2)')    ' begDay           = ', begday
00700     WRITE(*, '(a, i2.2)')    ' begHour          = ', beghour
00701     WRITE(*, '(a, i2.2)')    ' begMin           = ', begin
00702     WRITE(*, '(a, i2.2)')    ' begSec           = ', begsec
00703     WRITE(*, '(a, 11)')     ' begDateSpecified = ', begdatespecified
00704
00705     print *, "
00706     WRITE(*, '(a, a)')      ' endDateTime      = ', trim(adjustl(enddatetime))
00707     WRITE(*, '(a, i4.4)')    ' endYear          = ', endyear
00708     WRITE(*, '(a, i2.2)')    ' endMonth         = ', endmonth
00709     WRITE(*, '(a, i2.2)')    ' endDay           = ', endday
00710     WRITE(*, '(a, i2.2)')    ' endHour          = ', endhour
00711     WRITE(*, '(a, i2.2)')    ' endMin           = ', endmin
00712     WRITE(*, '(a, i2.2)')    ' endSec           = ', endsec
00713     WRITE(*, '(a, 11)')     ' endDateSpecified = ', enddatespecified
00714
00715     print *, "
00716     WRITE(*, '(a, a1)')     ' unitTime          = ', unittime
00717     WRITE(tmpstr, '(f20.5)') outdt
00718     WRITE(*, '(a, a)')      ' outDT            = ', trim(adjustl(tmpstr)) // " " //
00719     tolowercase(trim(unittime))
00720     WRITE(tmpstr, '(f20.5)') mdoutdt
00721     WRITE(*, '(a, a)')      ' mdOutDT          = ', trim(adjustl(tmpstr)) // " s"
00722     WRITE(tmpstr, '(f20.5)') begsimtime
00723     WRITE(*, '(a, a)')      ' begSimTime       = ', trim(adjustl(tmpstr)) // " " //
00724     tolowercase(trim(unittime))
00725     WRITE(tmpstr, '(f20.5)') mdbegsimtime
00726     WRITE(*, '(a, a)')      ' mdBegSimTime     = ', trim(adjustl(tmpstr)) // " s"
00727     WRITE(tmpstr, '(f20.5)') endsimtime
00728     WRITE(*, '(a, a)')      ' endSimTime       = ', trim(adjustl(tmpstr)) // " " //
00729     tolowercase(trim(unittime))
00730     WRITE(tmpstr, '(f20.5)') mdendsimtime
00731     WRITE(*, '(a, 11)')     ' mdEndSimTime     = ', trim(adjustl(tmpstr)) // " s"
00732     WRITE(tmpstr, '(i10)')   noutdt
00733     WRITE(*, '(a, a)')      ' nOutDT           = ', trim(adjustl(tmpstr))
00734
00735     print *, "
00736     WRITE(*, '(a, a)')      ' outFileName        = ', trim(adjustl(outfilename))
00737     WRITE(*, '(a, i1)')      ' ncShuffle          = ', ncshuffle
00738     WRITE(*, '(a, i1)')      ' ncDeflate          = ', ncdeflate
00739     WRITE(*, '(a, a)')      ' ncDLevel           = ', nclevel
00740     WRITE(*, '(a, a)')      ' ncVarNam_Pres      = ', trim(ncvarnam_pres)
00741     WRITE(*, '(a, a)')      ' ncVarNam_WndX      = ', trim(ncvarnam_wndx)
00742     WRITE(*, '(a, a)')      ' ncVarNam_WndY      = ', trim(ncvarnam_wndy)
00743
00744     print *, "
00745     WRITE(*, '(a, i1)')     ' modelType          = ', modeltype
00746
00747     WRITE(*, '(a)')          ' ----- MODEL PARAMETERS -----'
00748     print *, "
00749     END IF
00750
00751     END SUBROUTINE printmodelparams
00752 !=====
00753
00754 !-----
00755 ! FUNCTION GET LINE RECORD
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 !-----+
00846 ! F U N C T I O N   P A R S E   L I N E
00847 !-----+
00848
00849 !-----+
00850 INTEGER FUNCTION parseline(inpLine, outLine, keyWord, nVal, cVal, rVal) RESULT(myStatus)
00851
00852      IMPLICIT NONE
00853
00854      ! Imported variable declarations.
00855      CHARACTER(LEN=*, INTENT(IN))          :: inpline
00856      CHARACTER(LEN=LEN(inpLine)), INTENT(OUT) :: outline
00857      CHARACTER(LEN=40), INTENT(INOUT)        :: keyword
00858      INTEGER, INTENT(INOUT)                 :: nval
00859      CHARACTER(LEN=512), DIMENSION(200), INTENT(INOUT) :: cval
00860      REAL(sz), DIMENSION(200), INTENT(INOUT)    :: rval
00861
00862      ! Local variable declarations
00863      LOGICAL                           :: issstring, kextract, decflag, nested
00864      INTEGER                            :: iblank, icont, ipipe, kstr, kend

```

```

00891      INTEGER :: lend, lens, lstr, lval, nmul, schar
00892      INTEGER :: copies, i, ic, ie, ierr, is, j, status
00893      INTEGER, DIMENSION(20) :: imul
00894      CHARACTER(LEN=256) :: vstring, string
00895      CHARACTER(LEN=LEN(inpLine)) :: line
00896      INTEGER :: lenline
00897
00898 ! Table of some ASCII character symbols
00899 !   CHAR     ASCII
00900 !   TAB      9
00901 !   SPC      32
00902 !   !       33
00903 !   #       35
00904 !   *       42
00905 !   +       43
00906 !   -       45
00907 !   /       47
00908 !   :       58
00909 !   =       61
00910 !   \       92
00911 !   |       124
00912
00913 ! Initialize.
00914 line      = blank
00915 vstring   = blank
00916 string    = blank
00917
00918 lenline = getlinerecord(inpLine, line, 1)
00919 outline = line
00920
00921 ! If not a blank or comment line [CHAR(33)!=!], decode and extract input
00922 ! values. Find equal sign [CHAR(61)].
00923 status = -1
00924 nested = .false.
00925 IF ((lenline > 0) .AND. (line(1:1) /= char(33)) .AND. (line(1:1) /= char(35))) THEN
00926   status = 1
00927   kstr = 1
00928   kend = index(line, char(61), back = .false.) - 1
00929   lstr = index(line, char(61), back = .true.) + 1
00930
00931 ! Determine if KEYWORD is followed by double equal sign (==) indicating
00932 ! nested parameter.
00933 IF ((lstr - kend) == 3) nested = .true.
00934
00935 ! Extract KEYWORD, trim leading and trailing blanks.
00936 kextract = .false.
00937 IF (kend > 0) THEN
00938   lend = lenline
00939   keyword = line(kstr:kend)
00940   nval = 0
00941   kextract = .true.
00942 ELSE
00943   lstr = 1
00944   lend = lenline
00945   kextract = .true.
00946 END IF
00947
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 parsesline
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 !-----+
01146 !-----+
01147 INTEGER FUNCTION checkcontrolfileinputs() RESULT(errStatus)

```

```

01148
01149     USE pahm_global
01150     USE timedateutils, ONLY : firstgregdate, firstgregtime, &
01151         gregtojulday, joindate, gettimeconvsec
01152
01153     IMPLICIT NONE
01154
01155     ! Local variables
01156     INTEGER             :: errio, errnum
01157     INTEGER             :: icnt
01158     LOGICAL             :: filefound
01159     REAL(sz)           :: gregjd, refjd, jd0, jd1
01160     REAL(sz)           :: timesec
01161     CHARACTER(LEN=64)   :: tmpstr, tmpstr1, tmpstr2
01162
01163
01164     !----- Initialize variables
01165     errio      = 0
01166     errnum     = 0
01167     errstatus   = 0
01168     scratchmessage = blank
01169
01170
01171     CALL setmessagesource("CheckControlFileInputs")
01172
01173     !---- 1) Best track files (mandatory variables) ----
01174     IF (nbtrfiles <= 0) THEN
01175         errnum = 1
01176
01177         WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01178             ' . Invalid value supplied for dimension parameter: nBTrFiles = ', &
01179             nbtrfiles, ' (should be greater than zero).'
01180         CALL logmessage(error, scratchmessage)
01181     ELSE IF (besttrackfilenamespecified) THEN
01182         IF (numbtfiles /= nbtrfiles) THEN
01183             errnum = 2
01184
01185         WRITE(scratchmessage, '("errNum = ", i0, a, i0, a)') errnum, &
01186             ' . The number of files for <bestTrackFileName> should be equal to nBTrFiles:
01187         ', &
01188             nbtrfiles, '.'
01189         CALL logmessage(error, scratchmessage)
01190     END IF
01191
01192     DO icnt = 1, numbtfiles
01193         INQUIRE(file=trim(adjustl(besttrackfilename(icnt))), iostat=errio, exist=filefound)
01194         IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01195             errnum = 3
01196
01197             WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01198                 ' . Could not access the best track file for read: ' // &
01199                 trim(adjustl(besttrackfilename(icnt)))
01200             CALL logmessage(error, scratchmessage)
01201         END IF
01202     END DO
01203
01204     ELSE
01205         errnum = 4
01206
01207         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01208             ' . bestTrackFileName(s) not specified. This is a mandatory variable. '
01209         CALL logmessage(error, scratchmessage)
01210     END IF
01211
01212     !---- 2) Mesh file (mandatory variables) ----
01213     IF (.NOT. meshfilenamespecified) THEN
01214         errnum = 5
01215
01216         WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01217             ' . Parameter <meshFileName> is not specified (mandatory variable) '
01218         CALL logmessage(error, scratchmessage)
01219
01220     ELSE
01221         meshfilename = adjustl(meshfilename)
01222         INQUIRE(file=trim(meshfilename), iostat=errio, exist=filefound)
01223         IF ((.NOT. filefound) .OR. (errio /= 0)) THEN
01224             errnum = 6
01225
01226             WRITE(scratchmessage, '("errNum = ", i0, a)') errnum, &
01227                 ' . Could not access the mesh file for read: ' // trim(meshfilename)
01228             CALL logmessage(error, scratchmessage)
01229         END IF
01230     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          meshfiletype = 'ADCIRC'
01241
01242          WRITE(scratchmessage, '(a)') 'This value of meshFileType is adjusted to: meshFileType = ' //
01243          trim(meshfiletype)
01244          CALL logmessage(info, scratchmessage)
01245
01246      ! Check for meshFileForm
01247      meshfileform = toupper(trim(adjustl(meshfileform)))
01248      SELECT CASE (trim(meshfileform))
01249          !CASE ('ASCII', 'NETCDF')
01250          CASE ('ASCII')
01251              ! These are valid values
01252
01253      CASE DEFAULT
01254          WRITE(scratchmessage, '(a)') 'This file format is not supported: meshFileForm = ' //
01255          trim(meshfileform)
01256          CALL logmessage(info, scratchmessage)
01257
01258          meshfileform = 'ASCII'
01259
01260          WRITE(scratchmessage, '(a)') 'This value of meshFileForm is adjusted to: meshFileForm = ' //
01261          trim(meshfileform)
01262          CALL logmessage(info, scratchmessage)
01263
01264      !---- 3) Reference date and time (mandatory variables) -----
01265      gregjd = gregtojulday(firstgregdate, firstgregtime)
01266      refjd = gregtojulday(refdate, reftime)
01267      IF (refjd < gregjd) THEN
01268          errnum = 7
01269          WRITE(scratchmessage, '(errNum = ', i0, a)') errnum, &
01270              '. Invalid DateTime string was supplied: refDateTime = ' // trim(refdatetime)
01271          CALL logmessage(error, scratchmessage)
01272      END IF
01273
01274      !---- 4) Stepping parameters (mandatory variables) -----
01275      ! check for valid start time
01276      IF (begsimspecified) THEN
01277          IF (refjd + (mdbegsimtime * gettimeconvsec('D', 1)) < gregjd) THEN
01278              errnum = 8
01279              WRITE(tmpstr, '(f20.5)') begsimtime
01280              WRITE(scratchmessage, '(errNum = ', i0, a)') errnum, &
01281                  '. Invalid start time in reference to refDateTime was supplied: begSimTime =
01282                  ' // &
01283                  trim(adjustl(tmpstr))
01284          END IF
01285      ELSE
01286          errnum = 81
01287          WRITE(scratchmessage, '(errNum = ', i0, a)') errnum, &
01288              '. Neither "begDateTime" or "begSimTime" are defined properly'
01289          CALL logmessage(error, scratchmessage)
01290      END IF
01291
01292      ! check for valid stop time
01293      IF (endsimspecified) THEN
01294          IF (comparereals(endsimtime, begsimtime, closetol) <= 0) THEN
01295              errnum = 9
01296              WRITE(tmpstr1, '(f20.5)') begsimtime
01297              WRITE(tmpstr2, '(f20.5)') endsimtime
01298              WRITE(scratchmessage, '(errNum = ', i0, a)') errnum, &
01299                  '. Stop time should be greater than start time: begSimTime = ' // &
01300                  trim(adjustl(tmpstr1)) // ', endSimTime = ' // trim(adjustl(tmpstr2))
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(trimstr1)) // &
01397             ' is adjusted to: gravity = ' // trim(adjustl(trimstr2))
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(trimstr1)) // &
01410             ' is adjusted to: rhoWater = ' // trim(adjustl(trimstr2))
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(trimstr1)) // &
01423             ' is adjusted to: rhoAir = ' // trim(adjustl(trimstr2))
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(trimstr1)) // &
01436             ' is adjusted to: backgroundAtmPress = ' // trim(adjustl(trimstr2))
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(trimstr1)) // &
01447             ' is adjusted to: windReduction = ' // trim(adjustl(trimstr2))
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 !-----
01462 !-----
01463 !-----
01464 !----- INTEGER FUNCTION loadintvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01465
01466

```

```

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 !-----  

01529 !-----  

01530 !-----  

01531      INTEGER FUNCTION loadlogvar (nInp, vInp, nOut, vOut) RESULT(nValsOut)
01532
01533      IMPLICIT NONE
01534
01535      INTEGER, INTENT(IN) :: ninp, nout
01536      CHARACTER(LEN=*) , INTENT(IN) :: vinp(ninp)
01537      LOGICAL, INTENT(OUT) :: vout(nout)
01538
01539      INTEGER :: i, ic
01540
01541 !-----
01542 ! Load INTEGER variable with input values.
01543 !-----
01544
01545 ! If not all values are provided for variable, assume the last value
01546 ! for the rest of the array.
01547      ic = 0
01548      IF (ninp <= nout) THEN
01549          DO i = 1, ninp
01550              ic = ic + 1
01551              IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01552                  vout(i) = .true.
01553              ELSE
01554                  vout(i) = .false.
01555              END IF
01556          END DO
01557          DO i = ninp + 1, nout
01558              ic = ic + 1
01559              IF ((vinp(ninp)(1:1) == 'T') .OR. (vinp(ninp)(1:1) == 't')) THEN
01560                  vout(i) = .true.
01561              ELSE
01562                  vout(i) = .false.
01563              END IF
01564          END DO
01565      ELSE
01566          DO i = 1, nout
01567              ic = ic + 1
01568              IF ((vinp(i)(1:1) == 'T') .OR. (vinp(i)(1:1) == 't')) THEN
01569                  vout(i) = .true.
01570              ELSE
01571                  vout(i) = .false.
01572              END IF
01573          END DO
01574      END IF
01575
01576      nvalsout = ic
01577
01578  END FUNCTION loadlogvar

```

```

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 !-----
01607 !
01608 !
01609 INTEGER FUNCTION loadrealvar(nInp, vInp, nOut, vOut) RESULT(nValsOut)
01610
01611 IMPLICIT NONE
01612
01613 INTEGER, INTENT(IN) :: ninp, nout
01614 REAL(sz), INTENT(IN) :: vinp(ninp)
01615 REAL(sz), INTENT(OUT) :: vout(nout)
01616
01617 INTEGER :: i, ic
01618
01619 !-----
01620 ! Load INTEGER variable with input values.
01621 !-----
01622
01623 ! If not all values are provided for variable, assume the last value
01624 ! for the rest of the array.
01625 ic = 0
01626 IF (ninp <= nout) THEN
01627     DO i = 1, ninp
01628         ic = ic + 1
01629         vout(i) = vinp(i)
01630     END DO
01631     DO i = ninp + 1, nout
01632         ic = ic + 1
01633         vout(i) = vinp(ninp)
01634     END DO
01635 ELSE
01636     DO i = 1, nout
01637         ic = ic + 1
01638         vout(i) = vinp(i)
01639     END DO
01640 END IF
01641
01642
01643 nvalsout = ic
01644
01645 RETURN
01646
01647 END FUNCTION loadrealvar
01648
01649 !=====
01650
01651 !-----
01652 ! F U N C T I O N   T O   L O W E R   C A S E
01653 !-----
01654 !
01655 PURE FUNCTION tolowercase(inpString) RESULT(outString)
01656
01657 IMPLICIT NONE
01658
01659 CHARACTER(*), INTENT(IN) :: inpstring
01660
01661 INTEGER, PARAMETER :: duc = ichar('A') - ichar('a')
01662 CHARACTER(LEN(inpString)) :: outstring
01663 CHARACTER :: ch
01664 INTEGER :: i
01665
01666 DO i = 1, len(inpstring)
01667     ch = inpstring(i:i)
01668     IF ((ch >= 'A') .AND. (ch <= 'Z')) ch = char(ichar(ch) - duc)
01669     outstring(i:i) = ch
01670 END DO
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     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 !
01853 !
01854 SUBROUTINE geotocpp_1d(lat, lon, lat0, lon0, x, y)
01855

```

```

01856     USE pahm_global, ONLY : rearth, deg2rad
01857
01858     IMPLICIT NONE
01859
01860     REAL(SZ), INTENT(IN) :: lat(:)
01861     REAL(SZ), INTENT(IN) :: lon(:)
01862     REAL(SZ), INTENT(IN) :: lat0
01863     REAL(SZ), INTENT(IN) :: lon0
01864     REAL(SZ), INTENT(OUT) :: x(:)
01865     REAL(SZ), INTENT(OUT) :: y(:)
01866
01867     x = deg2rad * rearth * (lon - lon0) * cos(lat0)
01868     y = deg2rad * rearth * lat
01869
01870 END SUBROUTINE geotocpp_1d
01871
01872 !=====
01873 !-----
01874 !----- S U B R O U T I N E   C P P   T O   G E O   S C A L A R
01875 !-----
01876 !-----
01877 !-----
01878 !----- SUBROUTINE cpptogeoo_scalar(x, y, lat0, lon0, lat, lon)
01879
01880 USE pahm_global, ONLY : rearth, deg2rad
01881
01882     IMPLICIT NONE
01883
01884     REAL(SZ), INTENT(IN) :: x
01885     REAL(SZ), INTENT(IN) :: y
01886     REAL(SZ), INTENT(IN) :: lat0
01887     REAL(SZ), INTENT(IN) :: lon0
01888     REAL(SZ), INTENT(OUT) :: lat
01889     REAL(SZ), INTENT(OUT) :: lon
01890
01891     lat = y / (deg2rad * rearth)
01892     lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01893
01894 END SUBROUTINE cpptogeoo_scalar
01895
01896 !=====
01897 !-----
01898 !----- S U B R O U T I N E   C P P   T O   G E O   1D
01899 !-----
01900 !-----
01901 !----- SUBROUTINE cpptogeoo_1d(x, y, lat0, lon0, lat, lon)
01902
01903 USE pahm_global, ONLY : rearth, deg2rad
01904
01905     IMPLICIT NONE
01906
01907     REAL(SZ), INTENT(IN) :: x(:)
01908     REAL(SZ), INTENT(IN) :: y(:)
01909     REAL(SZ), INTENT(IN) :: lat0
01910     REAL(SZ), INTENT(IN) :: lon0
01911     REAL(SZ), INTENT(OUT) :: lat(:)
01912     REAL(SZ), INTENT(OUT) :: lon(:)
01913
01914     lat = y / (deg2rad * rearth)
01915     lon = lon0 + x / (deg2rad * rearth * cos(deg2rad * lat0))
01916
01917 END SUBROUTINE cpptogeoo_1d
01918
01919 !-----
01920 !-----
01921 !----- 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
01922 !-----
01923 !-----
01924 !----- REAL(sz) function sphericaldistance_scalar(lat1, lon1, lat2, lon2) result(myvalout)
01925
01926 USE pahm_global, ONLY : rearth, deg2rad
01927
01928     IMPLICIT NONE
01929
01930     REAL(sz), INTENT(IN) :: lat1      ! latitude of point 1 on the sphere (degrees north)
01931     REAL(sz), INTENT(IN) :: lon1      ! longitude of point 1 on the sphere (degrees east)
01932     REAL(sz), INTENT(IN) :: lat2      ! latitude of point 2 on the sphere (degrees north)
01933     REAL(sz), INTENT(IN) :: lon2      ! longitude of point 2 on the sphere (degrees east)
01934
01935

```

```

02011      REAL(sz)          :: phil, phi2, lamda1, lamda2, dphi, dlamda, dsigma
02012
02013      phil   = deg2rad * lat1
02014      phi2   = deg2rad * lat2
02015      dphi   = abs(phi2 - phil)
02016
02017      lamda1 = deg2rad * lon1
02018      lamda2 = deg2rad * lon2
02019      dlamda = abs(lamda2 - lamda1)
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 ! -----
02036 ! 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
02037 ! -----
02038 !
02039 !-----
02040 FUNCTION sphericaldistance_1d(lats, lons, lat0, lon0) RESULT(myValOut)
02041
02042     USE pahm_global, ONLY : rearth, deg2rad
02043
02044     IMPLICIT NONE
02045
02046     ! Global variables
02047     REAL(sz), INTENT(IN) :: lats(:) ! latitude of point 1 on the sphere (degrees north)
02048     REAL(sz), INTENT(IN) :: lons(:) ! longitude of point 1 on the sphere (degrees east)
02049     REAL(sz), INTENT(IN) :: lat0    ! latitude of point 2 on the sphere (degrees north)
02050     REAL(sz), INTENT(IN) :: lon0    ! longitude of point 2 on the sphere (degrees east)
02051
02052     REAL(sz), DIMENSION(:), ALLOCATABLE :: myvalout
02053
02054     ! Local variables
02055     REAL(sz), DIMENSION(:), ALLOCATABLE :: phis, lamdas, dphi, dlamda, dsigma
02056     REAL(sz)                         :: phi0, lamda0
02057     INTEGER                           :: status, n1
02058
02059
02060     CALL setmessagesource("SphericalDistance_1D")
02061
02062     IF (SIZE(lats) /= SIZE(lons)) THEN
02063         WRITE(scratchmessage, '(a)') 'The size of arrays "lats" and "lons" is not the same.'
02064         CALL allmessage(error, scratchmessage)
02065
02066         CALL terminate()
02067     END IF
02068
02069     n1 = SIZE(lats, 1)
02070     ALLOCATE(myvalout(n1), stat = status)
02071     ALLOCATE(phis(n1), lamdas(n1), dphi(n1), dlamda(n1), dsigma(n1), stat = status)
02072
02073     IF (status /= 0) THEN
02074         WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02075         CALL allmessage(error, scratchmessage)
02076
02077         CALL terminate()
02078     END IF
02079
02080     phis   = deg2rad * lats
02081     phi0   = deg2rad * lat0
02082     dphi   = abs(phi0 - phis)
02083
02084     lamdas = deg2rad * lons
02085     lamda0 = deg2rad * lon0
02086     dlamda = abs(lamda0 - lamdas)
02087
02088     ! Vincenty formula to calculate a distance along a sphere
02089     dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
02090                      (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
02091     dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))
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 ! This is the great-circle distance; REARTH in meters
02122 myvalout = rearth * dsigma
02123
02124 DEALLOCATE(phis, 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 :: phis, 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
02173 ALLOCATE(myvalout(n1, n2), stat = status)
02174 ALLOCATE(phis(n1, n2), lamdas(n1, n2), dphi(n1, n2), dlamda(n1, n2), dsigma(n1, n2), stat = status)
02175
02176 IF (status /= 0) THEN
02177   WRITE(scratchmessage, '(a)') 'Could no allocate memory for the internal arrays.'
02178   CALL allmessage(error, scratchmessage)
02179
02180   CALL unsetmessagesource()
02181
02182   CALL terminate()
02183 END IF
02184
02185 phis = deg2rad * lats
02186 phi0 = deg2rad * lat0
02187 dphi = abs(phi0 - phis)
02188
02189 lamdas = deg2rad * lons
02190 lamda0 = deg2rad * lon0
02191 dlamda = abs(lamda0 - lamdas)
02192
02193 ! Vincenty formula to calculate a distance along a sphere
02194 dsigma = atan(sqrt((cos(phi0) * sin(dlamda))**2 + &
02195                 (cos(phis) * sin(phi0) - sin(phis) * cos(phi0) * cos(dlamda))**2))
02196 dsigma = dsigma / (sin(phis) * sin(phi0) + cos(phis) * cos(phi0) * cos(dlamda))
02197
02198 ! This is the great-circle distance; REARTH in meters
02199 myvalout = rearth * dsigma
02200
02201 DEALLOCATE(phis, lamdas, dphi, dlamda, dsigma)
02202
02203 CALL unsetmessagesource()

```

```

02231
02232      RETURN
02233
02234  END FUNCTION sphericaldistance_2d
02235
02236 !=====
02237
02238 ! -----
02239 !   F   U   N   C   T   O   N   S   P   H   E   R   I   C   A   L   D   I   S   T   A   N   C   E   H   A   R   V
02240 ! -----
02241
02242 !-----
02243 REAL(sz) function sphericaldistancehav(lat1, lon1, lat2, lon2) result(myvalout)
02244
02245 USE pahm_global, ONLY : rearth, deg2rad
02246
02247 IMPLICIT NONE
02248
02249     REAL(sz), INTENT(IN) :: lat1      ! latitude of point 1 on the sphere (degrees north)
02250     REAL(sz), INTENT(IN) :: lon1      ! longitude of point 1 on the sphere (degrees east)
02251     REAL(sz), INTENT(IN) :: lat2      ! latitude of point 2 on the sphere (degrees north)
02252     REAL(sz), INTENT(IN) :: lon2      ! longitude of point 2 on the sphere (degrees east)
02253
02254     REAL(sz)              :: phil, phi2, lamdal, lamda2, dphi, dlamda, dsigma
02255
02256     phil    = deg2rad * lat1
02257     phi2   = deg2rad * lat2
02258     dphi   = abs(phi2 - phil)
02259
02260     lamdal = deg2rad * lon1
02261     lamda2 = deg2rad * lon2
02262     dlamda = abs(lamda2 - lamdal)
02263
02264     ! Haversine formula formula to calculate a distance along a sphere
02265     dsigma = sqrt(sin(dphi / 2.0_sz)**2 + cos(phi1) * cos(phi2) * sin(dlamda / 2.0_sz)**2)
02266     dsigma = 2.0_sz * asin(dsigma)
02267
02268     ! This is the great-circle distance; REARTH in meters
02269     myvalout = rearth * dsigma
02270
02271     RETURN
02272
02273 END FUNCTION sphericaldistancehav
02274
02275 !=====
02276
02277 ! -----
02278 !   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
02279 ! -----
02280 !-----
02281 SUBROUTINE sphericalfracpoint(lat1, lon1, lat2, lon2, fraction, latf, lonf, distf, dist12)
02282
02283 USE pahm_global, ONLY : rearth, deg2rad, rad2deg
02284
02285 IMPLICIT NONE
02286
02287 ! Global variables
02288     REAL(SZ), INTENT(IN)      :: lat1      ! latitude of point 1 on the sphere (degrees north)
02289     REAL(SZ), INTENT(IN)      :: lon1      ! longitude of point 1 on the sphere (degrees east)
02290     REAL(SZ), INTENT(IN)      :: lat2      ! latitude of point 2 on the sphere (degrees north)
02291     REAL(SZ), INTENT(IN)      :: lon2      ! longitude of point 2 on the sphere (degrees east)
02292     REAL(SZ), INTENT(IN)      :: fraction   ! distance fraction of the intermediate point (0 <= f
02293     <= 1)
02294     REAL(SZ), INTENT(OUT)     :: latf, lonf ! the calculated latitude and longitude of the
02295                                         ! intermediate point
02296     REAL(SZ), OPTIONAL, INTENT(OUT) :: distf   ! the distance between point 1 and the intermediate
02297                                         ! point
02298     REAL(SZ), OPTIONAL, INTENT(OUT) :: dist12  ! the distance between point 1 and point 2
02299
02300 ! Local variables
02301     REAL(SZ)                  :: myFrac
02302     REAL(SZ)                  :: phil, phi2, lamdal, lamda2, delta
02303     REAL(SZ)                  :: aa, bb, xx, yy, zz
02304     REAL(SZ) :: myDist12, myDistF
02305
02306     myfrac = fraction
02307     IF (myfrac < 0) myfrac = 0.0_sz
02308     IF (myfrac > 1) myfrac = 1.0_sz
02309
02310     ! Calculate the great circle distance between points 1 and 2
02311     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 !-----
02414
02415 !-----
02416
02417 SUBROUTINE getlocandratio(val, arrVal, idx1, idx2, wtRatio)
02418
02419     IMPLICIT NONE
02420
02421     ! Global variables
02422     REAL(SZ), INTENT(IN) :: val           ! value to search for
02423     REAL(SZ), INTENT(IN) :: arrVal(:)      ! search array (1D)
02424     INTEGER, INTENT(OUT) :: idx1           ! the index of the lowest bound
02425     INTEGER, INTENT(OUT) :: idx2           ! the index of the highest bound
02426     REAL(SZ), INTENT(OUT) :: wtRatio        ! the ratio factor that used in the linear interpolation
02427                                         ! calculations: F = F(idx1) + wtRatio * (F(idx2) - F(idx1))
02428                                         ! 0 <= wtRatio <= 1.0
02429
02430     ! Local variables
02431     INTEGER                  :: nn, jl, j11, j12
02432     REAL(SZ)                 :: diffVal
02433
02434
02435     idx1 = -1
02436     idx2 = -1
02437     wtratio = 0.0_sz
02438
02439     nn = SIZE(arrval, 1)
02440     jl = minloc(abs(val - arrval), 1)
02441
02442     !----- Check if we got an exact bin value
02443     IF (comparereals(val - arrval(jl), 0.0_sz) == 0) THEN
02444         idx1 = jl
02445         idx2 = jl
02446         wtratio = 0.0_sz
02447
02448         RETURN
02449     END IF
02450
02451 !-----
02452
02453
02454
02455
02456
02457
02458
02459
02460
02461
02462
02463
02464
02465
02466
02467
02468
02469
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     jl1 = jl
02476     jl2 = jl + 1
02477   ELSE
02478     jl1 = jl - 1
02479     jl2 = jl
02480   END IF
02481
02482   diffval = arrval(jl2) - arrval(jl1)
02483
02484   IF (comparereals(diffval, 0.0_sz) == 0) THEN
02485     idx1 = jl1
02486     idx2 = jl1
02487     wtratio = 0.0_sz
02488
02489   ELSE
02490     IF (comparereals(val - arrval(jl1), 0.0_sz) * &
02491        comparereals(val - arrval(jl2), 0.0_sz) < 0) THEN
02492       idx1 = jl1
02493       idx2 = jl2
02494       wtratio = (val - arrval(jl1)) / 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   jl1 = jl - 1
02506   jl2 = jl
02507
02508   diffval = arrval(jl2) - arrval(jl1)
02509
02510   idx1 = jl1
02511   idx2 = jl2
02512   wtratio = (val - arrval(jl1)) / diffval
02513 ELSE IF (comparereals(val - arrval(jl), 0.0_sz) * &
02514    comparereals(val - arrval(jl + 1), 0.0_sz) < 0) THEN
02515
02516   jl1 = jl
02517   jl2 = jl + 1
02518
02519   diffval = arrval(jl2) - arrval(jl1)
02520
02521   idx1 = jl1
02522   idx2 = jl2
02523   wtratio = (val - arrval(jl1)) / diffval
02524 END IF
02525
02526   RETURN
02527
02528 END SUBROUTINE getlocandratio
02529
02530 !=====
02531
02532 !----- F U N C T I O N  C H A R  U N I Q U E
02533 !-----
02534 !
02535 !
02536 INTEGER FUNCTION charunique(inpVec, outVec, idxVec) RESULT (myRec)
02537
02538 IMPLICIT NONE
02539
02540 CHARACTER(LEN=*) , INTENT(IN) :: inpvec(:)
02541 CHARACTER(LEN=*) , INTENT(OUT) :: outvec(:)
02542 INTEGER, ALLOCATABLE, INTENT(OUT) :: idxvec(:)
02543
02544 CHARACTER(LEN=LEN(inpVec(1))), ALLOCATABLE :: chkstr(:)
02545 INTEGER, ALLOCATABLE :: chkint(:)
02546 INTEGER :: nels
02547 INTEGER :: icnt, jcnt ! counters
02548
02549 nels = SIZE(inpvec, 1)
02550
02551 ALLOCATE(chkstr(nels))

```

```

02570     ALLOCATE(chkint(nels))
02571
02572
02573     jcnt = 1
02574     DO icnt = 1, nels
02575       IF (trim(inpvec(icnt)) == "")      cycle
02576       IF (any(chkstr == inpvec(icnt))) cycle
02577
02578       ! No match found so add it to the output
02579       chkstr(jcnt) = inpvec(icnt)
02580       chkint(jcnt) = icnt
02581       jcnd = jcnd + 1
02582   END DO
02583
02584   myrec = jcnd - 1
02585   outvec = chkstr
02586   idxvec = chkint
02587
02588   DEALLOCATE(chkstr)
02589   DEALLOCATE(chkint)
02590
02591   RETURN
02592
02593 END FUNCTION charunique
02594
02595 !=====
02596
02597 !-----
02598 ! F U N C T I O N   V A L   S T R
02599 !-----
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 !-----
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 !-----+
02726 !
02727
02728 !-----+
02729      INTEGER FUNCTION realscan(String, Pos, Value) Result(myVal)
02730
02731      IMPLICIT NONE
02732
02733      ! Dummy arguments
02734      INTEGER, INTENT(IN)          :: pos
02735      CHARACTER(LEN=*) , INTENT(IN) :: string
02736      REAL(sp), INTENT(OUT)        :: value
02737
02738      ! Local variables
02739      INTEGER :: fract, intg, kfract, pmsign, power, ptr
02740
02741      ! CHECK POS.
02742      myval = pos
02743      Value = 0.0_sp
02744      IF(pos < 1 .OR. len(string) < pos)RETURN
02745
02746      ! SET UP WORKING VARIABLES.
02747      intg = 0
02748      fract = 0
02749      kfract = 0
02750      power = 0
02751      DO WHILE (.true.)
02752          ! SKIP LEADING BLANKS.
02753          IF(string(myval:myval) == ' ') THEN
02754              myval = myval + 1
02755              IF(myval > len(string))RETURN
02756              cycle
02757          END IF
02758
02759          ! LOOK FOR SIGN.
02760          ! NOTE: SEPARATE CHECK FOR SIGN SINCE INTEGER PART MAY BE OMITTED.
02761          pmsign = 0
02762          IF(string(myval:myval) == '+') THEN
02763              pmsign = +1
02764          ELSE IF(string(myval:myval) == '-') THEN
02765              pmsign = -1
02766          END IF
02767          IF(pmsign.NE.0)myval = myval + 1
02768
02769          ! LOOK FOR INTEGER PART.
02770          myval = intscan(string,myval,.false.,intg)
02771
02772          ! LOOK FOR FRACTION PART.
02773          IF(myval.LE.len(string)) THEN
02774              IF(myval > pos+abs(pmsign)) THEN
02775                  ! DETERMINE IF FIRST FORM OR SECOND FORM.
02776                  ! HANDLE FIRST FORM:  D+ ['.'] D*
02777                  IF(string(myval:myval) == '.') THEN
02778                      myval = myval + 1
02779                  IF(myval.LE.len(trim(string))) THEN
02780                      IF(string(myval:myval).NE.' ') THEN
02781                          ptr = intscan(string,myval,.false.,fract)
02782                          kfract = ptr - myval
02783                          myval = ptr
02784                      END IF
02785                  END IF
02786              END IF
02787          ! 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      ELSE
02829          myval = myval + 1
02830          ptr = intscan(string,myval,.false.,fract)
02831          kfract = ptr - myval
02832          IF(kfract == 0) THEN
02833              ! IF FRACTION MISSING, THEN WE STILL HAVE NOTHING.
02834              myval = pos
02835              RETURN
02836          ELSE
02837              myval = ptr
02838          END IF
02839      END IF
02840
02841      ! LOOK FOR EXPONENT PART.
02842      IF(myval.LE.len(string)) THEN
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  END IF
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 !-----+
02918 !-----
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, 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     ! COMPUTE REAL VALUE FROM ITS PARTS.
03016     IF(kfract.NE.0) THEN
03017         Value = (intg+fract/10.0**kfract)*10.0**power
03018     ELSE
03019         Value = intg*10.0**power
03020     END IF
03021     IF(pmsign == -1) Value = -Value
03022     EXIT
03023

```

```

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

```

17.43 /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::spinterp](#) (angle, dist, opt)
Spatial Interpolation function based on angle and r.
- real(sz) function, public [pahm_vortex::interpr](#) (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`

17.43.1 Detailed Description

Author

Panagiotis Velissariou panagiotis.velissariou@noaa.gov

Note

Adopted from the ADCIRC source code.

Definition in file [vortex.F90](#).

17.44 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, uvpr, uvtrans, fang,  

00027      &  

00028      getshapeparameter, setshapeparameter,  

00029      &  

00030      getlatestmax, 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      &  

00052      rmaxes4, quadflag4, quadir4, bs4, vmb14, &  

00053      &  

00054      calcintensitychange, uvtranspoint  

00055  

00056 PRIVATE  

00057  

00058 INTEGER, PARAMETER      :: nquads = 4           ! Number of quadrants for which wind radii are  

00059 provided  

00060 INTEGER, PARAMETER      :: npoints = nquads + 2    ! Number of (theta, rMax) points for curve fit  

00061 REAL(sz), DIMENSION(NPOINTS) :: rmaxes          ! Radius of maximum winds  

00062 REAL(sz), DIMENSION(NPOINTS, 4) :: rmaxes4        ! (nautical miles)  

00063  

00064 REAL(sz)                  :: pn                ! Ambient surface pressure (mb) !PV global  

00065 var?  

00066 REAL(sz)                  :: pc                ! Surface pressure at center of storm (mb) !PV  

00067 global var?  

00068 REAL(sz)                  :: clat              ! Latitude of storm center (degrees north)  

00069 !PV global var?  

00070 REAL(sz)                  :: clon              ! Longitude of storm center (degrees east )  

00071 !PV global var?  

00072 REAL(sz)                  :: vmax              ! Max sustained wind velocity in storm (knots)  

00073 !PV global var?  

00074  

00075 REAL(sz)                  :: b                 ! Exponential shape parameter  

00076 REAL(sz)                  :: corio             ! Coriolis force (1/s)  

00077 REAL(sz)                  :: vr                ! Velocity @ wind radii (knots)

```

```

00057  REAL(sz)          :: phi
00058  REAL(sz), DIMENSION(NPOINTS)    :: phis           ! Correction factor to B and vh
00059  REAL(sz), DIMENSION(NPOINTS, 4) :: phis4          ! Correction factor to B and vh
00060
00061  REAL(sz), DIMENSION(NPOINTS)    :: bs
00062  REAL(sz), DIMENSION(NPOINTS, 4) :: bs4
00063  REAL(sz), DIMENSION(NPOINTS)    :: vmb1
00064  REAL(sz), DIMENSION(NPOINTS, 4) :: vmb14
00065  INTEGER, DIMENSION(NPOINTS, 4) :: quadflag4
00066  REAL(sz), DIMENSION(NPOINTS, 4) :: quadir4
00067  REAL(sz), DIMENSION(NQUADS)     :: vrquadrant
00068  REAL(sz), DIMENSION(NQUADS)     :: radius         ! Wind radii - the distance
00069
00070  INTEGER                   :: quad            ! Quadrant counter
00071
00072  REAL(sz)          :: latestrmax      ! most recently calculated value of fitted
rmax
00073  REAL(sz)          :: latestangle       ! angle of the most recently calculated node
w.r.t. the storm location
00074  LOGICAL             :: usequadrantvr
00075  LOGICAL             :: usevmaxesbl
00076
00077
00078  CONTAINS
00079
00080
00081 !-----
00082 ! S U B R O U T I N E   C A L C   I N T E N S I T Y   C H A N G E
00083 !-----
00107 !
00108 SUBROUTINE calcintensitychange(var, times, calcInt, status, order)
00109
00110 USE pahm_global, ONLY : deg2rad
00111 USE utilities, ONLY : sphericaldistance
00112
00113 IMPLICIT NONE
00114
00115 REAL(sz), DIMENSION(:), INTENT(IN)  :: var, times
00116 INTEGER, OPTIONAL, INTENT(IN)       :: order
00117
00118 REAL(sz), DIMENSION(:), INTENT(OUT) :: calcint
00119 INTEGER, INTENT(OUT)                 :: status
00120
00121 INTEGER                      :: ordacur
00122 REAL(sz)                     :: dt1, dt2
00123 LOGICAL                       :: dtlok, dt2ok
00124 REAL(sz)                     :: val1, val2
00125 INTEGER                      :: icnt, maxcnt
00126
00127 status = 0
00128 maxcnt = 0
00129
00130 CALL setmessagesource("CalcIntensityChange")
00131
00132 IF ((SIZE(shape(var)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
00133   WRITE(scratchmessage, '(a)') 'The rank of arrays var and times should be equal to 1 (vectors)'
00134   CALL allmessage(error, scratchmessage)
00135
00136 CALL unsetmessagesource()
00137
00138 status = 1
00139
00140   RETURN
00141 ELSE
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 orderd accuracy using backward differences
00153 IF (ordacur == 1 )THEN
00154   DO icnt = 2, maxcnt
00155     dt1 = times(icnt) - times(icnt - 1)
00156     dtlok = (comparereals(dt1, 0.0_sz) /=0)
00157     val1 = 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. dt1ok) THEN
00216              calcint(icnt) = vall
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      dt1ok = (comparereals(dt1, 0.0_sz) /=0)
00227      dt2 = times(icnt - 1) - times(icnt - 2)
00228      dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00229
00230      vall = 0.0_sz
00231      IF (dt1ok) vall = (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 (dt1ok .AND. dt2ok) THEN
00237          calcint(icnt) = ((2.0_sz * dt1 + dt2) / (dt1 + dt2)) * vall - (dt1 / (dt1 + dt2)) * val2
00238      ELSE IF (.NOT. dt1ok) THEN
00239          calcint(icnt) = vall

```

```

00240      ELSE
00241          calcint(icnt) = 2.0_sz * vall - val2
00242      END IF
00243      !---- Backward differences (last point)
00244
00245      CALL unsetmessagesource()
00246
00247  END SUBROUTINE calcintensitychange
00248
00249 !=====
00250
00251 !-----
00252 ! S U B R O U T I N E   U V   T R A N S
00253 !-----
00281 !
00282 SUBROUTINE uvtrans(lat, lon, times, u, v, status, order)
00283
00284     USE pahm_global, ONLY : deg2rad
00285     USE utilities, ONLY : sphericaldistance
00286
00287     IMPLICIT NONE
00288
00289     REAL(sz), DIMENSION(:, INTENT(IN) :: lat, lon, times
00290     INTEGER, OPTIONAL, INTENT(IN) :: order
00291
00292     REAL(sz), DIMENSION(:, INTENT(OUT) :: u, v
00293     INTEGER, INTENT(OUT) :: status
00294
00295     INTEGER :: ordacur
00296     REAL(sz) :: dx1, dy1, dx2, dy2
00297     REAL(sz) :: dt1, dt2
00298     LOGICAL :: dtlok, dt2ok
00299     REAL(sz) :: ul, u2, v1, v2
00300     INTEGER :: icnt, maxcnt
00301
00302     status = 0
00303     maxcnt = 0
00304
00305     CALL setmessagesource("UVTrans")
00306
00307     IF ((SIZE(shape(lat)) /= 1) .OR. (SIZE(shape(lon)) /= 1) .OR. (SIZE(shape(times)) /= 1)) THEN
00308         WRITE(scratchmessage, '(a)') 'The rank of arrays lat, lon and times should be equal to 1 (vectors)'
00309         CALL allmessage(error, scratchmessage)
00310
00311     CALL unsetmessagesource()
00312
00313     status = 1
00314
00315     RETURN
00316
00317     maxcnt = SIZE(lat)
00318 END IF
00319
00320     ordacur = 2
00321     IF (PRESENT(order)) THEN
00322         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 orded 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             dtlok = (comparereals(dt1, 0.0_sz) /=0)
00334
00335             u1 = 0.0_sz
00336             v1 = 0.0_sz
00337             IF (dtlok) 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
00405     ! Backward
00406     dx2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt - 1), lon(icnt))
00407     dy2 = sphericaldistance(lat(icnt - 1), lon(icnt - 1), lat(icnt), lon(icnt - 1))
00408     dt2 = abs(times(icnt) - times(icnt - 1))
00409     dt2ok = (comparereals(dt2, 0.0_sz) /=0)
00410
00411     u1 = 0.0_sz
00412     v1 = 0.0_sz
00413     IF (dt1ok) THEN
00414         u1 = sign(dx1 / dt1, (lon(icnt + 1) - lon(icnt)))
00415         v1 = sign(dy1 / dt1, (lat(icnt + 1) - lat(icnt)))
00416     END IF
00417
00418     u2 = 0.0_sz
00419     v2 = 0.0_sz
00420     IF (dt2ok) THEN
00421         u2 = sign(dx2 / dt2, (lon(icnt) - lon(icnt - 1)))
00422         v2 = sign(dy2 / dt2, (lat(icnt) - lat(icnt - 1)))
00423     END IF
00424
00425     IF (dt1ok .AND. dt2ok) THEN
00426         u(icnt) = (dt2 / (dt1 + dt2)) * u1 + (dt1 / (dt1 + dt2)) * u2
00427         v(icnt) = (dt2 / (dt1 + dt2)) * v1 + (dt1 / (dt1 + dt2)) * v2
00428 ELSE IF (.NOT. dt1ok) THEN
00429     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 !-----  

! S U B R O U T I N E   N E W   V O R T E X
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 !-----  

00575 !-----  

00576 !-----  

00577 !-----  

00578 !-----  

00579 !-----  

00580 !-----  

00581 !-----  

00582 !-----  

00583 !-----  

00584 !-----  

00585 !-----  

00586 !-----  

00587 !-----  

00588 !-----  

00589 !-----  

00590 !-----  

00591 !-----  

00592 !-----  

00593 !=====  

00594 !-----  

00595 !-----  

00596 !-----  

00597 !-----  

00598 !-----  

00599 !-----  

00600 !-----  

00601 !-----  

00602 !-----  

00603 !-----  

00604 !-----  

00605 !-----  

00606 !-----  

00607 !-----  

00608 !-----  

00609 !-----  

00610 !-----  

00611 !-----  

00612 !-----  

00613 !-----  

00614 !-----  

00615 !-----  

00616 !-----  

00617 !-----  

00618 !-----  

00619 !-----  

00620 !-----  

00621 !-----  

00622 !-----  

00623 !-----  

00624 !-----  

00625 !-----  

00626 !-----  

00627 !-----  

00628 !-----  

00629 !-----  

00630 !-----  

00631 !-----  

00632 !-----  

00633 !-----  

00634 !-----  

00635 !-----  

00636 !-----  

00637 !-----  

00638 !-----  

00639 !-----  

00640 !-----  

00641 !-----  

00642 !-----  

00643 !-----  

00644 !-----  

00645 !-----  

00646 !-----  

00647 !-----  

00648 !=====  

00649 !-----  

00650 !-----  


```

```

00651 ! S U B R O U T I N E   S E T   V O R T E X
00652 !-----
00669 !
00670 SUBROUTINE setvortex(pinf, p0, lat, lon)
00671 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 !-----S U B R O U T I N E   S E T   R M A X E S
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 !-----S U B R O U T I N E   G E T   R M A X E S
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 !-----S U B R O U T I N E   C A L C   R M A X E S
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 !-----
00811 ! S U B R O U T I N E   C A L C   R M A X E S   F U L L
00812 !-----
00813
00814 SUBROUTINE calcrmaxesfull()
00815
00816 USE pahm_global, ONLY : rhoair, nm2m, kt2ms, mb2pa
00817
00818 IMPLICIT NONE
00819
00820 REAL(sz) :: root          ! Radius of maximum winds
00821 REAL(sz), PARAMETER :: innerradius = 1.0_sz
00822 REAL(sz), PARAMETER :: outerradius = 500.0_sz
00823 REAL(sz), PARAMETER :: accuracy = 0.0001_sz
00824 REAL(sz), PARAMETER :: zoom = 0.01_sz
00825 INTEGER, PARAMETER :: itermax = 3
00826 REAL(sz) :: r1, r2, r3, r4, dr
00827 INTEGER :: n, iter, norootflag
00828 REAL(sz) :: bnew, bnewl
00829 REAL(sz) :: phinew
00830 INTEGER, PARAMETER :: cont = 400      ! Max # of iterations
00831 INTEGER :: icont, ibcont ! iteration counter
00832
00833 !-----
00834 ! Loop over quadrants of storm
00835 !-----
00836 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 rMxes, 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
00901             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 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 SUBROUTINE fitrmaxes4()
00965
00966     IMPLICIT NONE
00967
00968 !-----
00969 !-----
00970 !-----
00971 !-----
00972 !-----
00973     IMPLICIT NONE
00974
00975     ! Generate 2 additional points for curve-fit
00976     quadflag4(1, 1:4) = quadflag4(5, 1:4)
00977     quadflag4(6, 1:4) = quadflag4(2, 1:4)
00978
00979     quadir4(1, 1:4) = quadir4(5, 1:4)
00980     quadir4(6, 1:4) = quadir4(2, 1:4)
00981
00982     rmaxes4(1, 1:4) = rmaxes4(5, 1:4)
00983     rmaxes4(6, 1:4) = rmaxes4(2, 1:4)
00984
00985     bs4(1, 1:4) = bs4(5, 1:4)
00986     bs4(6, 1:4) = bs4(2, 1:4)
00987
00988     phis4(1, 1:4) = phis4(5, 1:4)
00989     phis4(6, 1:4) = phis4(2, 1:4)
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 !-----
00999 ! 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     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     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     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     IMPLICIT NONE  
01070  
01071     myvalout = b  
01072  
01073     RETURN  
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     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 !-----  
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 !-----  
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 !-----  
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 !-----  
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 !-----  
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 REAL(sz) function spinterp(angle, dist, opt) result(myvalout)
01206
01207     IMPLICIT NONE
01208
01209     REAL(sz), INTENT(IN)          :: angle, dist
01210     INTEGER, INTENT(IN)          :: opt
01211     REAL(sz), DIMENSION(NPOINTS, 4) :: param
01212     REAL(sz)                      :: templ, temp2
01213     REAL(sz)                      :: deltaangle
01214     INTEGER                        :: iquad
01215
01216     IF (opt == 1) THEN
01217         param = rmaxes4
01218     ELSE IF (opt == 2) THEN
01219         param = bs4
01220     ELSE IF (opt == 3) THEN
01221         param = vmb14
01222     END IF
01223
01224     deltaangle = 0.0_sz
01225
01226     ! CompareReals(r1, r2)
01227     ! -1 (if r1 < r2)
01228     !  0 (if r1 = r2)
01229     ! +1 (if r1 > r2)
01230
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         iquad = 2
01236         deltaangle = angle - 45.0_sz
01237     ELSE IF (comparereals(angle, 225.0_sz) /= 1) THEN
01238         iquad = 3
01239         deltaangle = angle - 135.0_sz
01240     ELSE IF (comparereals(angle, 315.0_sz) /= 1) THEN
01241         iquad = 4
01242         deltaangle = angle - 225.0_sz
01243     ELSE IF (angle > 315.0_sz) THEN
01244         iquad = 5
01245         deltaangle = angle - 315.0_sz
01246     END IF
01247
01248     ! nearest neighbor weighted interpolation
01249     IF (deltaangle < 1.0_sz) THEN
01250         myvalout = interpr(param, iquad, dist)
01251     ELSE IF (deltaangle > 89.0_sz) THEN
01252         myvalout = interpr(param, iquad + 1, dist)
01253     ELSE
01254         templ = interpr(param, iquad, dist)
01255         temp2 = interpr(param, iquad + 1, dist)
01256         myvalout = (templ / deltaangle**2 + temp2 / (90.0 - deltaangle)**2) /      &
01257                     (1.0_sz / deltaangle**2 + 1.0_sz / (90.0_sz - deltaangle)**2)
01258     END IF
01259
01260
01261
01262
01263
01264
01265
01266
01267
01268
01269
01270
01271
01272
01273
01274
01275

```

```

01276   END FUNCTION spinterp
01277
01278 !=====
01279
01280 !-----
01281 ! F U N C T I O N   I N T E R P R
01282 !-----
01283 REAL(sz) function interpr(quadval, quadsel, quaddis) result(myvalout)
01284
01285   IMPLICIT NONE
01286
01287   REAL(sz), DIMENSION(NPOINTS, 4), INTENT(IN) :: quadval
01288   INTEGER, INTENT(IN) :: quadsel
01289   REAL(sz), INTENT(IN) :: quaddis
01290
01291   REAL(sz) :: fac
01292   INTEGER :: totalisot
01293
01294   totalisot = sum(quadflag4(quadsel, :))
01295   SELECT CASE(totalisot)
01296     CASE(1)
01297       myvalout = quadval(quadsel, maxloc(quadflag4(quadsel, :), 1))
01298     CASE(2)
01299       IF (quaddis > quadir4(quadsel, 1)) THEN
01300         myvalout = quadval(quadsel, 1)
01301       ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01302         fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01303         myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01304       ELSE
01305         myvalout = quadval(quadsel, 2)
01306       END IF
01307     CASE(3)
01308       IF (quaddis > quadir4(quadsel, 1)) THEN
01309         myvalout = quadval(quadsel, 1)
01310       ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01311         fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01312         myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01313       ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
01314         fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
01315         myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
01316       ELSE
01317         myvalout = quadval(quadsel, 3)
01318       END IF
01319     CASE(4)
01320       IF (quaddis > quadir4(quadsel, 1)) THEN
01321         myvalout = quadval(quadsel, 1)
01322       ELSE IF (quaddis > quadir4(quadsel, 2)) THEN
01323         fac = (quaddis - quadir4(quadsel, 2)) / (quadir4(quadsel, 1) - quadir4(quadsel, 2))
01324         myvalout = quadval(quadsel, 1) * fac + quadval(quadsel, 2) * (1 - fac)
01325       ELSE IF (quaddis > quadir4(quadsel, 3)) THEN
01326         fac = (quaddis - quadir4(quadsel, 3)) / (quadir4(quadsel, 2) - quadir4(quadsel, 3))
01327         myvalout = quadval(quadsel, 2) * fac + quadval(quadsel, 3) * (1 - fac)
01328       ELSE IF (quaddis > quadir4(quadsel, 4)) THEN
01329         fac = (quaddis - quadir4(quadsel, 4)) / (quadir4(quadsel, 3) - quadir4(quadsel, 4))
01330         myvalout = quadval(quadsel, 3) * fac + quadval(quadsel, 4) * (1 - fac)
01331       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 !-----
01346 ! F U N C T I O N   R M W
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     !           (rMaxes(baseQuadrant + 1) - rMaxes(baseQuadrant)) +   &
01408     !           rMaxes(baseQuadrant)
01409
01410 END FUNCTION rmw
01411
01412 =====
01413
01414 !-----
01415 ! S U B R O U T I N E   U V P
01416 !-----
01417 !-----
01418
01419 SUBROUTINE uvp(lat, lon, uTrans, vTrans, u, v, p)
01420
01421 USE pahm_global, ONLY : windreduction, one2ten, deg2rad, rad2deg, mb2pa, kt2ms, nm2m, m2nm, rearth
01422
01423 IMPLICIT NONE
01424
01425 REAL(sz), INTENT(IN) :: lat
01426 REAL(sz), INTENT(IN) :: lon
01427 REAL(sz), INTENT(IN) :: utrans
01428 REAL(sz), INTENT(IN) :: vtrans
01429
01430 REAL(sz), INTENT(OUT) :: u
01431 REAL(sz), INTENT(OUT) :: v
01432 REAL(sz), INTENT(OUT) :: p
01433
01434 REAL(sz) :: transspd_x !NWS8-style translation speed
01435 REAL(sz) :: transspd_y !NWS8-style translation speed
01436
01437 REAL(sz) :: dx
01438 REAL(sz) :: dy
01439 REAL(sz) :: dist
01440 REAL(sz) :: rmx
01441 REAL(sz) :: angle
01442 REAL(sz) :: speed
01443 REAL(sz) :: uf
01444 REAL(sz) :: vf
01445 REAL(sz) :: percentcoriolis
01446 REAL(sz) :: speedatrmx
01447 REAL(sz) :: vmaxfactor
01448
01449 !-----
01450 ! Calculate distance and angle between eye of hurricane
01451 ! 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 speedatrmax = 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 / speedatrmax
01524
01525 ! Calculate NWS8-like translation speed
01526 transspdX = (abs(speed / speedatrmax)) * utrans * kt2ms
01527 transspdY = (abs(speed / speedatrmax)) * 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 uvpr
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 = (abs(speed / (vmax * kt2ms))) * utrans * kt2ms
01680     transspd = (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
01695     v = v + transspd
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 !----- F U N C T I O N   F A N G
01723 !----- F
01724 !
01725 !
01726 !----- REAL(sz) function fang(r, rmx) result(myvalout)
01727
01728 IMPLICIT NONE
01729
01730 REAL(sz), INTENT(IN) :: r
01731 REAL(sz), INTENT(IN) :: rmx
01732
01733 ! CompareReals(r1, r2)
01734 !    -1 (if r1 < r2)
01735 !    0 (if r1 = r2)
01736 !    +1 (if r1 > r2)
01737 IF (comparereals(0.0_sz, r) /= 1 .AND. comparereals(r, rmx) == -1) THEN
01738     myvalout = 10.0_sz * r / rmx
01739 ELSE IF (comparereals(rmx, r) /= 1 .AND. comparereals(r, 1.2_sz * rmx) == -1) THEN
01740     myvalout = 10.0_sz + 75.0_sz * (r / rmx - 1.0_sz)
01741 ELSE IF (comparereals(r, 1.2_sz * rmx) /= -1) THEN
01742     myvalout = 25.0_sz
01743 ELSE
01744     myvalout = 0.0_sz
01745 END IF
01746
01747 END FUNCTION fang
01748
01749 !=====
01750 !----- S U B R O U T I N E   R O T A T E
01751

```

```

01769 !-----
01794 !-----
01795 SUBROUTINE rotate(x, y, angle, whichWay, xr, yr)
01796 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 !
01851 !
01852 !-----
01853 REAL(sz) function vhwithcorifull(testrmax) result(myvalout)
01854
01855 USE pahm_global, ONLY : nm2m, kt2ms, ms2kt
01856
01857 IMPLICIT NONE
01858
01859 REAL(sz), INTENT(IN) :: testrmax
01860
01861 REAL(sz) :: thisvr ! the radial wind speed we've been given
01862 REAL(sz) :: vh
01863
01864 !
01865 !-----
01866 ! func(x = rMax) = vh - vr
01867 !
01868 IF (getusequadrantvr() .EQV. .true.) THEN
01869     thisvr = vrquadrant(quad)
01870 ELSE
01871     thisvr = vr
01872 END IF
01873
01874 vh = ms2kt * (sqrt(((vmax * kt2ms)**2 + vmax * kt2ms * testrmax * nm2m * corio) * &
01875                  (testrmax / radius(quad))**b) * &
01876
01877
01878
01879
01880
01881
01882
01883
01884
01885
01886
01887

```

```

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 vhwitħcorifull
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 vhwitħcori(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 vhwitħcori
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 vhnocori
01976
01977 !=====
01978 !-----+
01979 ! F U N C T I O N   F I N D   R O O T
01980 !-----+
01981 !-----+
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, pahm_global, 71
 341, 344
/home/takis/CSDL/parwinds-doc/src/csv_parameters.F90, pahm_messages, 101
 361
/home/takis/CSDL/parwinds-doc/src/csv_utilities.F90, allmessage_1
 362, 364
/home/takis/CSDL/parwinds-doc/src/driver_mod.F90, pahm_messages::allmessage, 254
 367, 368
/home/takis/CSDL/parwinds-doc/src/global.F90, pahm_messages, 101
 370, 373
/home/takis/CSDL/parwinds-doc/src/mesh.F90, 376, pahm_messages::allmessage, 254
 378
/home/takis/CSDL/parwinds-doc/src/messages.F90, allocasymvortstruct
 381, 383
 parwind, 162
/home/takis/CSDL/parwinds-doc/src/netcdfio.F90, allocatenodalandelementalarrays
 388, 390
 parwind, 162
/home/takis/CSDL/parwinds-doc/src/pahm.F90, allocbtrstruct
 400, 402
 parwind, 163
/home/takis/CSDL/parwinds-doc/src/parwind.F90, allochollstruct
 403, 404
 parwind, 163
/home/takis/CSDL/parwinds-doc/src/sizes.F90, 447, approach
 448
 parwind, 175
/home/takis/CSDL/parwinds-doc/src/sortutils.F90, arraycopydouble
 451, 454
 sortutils, 178
/home/takis/CSDL/parwinds-doc/src/timedateutils.F90, sortutils::arraycopy, 255
 473, 475
 sortutils, 179
/home/takis/CSDL/parwinds-doc/src/utilities.F90, arraycopyint
 489, 491
 sortutils::arraycopy, 256
 sortutils, 179
/home/takis/CSDL/parwinds-doc/src/vortex.F90, arraycopiesingle
 522, 524
 sortutils::arraycopy, 256
 sortutils, 179
add arrayequaldouble
 csv_module::csv_file, 289 sortutils, 180
add_cell arrayequalsingle
 csv_module, 33 sortutils::arrayequal, 258
 csv_module::csv_file, 290 sortutils, 181
add_matrix arthdouble
 csv_module, 34 sortutils, 181
 csv_module::csv_file, 290 sortutils::arth, 260
add_vector arthint
 csv_module, 35 sortutils, 182
 csv_module::csv_file, 290 sortutils::arth, 261
agrid arthsingle
 pahm_mesh, 96 sortutils, 183
airdensity asyvortstru

parwind, 175

b
pahm_vortex, 155

backgroundatmpress
pahm_global, 71

base_ncdfcheckerr
pahm_ncdfio, 114

basee
pahm_global, 71

basin
parwind::asymmetricvortexdata_t, 264
parwind::besttrackdata_t, 275
parwind::hollanddata_t, 310

begdate
pahm_global, 72

begdatespecified
pahm_global, 72

begdatetime
pahm_global, 72

begday
pahm_global, 72

beghour
pahm_global, 72

begmin
pahm_global, 73

begmonth
pahm_global, 73

begsec
pahm_global, 73

begsimspecified
pahm_global, 73

begsimtime
pahm_global, 73

begtime
pahm_global, 74

begyear
pahm_global, 74

besttrackdata
parwind, 176

besttrackfilename
pahm_global, 74

besttrackfilenamespecified
pahm_global, 74

blank
pahm_sizes, 126

bs
pahm_vortex, 155

bs4
pahm_vortex, 155

calcintensitychange
pahm_vortex, 132

calcrmaxes
pahm_vortex, 133

calcrmaxesfull
pahm_vortex, 133

casttime
parwind::asymmetricvortexdata_t, 265
parwind::hollanddata_t, 310

casttype
parwind::asymmetricvortexdata_t, 265
parwind::hollanddata_t, 310

casttypenum
parwind::asymmetricvortexdata_t, 265

charunique
utilities, 217

checkcontrolfileinputs
utilities, 218

chunk_size
csv_module::csv_file, 295

clat
pahm_vortex, 156

clon
pahm_vortex, 156

close
csv_module::csv_file, 290

close_csv_file
csv_module, 35

closelogfile
pahm_messages, 101

closetol
utilities, 244

cnttimebegin
pahm_drivermod, 68

cnttimeend
pahm_drivermod, 68

comparedoublereals
pahm_sizes, 124
pahm_sizes::comparereals, 283

comparesinglereals
pahm_sizes, 125
pahm_sizes::comparereals, 284

controlfilename
pahm_global, 74

convlon
utilities, 219

corio
pahm_vortex, 156

count
pahm_ncdfio::adcirccoorddata_t, 246
pahm_ncdfio::adcircvardata3d_t, 249
pahm_ncdfio::adcircvardata_t, 251

cpptogeo_1d
utilities, 219
utilities::cpptogeo, 285

cpptogeo_scalar
utilities, 220
utilities::cpptogeo, 286

cprdt
 parwind::hollanddata_t, 310
cpress
 parwind::asymetricvortexdata_t, 265
 parwind::hollanddata_t, 311
crdlats
 pahm_netcdfio, 119
crdlons
 pahm_netcdfio, 119
crdtime
 pahm_netcdfio, 119
crdxcs
 pahm_netcdfio, 119
crdycs
 pahm_netcdfio, 120
csv_data
 csv_module::csv_file, 295
csv_get_value
 csv_module, 36
 csv_module::csv_file, 291
csv_module, 31
 add_cell, 33
 add_matrix, 34
 add_vector, 35
 close_csv_file, 35
 csv_get_value, 36
 csv_type_double, 57
 csv_type_integer, 57
 csv_type_logical, 57
 csv_type_string, 58
 destroy_csv_file, 37
 get_character_column, 37
 get_column, 38
 get_csv_data_as_str, 39
 get_csv_string_column, 40
 get_header_csv_str, 41
 get_header_str, 42
 get_integer_column, 43
 get_logical_column, 44
 get_real_column, 45
 infer_variable_type, 46
 initialize_csv_file, 47
 next_row, 48
 number_of_lines_in_file, 48
 open_csv_file, 49
 read_csv_file, 50
 read_line_from_file, 51
 split, 52
 to_integer, 53
 to_logical, 53
 to_real, 54
 tokenize_csv_line, 55
 variable_types, 56
 zero, 58
csv_module::csv_file, 288
 add, 289
 add_cell, 290
 add_matrix, 290
 add_vector, 290
 chunk_size, 295
 close, 290
 csv_data, 295
 csv_get_value, 291
 delimiter, 295
 destroy, 291
 enclose_all_in_quotes, 295
 enclose_strings_in_quotes, 295
 get, 291
 get_character_column, 291
 get_column, 292
 get_csv_data_as_str, 292
 get_csv_string_column, 292
 get_header, 292
 get_header_csv_str, 292
 get_header_str, 293
 get_integer_column, 293
 get_logical_column, 293
 get_real_column, 293
 header, 295
 icol, 296
 initialize, 293
 iunit, 296
 logical_false_string, 296
 logical_true_string, 296
 n_cols, 296
 n_rows, 296
 next_row, 293
 open, 294
 quote, 297
 read, 294
 read_line_from_file, 294
 tokenize, 294
 variable_types, 294
csv_module::csv_string, 297
 str, 298
csv_parameters, 58
 default_int_fmt, 58
 default_real_fmt, 59
 max_integer_str_len, 59
 max_real_str_len, 59
csv_type_double
 csv_module, 57
csv_type_integer
 csv_module, 57
csv_type_logical
 csv_module, 57
csv_type_string
 csv_module, 58

csv_utilities, 59
expand_vector, 60
max_size_for_insertion_sort, 63
sortAscending, 60
swap, 61
unique, 62
csv_utilities.F90
partition, 363
quicksort, 363
cyclenum
parwind::besttrackdata_t, 276
cnum
parwind::besttrackdata_t, 276

datatmpres
pahm_netcdfio, 120
datelements
pahm_netcdfio, 120
datestimes
pahm_global, 75
datetime2string
timedateutils, 194
datwindx
pahm_netcdfio, 120
datwindy
pahm_netcdfio, 120
day
parwind::asymmetricvortexdata_t, 265
parwind::besttrackdata_t, 276
parwind::hollanddata_t, 311
dayofyear
timedateutils, 196
dayofyeartogreg
timedateutils, 197
deallocasymvortstruct
parwind, 164
deallocbtrstruct
parwind, 164
deallochollstruct
parwind, 166
debug
pahm_messages, 109
def_ncnam_pres
pahm_global, 75
def_ncnam_wndx
pahm_global, 75
def_ncnam_wndy
pahm_global, 75
default_int_fmt
csv_parameters, 58
default_real_fmt
csv_parameters, 59
defv_atmpress
pahm_global, 75

defv_gravity
pahm_global, 76
defv_rhoair
pahm_global, 76
defv_rhowater
pahm_global, 76
defv_windreduction
pahm_global, 76
deg2rad
pahm_global, 76
delimiter
csv_module::csv_file, 295
destroy
csv_module::csv_file, 291
destroy_csv_file
csv_module, 37
dimid
pahm_netcdfio::adcirccoorddata_t, 246
dir
parwind::asymmetricvortexdata_t, 265
parwind::besttrackdata_t, 276
dp
pahm_mesh, 96
drealscan
utilities, 221
dtg
parwind::asymmetricvortexdata_t, 266
parwind::besttrackdata_t, 276
parwind::hollanddata_t, 311
dvalstr
utilities, 222

echo
pahm_messages, 110
elapsedsecs
timedateutils, 197
elemdimid
pahm_netcdfio, 121
enclose_all_in_quotes
csv_module::csv_file, 295
enclose_strings_in_quotes
csv_module::csv_file, 295
enddate
pahm_global, 77
enddatespecified
pahm_global, 77
enddatetime
pahm_global, 77
endday
pahm_global, 77
endhour
pahm_global, 78
endmin
pahm_global, 78

endmonth
 pahm_global, 78
endsec
 pahm_global, 78
endsimspecified
 pahm_global, 78
endsimtime
 pahm_global, 79
endtime
 pahm_global, 79
endyear
 pahm_global, 79
error
 pahm_messages, 110
ew
 parwind::asymmetricvortexdata_t, 266
 parwind::besttrackdata_t, 276
expand_vector
 csv_utilities, 60
eye
 parwind::asymmetricvortexdata_t, 266
 parwind::besttrackdata_t, 277

fang
 pahm_vortex, 134
fcstinc
 parwind::asymmetricvortexdata_t, 266
 parwind::hollanddata_t, 311
filefound
 pahm_ncdfio::filedata_t, 299
filename
 pahm_ncdfio::filedata_t, 299
 parwind::asymmetricvortexdata_t, 266
 parwind::besttrackdata_t, 277
 parwind::hollanddata_t, 311
filereccounter
 pahm_ncdfio::filedata_t, 299
findroot
 pahm_vortex, 135
firstgregdate
 timedateutils, 212
firstgregtime
 timedateutils, 212
firtrmaxes
 pahm_vortex, 136
firtrmaxes4
 pahm_vortex, 136
fixnearwholedoublereal
 pahm_sizes, 125
 pahm_sizes::fixnearwholereal, 300
fixnearwholesinglereal
 pahm_sizes, 126
 pahm_sizes::fixnearwholereal, 301
fnamelen
 pahm_sizes, 127
geofactor
 parwind, 176
geostrophicswitch
 parwind, 176
geotocpp_1d
 utilities, 223
 utilities::geotocpp, 302
geotocpp_scalar
 utilities, 224
 utilities::geotocpp, 303
get
 csv_module::csv_file, 291
get_character_column
 csv_module, 37
 csv_module::csv_file, 291
get_column
 csv_module, 38
 csv_module::csv_file, 292
get_csv_data_as_str
 csv_module, 39
 csv_module::csv_file, 292
get_csv_string_column
 csv_module, 40
 csv_module::csv_file, 292
get_header
 csv_module::csv_file, 292
get_header_csv_str
 csv_module, 41
 csv_module::csv_file, 292
get_header_str
 csv_module, 42
 csv_module::csv_file, 293
get_integer_column
 csv_module, 43
 csv_module::csv_file, 293
get_logical_column
 csv_module, 44
 csv_module::csv_file, 293
get_real_column
 csv_module, 45
 csv_module::csv_file, 293
getgahmfields
 parwind, 166
gethollandfields
 parwind, 168
getlatestangle
 pahm_vortex, 136
getlatestrmax
 pahm_vortex, 137
getline record
 utilities, 224
getlocandratio

utilities, 225
getphifactors
 pahm_vortex, 137
getprogramcmdargs
 pahm_drivermod, 64
getrmaxes
 pahm_vortex, 137
getshapeparameter
 pahm_vortex, 138
getshapeparameters
 pahm_vortex, 138
gettmeconvsec
 timedateutils, 198
getusequadrantvr
 pahm_vortex, 138
getvmaxesbl
 pahm_vortex, 139
gravity
 pahm_global, 79
gregtojulday2
 timedateutils, 199
 timedateutils::gregtojulday, 304
gregtojuldayisec
 timedateutils, 200
 timedateutils::gregtojulday, 305
gregtojuldaysec
 timedateutils, 201
 timedateutils::gregtojulday, 307
gusts
 parwind::asymmetricvortexdata_t, 266
 parwind::besttrackdata_t, 277
header
 csv_module::csv_file, 295
hollb
 parwind::asymmetricvortexdata_t, 267
hollbs
 parwind::asymmetricvortexdata_t, 267
holSTRU
 parwind, 176
hour
 parwind::asymmetricvortexdata_t, 267
 parwind::besttrackdata_t, 277
 parwind::hollanddata_t, 311
hp
 pahm_sizes, 127
icol
 csv_module::csv_file, 296
icompxchg
 sortutils.F90, 453
icpress
 parwind::asymmetricvortexdata_t, 267
 parwind::hollanddata_t, 312
ics
 pahm_mesh, 97
idir
 parwind::asymmetricvortexdata_t, 267
ilat
 parwind::asymmetricvortexdata_t, 267
 parwind::hollanddata_t, 312
ilon
 parwind::asymmetricvortexdata_t, 268
 parwind::hollanddata_t, 312
imissv
 pahm_sizes, 127
indexxdouble
 sortutils, 183
 sortutils::indexx, 316
indexxitint
 sortutils, 184
 sortutils::indexx, 317
indexxitint8
 sortutils, 184
 sortutils::indexx, 317
indexxsingle
 sortutils, 185
 sortutils::indexx, 318
indexxstring
 sortutils, 185
 sortutils::indexx, 319
infer_variable_type
 csv_module, 46
info
 pahm_messages, 110
initadcircnetcdfoutfile
 pahm_netcdfio, 114
initialize
 csv_module::csv_file, 293
initialize_csv_file
 csv_module, 47
initialized
 pahm_netcdfio::adcirccoorddata_t, 246
 pahm_netcdfio::adcircvardata3d_t, 249
 pahm_netcdfio::adcircvardata_t, 252
 pahm_netcdfio::filedata_t, 299
 pahm_netcdfio::timedata_t, 338
initials
 parwind::asymmetricvortexdata_t, 268
 parwind::besttrackdata_t, 277
initlogging
 pahm_messages, 101
initval
 pahm_netcdfio::adcirccoorddata_t, 246
 pahm_netcdfio::adcircvardata3d_t, 249
 pahm_netcdfio::adcircvardata_t, 252
int1
 pahm_sizes, 127
int16

pahm_sizes, 127
int2
pahm_sizes, 128
int4
pahm_sizes, 128
int8
pahm_sizes, 128
interpr
pahm_vortex, 139
intlat
parwind::besttrackdata_t, 277
intlon
parwind::besttrackdata_t, 278
intmslp
parwind::besttrackdata_t, 278
intpouter
parwind::besttrackdata_t, 278
intrad1
parwind::besttrackdata_t, 278
intrad2
parwind::besttrackdata_t, 278
intrad3
parwind::besttrackdata_t, 278
intrad4
parwind::besttrackdata_t, 279
intrmw
parwind::besttrackdata_t, 279
introuter
parwind::besttrackdata_t, 279
intscan
utilities, 226
intspeed
parwind::besttrackdata_t, 279
intvalstr
utilities, 227
intvmax
parwind::besttrackdata_t, 279
ip
pahm_sizes, 128
iprp
parwind::asymmetricvortexdata_t, 268
ir
parwind::asymmetricvortexdata_t, 268
irmw
parwind::asymmetricvortexdata_t, 268
parwind::hollanddata_t, 312
irrp
parwind::asymmetricvortexdata_t, 268
parwind::hollanddata_t, 312
ismeshok
pahm_mesh, 97
isotachspercycle
parwind::asymmetricvortexdata_t, 269
ispeed
parwind::asymmetricvortexdata_t, 269
parwind::asymmetricvortexdata_t, 269
parwind::hollanddata_t, 312
istormspeed
parwind::asymmetricvortexdata_t, 269
iunit
csv_module::csv_file, 296
ivr
parwind::asymmetricvortexdata_t, 269
joindate
timedateutils, 202
juldaytogram
timedateutils, 203
kt2ms
pahm_global, 79
lat
parwind::asymmetricvortexdata_t, 269
parwind::besttrackdata_t, 279
parwind::hollanddata_t, 313
latestangle
pahm_vortex, 156
latestrmax
pahm_vortex, 156
leapyear
timedateutils, 205
llong
pahm_sizes, 128
loaded
parwind::asymmetricvortexdata_t, 269
parwind::besttrackdata_t, 280
parwind::hollanddata_t, 313
loadintvar
utilities, 228
loadlogvar
utilities, 229
loadrealvar
utilities, 229
filename
pahm_global, 80
logfileopened
pahm_messages, 110
logical_false_string
csv_module::csv_file, 296
logical_true_string
csv_module::csv_file, 296
loginitcalled
pahm_messages, 111
loglevelname
pahm_messages, 111
logmessage_1
pahm_messages, 102
pahm_messages::logmessage, 321
logmessage_2

pahm_messages, 103
pahm_messages::logmessage, 322

lon
 parwind::asymmetricvortexdata_t, 270
 parwind::besttrackdata_t, 280
 parwind::hollanddata_t, 313

long
 pahm_sizes, 128

lun_btrk
 pahm_global, 80

lun_btrk1
 pahm_global, 80

lun_ctrl
 pahm_global, 80

lun_inp
 pahm_global, 81

lun_inp1
 pahm_global, 81

lun_log
 pahm_global, 81

lun_out
 pahm_global, 81

lun_out1
 pahm_global, 81

lun_screen
 pahm_global, 82

m2nm
 pahm_global, 82

max_integer_str_len
 csv_parameters, 59

max_real_str_len
 csv_parameters, 59

max_size_for_insertion_sort
 csv_utilities, 63

maxfacenodes
 pahm_mesh, 97

maxseas
 parwind::asymmetricvortexdata_t, 270
 parwind::besttrackdata_t, 280

mb2kpa
 pahm_global, 82

mb2pa
 pahm_global, 82

mdbegsimtime
 pahm_global, 82

mdendsimtime
 pahm_global, 83

mdjdate
 timedateutils, 212

mdjoffset
 timedateutils, 213

mdjtime
 timedateutils, 213

mdoutdt
 pahm_global, 83

meshdimid
 pahm_netcdfio, 121

meshfileform
 pahm_global, 83

meshfilename
 pahm_global, 83

meshfilenamespecified
 pahm_global, 84

meshfiletype
 pahm_global, 84

meshvarid
 pahm_netcdfio, 121

messagesources
 pahm_messages, 111

method
 parwind, 176

modeldate
 timedateutils, 213

modeltime
 timedateutils, 213

modeltype
 pahm_global, 84

modjuldate
 timedateutils, 213

modjultime
 timedateutils, 214

month
 parwind::asymmetricvortexdata_t, 270
 parwind::besttrackdata_t, 280
 parwind::hollanddata_t, 313

monthdays
 timedateutils, 205

ms2kt
 pahm_global, 84

myfile
 pahm_netcdfio, 121

mytime
 pahm_netcdfio, 121

n_cols
 csv_module::csv_file, 296

n_rows
 csv_module::csv_file, 296

nbtrfiles
 pahm_global, 84

nbyte
 pahm_sizes, 129

nc3form
 pahm_netcdfio, 122

nc4form
 pahm_netcdfio, 122

ncdeflate

pahm_global, 85
ncdlevel
 pahm_global, 85
ncformat
 pahm_ncdfio, 122
ncshuffle
 pahm_global, 85
ncvarnam_pres
 pahm_global, 85
ncvarnam_wndx
 pahm_global, 86
ncvarnam_wndy
 pahm_global, 86
ncycles
 parwind::asymmetricvortexdata_t, 270
ne
 pahm_mesh, 97
NetCDFCheckErr
 netcdfio.F90, 389
netcdfio.F90
 NetCDFCheckErr, 389
netcdfterminate
 pahm_ncdfio, 115
newadcircnetcdfoutfile
 pahm_ncdfio, 116
newvortex
 pahm_vortex, 140
newvortexfull
 pahm_vortex, 141
next_row
 csv_module, 48
 csv_module::csv_file, 293
nfn
 pahm_mesh, 97
nm
 pahm_mesh, 98
nm2m
 pahm_global, 86
nodedimid
 pahm_ncdfio, 122
noutdt
 pahm_global, 86
np
 pahm_mesh, 98
npoints
 pahm_vortex, 157
nquads
 pahm_vortex, 157
ns
 parwind::asymmetricvortexdata_t, 270
 parwind::besttrackdata_t, 280
nscreen
 pahm_messages, 111
number_of_lines_in_file
 csv_module, 48
numbtfiles
 utilities, 244
numcycle
 parwind::asymmetricvortexdata_t, 270
numrec
 parwind::asymmetricvortexdata_t, 271
 parwind::besttrackdata_t, 280
 parwind::hollanddata_t, 313
offfirstgregday
 timedateutils, 214
offmodeljulday
 timedateutils, 214
offmodjulday
 timedateutils, 214
offunixjulday
 timedateutils, 214
omega
 pahm_global, 87
one2ten
 pahm_global, 87
open
 csv_module::csv_file, 294
open_csv_file
 csv_module, 49
openfileforread
 utilities, 230
openlogfile
 pahm_messages, 103
outdt
 pahm_global, 87
outfilename
 pahm_global, 87
outfilenamespecified
 pahm_global, 88
pahm
 pahm.F90, 401
pahm.F90
 pahm, 401
pahm_drivermod, 63
 cnttimebegin, 68
 cnttimeend, 68
 getprogramcmdlargs, 64
 pahm_finalize, 64
 pahm_init, 65
 pahm_run, 66
pahm_finalize
 pahm_drivermod, 64
pahm_global, 68
 airdensity, 71
 backgroundatmpress, 71
 baseee, 71
 begdate, 72

begdatespecified, 72
begdatetime, 72
begday, 72
beghour, 72
begmin, 73
begmonth, 73
begsec, 73
begsimspecified, 73
begsimtime, 73
begtime, 74
begyear, 74
besttrackfilename, 74
besttrackfilenamespecified, 74
controlfilename, 74
datestimes, 75
def_ncnam_pres, 75
def_ncnam_wndx, 75
def_ncnam_wndy, 75
defv_atmpress, 75
defv_gravity, 76
defv_rhoair, 76
defv_rhowater, 76
defv_windreduction, 76
deg2rad, 76
enddate, 77
enddatespecified, 77
enddatetime, 77
endday, 77
endhour, 78
endmin, 78
endmonth, 78
endsec, 78
endsimspecified, 78
endsimtime, 79
endtime, 79
endyear, 79
gravity, 79
kt2ms, 79
logfilename, 80
lun_btrk, 80
lun_btrk1, 80
lun_ctrl, 80
lun_inp, 81
lun_inp1, 81
lun_log, 81
lun_out, 81
lun_out1, 81
lun_screen, 82
m2nm, 82
mb2kpa, 82
mb2pa, 82
mdbegsimtime, 82
mdendsimtime, 83
mdoutdt, 83
meshfileform, 83
meshfilename, 83
meshfilenamespecified, 84
meshfiletype, 84
modeltype, 84
ms2kt, 84
nbtrfiles, 84
ncdeflate, 85
ncdlevel, 85
ncshuffle, 85
ncvarnam_pres, 85
ncvarnam_wndx, 86
ncvarnam_wndy, 86
nm2m, 86
noutdt, 86
omega, 87
one2ten, 87
outdt, 87
outfilename, 87
outfilenamespecified, 88
pi, 88
rad2deg, 88
rearth, 88
refdate, 88
refdatespecified, 89
refdatetime, 89
refday, 89
refhour, 89
refmin, 89
refmonth, 90
refsec, 90
reftime, 90
refyear, 90
rhoair, 91
rhowater, 91
ten2one, 91
times, 91
title, 91
unittime, 92
windreduction, 92
wpress, 92
writeparams, 92
wvelx, 93
wvely, 93
pahm_init
 pahm_drivermod, 65
pahm_mesh, 93
 agrid, 96
 allocatenodalandelementalarrays, 94
 dp, 96
 ics, 97
 ismeshok, 97
 maxfacenodes, 97
 ne, 97

nfn, 97
nm, 98
np, 98
readmesh, 94
readmeshasciifort14, 95
sfea, 98
sfea0, 98
slam, 99
slam0, 99
xslam, 99
ycsfea, 99
pahm_messages, 100
 allmessage_1, 101
 allmessage_2, 101
 closelogfile, 101
 debug, 109
 echo, 110
 error, 110
 info, 110
 initlogging, 101
 logfileopened, 110
 loginitcalled, 111
 loglevelname, 111
 logmessage_1, 102
 logmessage_2, 103
 messagesources, 111
 nscreen, 111
 openlogfile, 103
 programhelp, 103
 programversion, 104
 scratchformat, 112
 scratchmessage, 112
 screenmessage_1, 104
 screenmessage_2, 105
 setmessagesource, 105
 sourcenumber, 112
 terminate, 107
 unsetmessagesource, 108
 warning, 112
pahm_messages::allmessage, 253
 allmessage_1, 254
 allmessage_2, 254
pahm_messages::logmessage, 321
 logmessage_1, 321
 logmessage_2, 322
pahm_messages::screenmessage, 322
 screenmessage_1, 323
 screenmessage_2, 323
pahm_ncdfio, 113
 base_ncdfcheckerr, 114
 crdlats, 119
 crdlons, 119
 crdtime, 119
 crdxcs, 119
 crdycs, 120
 datatmpres, 120
 datelements, 120
 datwindx, 120
 datwindy, 120
 elemdimid, 121
 initadcircnetcdfoutfile, 114
 meshdimid, 121
 meshvarid, 121
 myfile, 121
 mytime, 121
 nc3form, 122
 nc4form, 122
 ncformat, 122
 netcdfterminate, 115
 newadcircnetcdfoutfile, 116
 nodedimid, 122
 projvarid, 122
 setrecordcounterandstoretime, 117
 vertdimid, 123
 writenetcdfrecord, 118
pahm_ncdfio::adcirccoorddata_t, 245
 count, 246
 dimid, 246
 initialized, 246
 initval, 246
 start, 247
 var, 247
 vardimids, 247
 vardims, 247
 varid, 247
 varname, 247
pahm_ncdfio::adcircvardata3d_t, 248
 count, 249
 initialized, 249
 initval, 249
 start, 249
 var, 249
 vardimids, 250
 vardims, 250
 varid, 250
 varname, 250
pahm_ncdfio::adcircvardata_t, 251
 count, 251
 initialized, 252
 initval, 252
 start, 252
 var, 252
 vardimids, 252
 vardims, 252
 varid, 253
 varname, 253
pahm_ncdfio::filedata_t, 298
 filefound, 299

filename, 299
filereccounter, 299
initialized, 299
pahm_ncdfio::timedata_t, 338
initialized, 338
time, 338
timedimid, 339
timedims, 339
timeid, 339
timelen, 339
pahm_run
 pahm_drivermod, 66
pahm_sizes, 123
 blank, 126
 comparedoublereals, 124
 comparesinglereals, 125
 fixnearwholedoublereal, 125
 fixnearwholesinglereal, 126
 fnamelen, 127
 hp, 127
 imissv, 127
 int1, 127
 int16, 127
 int2, 128
 int4, 128
 int8, 128
 ip, 128
 llong, 128
 long, 128
 nbyte, 129
 rmissv, 129
 sp, 129
 sz, 129
 wp, 129
pahm_sizes::comparereals, 283
 comparedoublereals, 283
 comparesinglereals, 284
pahm_sizes::fixnearwholereal, 300
 fixnearwholedoublereal, 300
 fixnearwholesinglereal, 301
pahm_vortex, 130
 b, 155
 bs, 155
 bs4, 155
 calcintensitychange, 132
 calcrmaxes, 133
 calcrmaxesfull, 133
 clat, 156
 clon, 156
 corio, 156
 fang, 134
 findroot, 135
 firmaxes, 136
 firmaxes4, 136
 getlatestangle, 136
 getlatestrmax, 137
 getphifactors, 137
 getrmaxes, 137
 getshapeparameter, 138
 getshapeparameters, 138
 getusequadrantvr, 138
 getvmaxesbl, 139
 interpr, 139
 latestangle, 156
 latestrmax, 156
 newvortex, 140
 newvortexfull, 141
 npoints, 157
 nquads, 157
 pc, 157
 phi, 157
 phis, 157
 phis4, 158
 pn, 158
 quad, 158
 quadflag4, 158
 quadir4, 158
 radius, 159
 rmaxes, 159
 rmaxes4, 159
 rmw, 141
 rotate, 142
 setisotachradii, 143
 setisotachwindspeed, 143
 setisotachwindspeeds, 143
 setrmaxes, 144
 setshapeparameter, 144
 setusequadrantvr, 144
 setusevmaxesbl, 145
 setvmaxesbl, 145
 setvortex, 146
 spinterp, 147
 usequadrantvr, 159
 usevmaxesbl, 159
 uvp, 148
 uvpr, 149
 uvtrans, 150
 uvtranspoint, 152
 vhncori, 152
 vhwithcori, 153
 vhwithcorifull, 154
 vmax, 160
 vmbl, 160
 vmbl4, 160
 vr, 160
 vrquadrant, 161
 parseline
 utilities, 231

partition
 csv_utilities.F90, 363
parwind, 161
 allocasymvortstruct, 162
 allocbtrstruct, 163
 allochollstruct, 163
 approach, 175
 asyvortstru, 175
 besttrackdata, 176
 deallocasymvortstruct, 164
 deallocbtrstruct, 164
 deallochollstruct, 166
 geofactor, 176
 geostrophicswitch, 176
 getgahmfields, 166
 gethollandfields, 168
 holSTRU, 176
 method, 176
 processasymmetricvortexdata, 169
 processhollanddata, 170
 readbesttrackfile, 172
 readcsvbesttrackfile, 173
 stormnamelen, 177
 writeasymmetricvortexdata, 174
 writebesttrackdata, 174
parwind::asymmetricvortexdata_t, 263
 basin, 264
 casttime, 265
 casttype, 265
 casttypenum, 265
 cypress, 265
 day, 265
 dir, 265
 dtg, 266
 ew, 266
 eye, 266
 fcstinc, 266
 filename, 266
 gusts, 266
 hollb, 267
 hollbs, 267
 hour, 267
 icpress, 267
 idir, 267
 ilat, 267
 ilon, 268
 initials, 268
 iprp, 268
 ir, 268
 irmw, 268
 irrp, 268
 isotachspercycle, 269
 ispeed, 269
 istormspeed, 269
 ivr, 269
 lat, 269
 loaded, 269
 lon, 270
 maxseas, 270
 month, 270
 ncycles, 270
 ns, 270
 numcycle, 270
 numrec, 271
 prp, 271
 quadflag, 271
 rmaxw, 271
 rmw, 271
 rrp, 271
 speed, 272
 stormname, 272
 stormnumber, 272
 stormspeed, 272
 subregion, 272
 thisstorm, 272
 trvx, 273
 trvy, 273
 ty, 273
 vmaxesbl, 273
 windcode, 273
 year, 273
parwind::besttrackdata_t, 274
 basin, 275
 cyclenum, 276
 cynum, 276
 day, 276
 dir, 276
 dtg, 276
 ew, 276
 eye, 277
 filename, 277
 gusts, 277
 hour, 277
 initials, 277
 intlat, 277
 intlon, 278
 intmslp, 278
 intpouter, 278
 intrad1, 278
 intrad2, 278
 intrad3, 278
 intrad4, 279
 intrmw, 279
 introuter, 279
 intspeed, 279
 intvmax, 279
 lat, 279
 loaded, 280

lon, 280
maxseas, 280
month, 280
ns, 280
numrec, 280
rad, 281
stormname, 281
subregion, 281
tau, 281
tech, 281
technum, 281
thisstorm, 282
ty, 282
windcode, 282
year, 282
parwind::hollanddata_t, 309
basin, 310
casttime, 310
casttype, 310
cprdt, 310
cpress, 311
day, 311
dtg, 311
fcstinc, 311
filename, 311
hour, 311
icpress, 312
ilat, 312
ilon, 312
irmw, 312
irrp, 312
ispeed, 312
lat, 313
loaded, 313
lon, 313
month, 313
numrec, 313
rmw, 313
rrp, 314
speed, 314
stormnumber, 314
thisstorm, 314
trvx, 314
trvy, 314
year, 315
pc
 pahm_vortex, 157
phi
 pahm_vortex, 157
phis
 pahm_vortex, 157
phis4
 pahm_vortex, 158
pi
 pahm_global, 88
pn
 pahm_vortex, 158
preprocessdatetimestring
 timedateutils, 206
printmodelparams
 utilities, 232
processasymmetricvortexdata
 parwind, 169
processhollanddata
 parwind, 170
programhelp
 pahm_messages, 103
programversion
 pahm_messages, 104
projvarid
 pahm_netcdfio, 122
prp
 parwind::asymmetricvortexdata_t, 271
quad
 pahm_vortex, 158
quadflag
 parwind::asymmetricvortexdata_t, 271
quadflag4
 pahm_vortex, 158
quadir4
 pahm_vortex, 158
quicksort
 csv_utilities.F90, 363
 sortutils, 186
quote
 csv_module::csv_file, 297
rad
 parwind::besttrackdata_t, 281
rad2deg
 pahm_global, 88
radius
 pahm_vortex, 159
read
 csv_module::csv_file, 294
read_csv_file
 csv_module, 50
read_line_from_file
 csv_module, 51
 csv_module::csv_file, 294
readbesttrackfile
 parwind, 172
readcontrolfile
 utilities, 233
readcsvbesttrackfile
 parwind, 173
readmesh
 pahm_mesh, 94

readmeshasciifort14
 pahm_mesh, 95
realscan
 utilities, 235
rearth
 pahm_global, 88
refdate
 pahm_global, 88
refdatespecified
 pahm_global, 89
refdatetime
 pahm_global, 89
refday
 pahm_global, 89
refhour
 pahm_global, 89
refmin
 pahm_global, 89
refmonth
 pahm_global, 90
refsec
 pahm_global, 90
reftime
 pahm_global, 90
refyear
 pahm_global, 90
rhoair
 pahm_global, 91
rhowater
 pahm_global, 91
rmaxes
 pahm_vortex, 159
rmaxes4
 pahm_vortex, 159
rmaxw
 parwind::asymmetricvortexdata_t, 271
rmissv
 pahm_sizes, 129
rmw
 pahm_vortex, 141
 parwind::asymmetricvortexdata_t, 271
 parwind::hollanddata_t, 313
rotate
 pahm_vortex, 142
rrp
 parwind::asymmetricvortexdata_t, 271
 parwind::hollanddata_t, 314
scratchformat
 pahm_messages, 112
scratchmessage
 pahm_messages, 112
screenmessage_1
 pahm_messages, 104
pahm_messages::screenmessage, 323
screenmessage_2
 pahm_messages, 105
 pahm_messages::screenmessage, 323
setisotachradii
 pahm_vortex, 143
setisotachwindspeed
 pahm_vortex, 143
setisotachwindspeeds
 pahm_vortex, 143
setmessagesource
 pahm_messages, 105
setrecordcounterandstoretime
 pahm_netcdfio, 117
setrmaxes
 pahm_vortex, 144
setshapeparameter
 pahm_vortex, 144
setusequadrantvr
 pahm_vortex, 144
setusevmaxesbl
 pahm_vortex, 145
setvmaxesbl
 pahm_vortex, 145
setvortex
 pahm_vortex, 146
sfea
 pahm_mesh, 98
sfea0
 pahm_mesh, 98
slam
 pahm_mesh, 99
slam0
 pahm_mesh, 99
sort2
 sortutils, 187
sortAscending
 csv_utilities, 60
sortutils, 177
 arraycopydouble, 178
 arraycopyint, 179
 arraycopysingle, 179
 arrayequaldouble, 180
 arrayequalint, 180
 arrayequalsingle, 181
 arthdouble, 181
 arthint, 182
 arthsingle, 183
 indexxdouble, 183
 indexxit, 184
 indexxit8, 184
 indexxsingle, 185
 indexxstring, 185
 quicksort, 186

sort2, 187
stringlexcomp, 188
swapdouble, 189
swapdoublevec, 190
swapint, 190
swapintvec, 191
swapsingle, 191
swapsinglevec, 192
sortutils.F90
 icompxchg, 453
sortutils::arraycopy, 255
 arraycopydouble, 255
 arraycopyint, 256
 arraycopsingle, 256
sortutils::arrayequal, 257
 arrayequaldouble, 258
 arrayequalint, 258
 arrayequalsingle, 259
sortutils::arth, 260
 arthdouble, 260
 arthint, 261
 arthsingle, 261
sortutils::indexx, 315
 indexxdouble, 316
 indexxit, 317
 indexxit8, 317
 indexxsingle, 318
 indexxstring, 319
sortutils::swap, 330
 swapdouble, 331
 swapdoublevec, 331
 swapint, 332
 swapintvec, 332
 swapsingle, 333
 swapsinglevec, 334
sourcenumber
 pahm_messages, 112
sp
 pahm_sizes, 129
speed
 parwind::asymmetricvortexdata_t, 272
 parwind::hollanddata_t, 314
sphericaldistance_1d
 utilities, 237
 utilities::sphericaldistance, 324
sphericaldistance_2d
 utilities, 238
 utilities::sphericaldistance, 325
sphericaldistance_scalar
 utilities, 239
 utilities::sphericaldistance, 326
sphericaldistanceharv
 utilities, 239
sphericalfracpoint

 utilities, 240
spinterp
 pahm_vortex, 147
split
 csv_module, 52
splitdate
 timedateutils, 207
splittatetimestring
 timedateutils, 208
 timedateutils::splittatetimestring, 328
splittatetimestring2
 timedateutils, 209
 timedateutils::splittatetimestring, 329
start
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 249
 pahm_netcdfio::adcircvardata_t, 252
stormname
 parwind::asymmetricvortexdata_t, 272
 parwind::besttrackdata_t, 281
stormnamelen
 parwind, 177
stormnumber
 parwind::asymmetricvortexdata_t, 272
 parwind::hollanddata_t, 314
stormspeed
 parwind::asymmetricvortexdata_t, 272
str
 csv_module::csv_string, 298
stringlexcomp
 sortutils, 188
subregion
 parwind::asymmetricvortexdata_t, 272
 parwind::besttrackdata_t, 281
swap
 csv_utilities, 61
swapdouble
 sortutils, 189
 sortutils::swap, 331
swapdoublevec
 sortutils, 190
 sortutils::swap, 331
swapint
 sortutils, 190
 sortutils::swap, 332
swapintvec
 sortutils, 191
 sortutils::swap, 332
swapsingle
 sortutils, 191
 sortutils::swap, 333
swapsinglevec
 sortutils, 192
 sortutils::swap, 334

sz
 pahm_sizes, 129

tau
 parwind::besttrackdata_t, 281

tech
 parwind::besttrackdata_t, 281

technum
 parwind::besttrackdata_t, 281

tenZone
 pahm_global, 91

terminate
 pahm_messages, 107

thisstorm
 parwind::asymmetricvortexdata_t, 272
 parwind::besttrackdata_t, 282
 parwind::hollanddata_t, 314

time
 pahm_ncdfio::timedata_t, 338

timeconviseC
 timedateutils, 209
 timedateutils::timeconv, 335

timeconvrsec
 timedateutils, 210
 timedateutils::timeconv, 336

timedateutils, 193
 datetime2string, 194
 dayofyear, 196
 dayofyear2greg, 197
 elapsedsecs, 197
 firstgregdate, 212
 firstgregtime, 212
 gettimeconvsec, 198
 gregtojulday2, 199
 gregtojuldayisec, 200
 gregtojuldaysec, 201
 joindate, 202
 juldaytogreg, 203
 leapyear, 205
 mdjdate, 212
 mdjoffset, 213
 mdjtime, 213
 modeldate, 213
 modeltime, 213
 modjuldate, 213
 modjultime, 214
 monthdays, 205
 offfirstgregday, 214
 offmodeljulday, 214
 offmodjulday, 214
 offunixjulday, 214
 preprocessdatetimestring, 206
 splitdate, 207
 splittdatetimestring, 208

 splittdatetimestring2, 209
 timeconviseC, 209
 timeconvrsec, 210
 unixdate, 215
 unixtime, 215
 upp, 211
 usemodjulday, 215
 yeardays, 211

timedateutils::gregtojulday, 304
 gregtojulday2, 304
 gregtojuldayisec, 305
 gregtojuldaysec, 307

timedateutils::splittdatetimestring, 327
 splittdatetimestring, 328
 splittdatetimestring2, 329

timedateutils::timeconv, 335
 timeconviseC, 335
 timeconvrsec, 336

timedimid
 pahm_ncdfio::timedata_t, 339

timedims
 pahm_ncdfio::timedata_t, 339

timeid
 pahm_ncdfio::timedata_t, 339

timelen
 pahm_ncdfio::timedata_t, 339

times
 pahm_global, 91

title
 pahm_global, 91

to_integer
 csv_module, 53

to_logical
 csv_module, 53

to_real
 csv_module, 54

tokenize
 csv_module::csv_file, 294

tokenize_csv_line
 csv_module, 55

tolowercase
 utilities, 241

touppercase
 utilities, 242

trvx
 parwind::asymmetricvortexdata_t, 273
 parwind::hollanddata_t, 314

trvy
 parwind::asymmetricvortexdata_t, 273
 parwind::hollanddata_t, 314

ty
 parwind::asymmetricvortexdata_t, 273
 parwind::besttrackdata_t, 282

unique
 csv_utilities, 62
unittime
 pahm_global, 92
unixdate
 timedateutils, 215
unixtime
 timedateutils, 215
unsetmessagesource
 pahm_messages, 108
upp
 timedateutils, 211
usemodjulday
 timedateutils, 215
usequadrantvr
 pahm_vortex, 159
user-guide/abstract.md, 341
user-guide/application.md, 341
user-guide/code.md, 341
user-guide/credits.md, 341
user-guide/deliverables.md, 341
user-guide/evaluation.md, 341
user-guide/features.md, 341
user-guide/figures.md, 341
user-guide/glossary.md, 341
user-guide/intro.md, 341
user-guide/models.md, 341
user-guide/pahm_manual.md, 341
user-guide/references-dox.md, 341
user-guide/tables.md, 341
usevmaxesbl
 pahm_vortex, 159
utilities, 215
 charunique, 217
 checkcontrolfileinputs, 218
 closetol, 244
 convlon, 219
 cpptogeo_1d, 219
 cpptogeo_scalar, 220
 drealscan, 221
 dvalstr, 222
 geotcpp_1d, 223
 geotcpp_scalar, 224
 getline record, 224
 getlocandratio, 225
 intscan, 226
 intvalstr, 227
 loadintvar, 228
 loadlogvar, 229
 loadrealvar, 229
 numbtfiles, 244
 openfileforread, 230
 parseline, 231
 printmodelparams, 232
 readcontrolfile, 233
 realscan, 235
 sphericaldistance_1d, 237
 sphericaldistance_2d, 238
 sphericaldistance_scalar, 239
 sphericaldistanceharv, 239
 sphericalfracpoint, 240
 tolowercase, 241
 touppercase, 242
 valstr, 243
 utilities::cpptogeo, 285
 cpptogeo_1d, 285
 cpptogeo_scalar, 286
 utilities::geotcpp, 302
 geotcpp_1d, 302
 geotcpp_scalar, 303
 utilities::sphericaldistance, 324
 sphericaldistance_1d, 324
 sphericaldistance_2d, 325
 sphericaldistance_scalar, 326
uvp
 pahm_vortex, 148
uvpr
 pahm_vortex, 149
uvtrans
 pahm_vortex, 150
uvtranspoint
 pahm_vortex, 152
valstr
 utilities, 243
var
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 249
 pahm_netcdfio::adcircvardata_t, 252
vardimids
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 250
 pahm_netcdfio::adcircvardata_t, 252
vardims
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 250
 pahm_netcdfio::adcircvardata_t, 252
variable_types
 csv_module, 56
 csv_module::csv_file, 294
varid
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 250
 pahm_netcdfio::adcircvardata_t, 253
varname
 pahm_netcdfio::adcirccoorddata_t, 247
 pahm_netcdfio::adcircvardata3d_t, 250
 pahm_netcdfio::adcircvardata_t, 253

vertdimid
 pahm_netcdfio, 123

vhnocori
 pahm_vortex, 152

vhwithcori
 pahm_vortex, 153

vhwithcorifull
 pahm_vortex, 154

vmax
 pahm_vortex, 160

vmaxesbl
 parwind::asymmetricvortexdata_t, 273

vmb1
 pahm_vortex, 160

vmb14
 pahm_vortex, 160

vr
 pahm_vortex, 160

vrquadrant
 pahm_vortex, 161

warning
 pahm_messages, 112

windcode
 parwind::asymmetricvortexdata_t, 273
 parwind::besttrackdata_t, 282

windreduction
 pahm_global, 92

wp
 pahm_sizes, 129

wpress
 pahm_global, 92

writeasymmetricvortexdata
 parwind, 174

writebesttrackdata
 parwind, 174

writenetcdffrecord
 pahm_netcdfio, 118

writeparams
 pahm_global, 92

wvelx
 pahm_global, 93

wvely
 pahm_global, 93

xcslam
 pahm_mesh, 99

ycsfea
 pahm_mesh, 99

year
 parwind::asymmetricvortexdata_t, 273
 parwind::besttrackdata_t, 282
 parwind::hollanddata_t, 315

yeardays