

# Dikes Overtopping Kernel - Technical documentation

Generated by Doxygen 1.8.9.1

Tue Sep 29 2015 10:07:35



# Contents

<b>1</b>	<b>Todo List</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Libraries . . . . .	3
<b>3</b>	<b>Modules Index</b>	<b>5</b>
3.1	Modules List . . . . .	5
<b>4</b>	<b>Data Type Index</b>	<b>7</b>
4.1	Class List . . . . .	7
<b>5</b>	<b>File Index</b>	<b>9</b>
5.1	File List . . . . .	9
<b>6</b>	<b>Module Documentation</b>	<b>11</b>
6.1	Feedback library . . . . .	11
6.1.1	Detailed Description . . . . .	13
6.1.2	Function/Subroutine Documentation . . . . .	14
6.1.2.1	allow_log . . . . .	14
6.1.2.2	errormessagedouble . . . . .	15
6.1.2.3	errormessageint . . . . .	15
6.1.2.4	errormessagereal . . . . .	15
6.1.2.5	errormessageetxt . . . . .	15
6.1.2.6	fatalerrordouble . . . . .	16
6.1.2.7	fatalerrorint . . . . .	16
6.1.2.8	fatalerroroccured . . . . .	16
6.1.2.9	fatalerrorreal . . . . .	16
6.1.2.10	fatalerrortxt . . . . .	17
6.1.2.11	feedbackclose . . . . .	17
6.1.2.12	feedbackcontextdouble . . . . .	18
6.1.2.13	feedbackcontextint . . . . .	18
6.1.2.14	feedbackcontextreal . . . . .	18
6.1.2.15	feedbackcontexttxt . . . . .	18

6.1.2.16	<code>feedbackdropcontext</code>	19
6.1.2.17	<code>feedbackenter</code>	19
6.1.2.18	<code>feedbackerror</code>	19
6.1.2.19	<code>feedbackinitialise</code>	20
6.1.2.20	<code>feedbackleave</code>	21
6.1.2.21	<code>feedbackprintcontext</code>	21
6.1.2.22	<code>feedbackprintstack</code>	21
6.1.2.23	<code>getfatalerrorexpected</code>	22
6.1.2.24	<code>getfatalerrormessage</code>	23
6.1.2.25	<code>getthreadid</code>	24
6.1.2.26	<code>p</code>	24
6.1.2.27	<code>progressreportarray</code>	25
6.1.2.28	<code>progressreportdouble</code>	26
6.1.2.29	<code>progressreportint</code>	26
6.1.2.30	<code>progressreportreal</code>	26
6.1.2.31	<code>progressreporttxt</code>	26
6.1.2.32	<code>progressreportvalueandarray</code>	27
6.1.2.33	<code>resetfatalerroroccurred</code>	27
6.1.2.34	<code>setallowlogging</code>	28
6.1.2.35	<code>setfatalerroraction</code>	28
6.1.2.36	<code>setfatalerrorexpected</code>	29
6.1.2.37	<code>setfatalerroroccurred</code>	29
6.1.2.38	<code>throw</code>	29
6.1.2.39	<code>todll</code>	30
6.1.2.40	<code>warningmessagedouble</code>	31
6.1.2.41	<code>warningmessageint</code>	31
6.1.2.42	<code>warningmessagereal</code>	31
6.1.2.43	<code>warningmessagetxt</code>	31
6.1.3	<b>Variable Documentation</b>	32
6.1.3.1	<code>dll</code>	32
6.1.3.2	<code>onlyfile</code>	32
6.1.3.3	<code>onlyscreen</code>	32
6.1.3.4	<code>screenandfile</code>	32
6.1.3.5	<code>verbosebasic</code>	32
6.1.3.6	<code>verbosedebugging</code>	32
6.1.3.7	<code>verbosetailed</code>	33
6.1.3.8	<code>verbosenone</code>	33
<b>7</b>	<b>Module Documentation</b>	35
7.1	<b>angleutilities Module Reference</b>	35

---

7.1.1	Detailed Description	35
7.1.2	Function/Subroutine Documentation	35
7.1.2.1	anglebetween2directions	35
7.2	conversionfunctions Module Reference	36
7.2.1	Function/Subroutine Documentation	36
7.2.1.1	betafromq	36
7.2.1.2	freqfrombeta	37
7.2.1.3	logqfrombeta	38
7.2.1.4	pfrombeta	39
7.2.1.5	pqfrombeta	39
7.2.1.6	qfrombeta	40
7.2.1.7	returntimefrombeta	41
7.2.2	Variable Documentation	41
7.2.2.1	qmin	41
7.2.2.2	ulimit	41
7.2.2.3	upperlog	41
7.3	dlovertopping Module Reference	42
7.3.1	Detailed Description	42
7.3.2	Function/Subroutine Documentation	42
7.3.2.1	calculateqo	42
7.3.2.2	calculateqof	43
7.3.2.3	calcvalue	44
7.3.2.4	geometry_c_f	44
7.3.2.5	validateinputc	45
7.3.2.6	validateinputf	45
7.3.2.7	versionnumber	46
7.4	equalreals Module Reference	46
7.4.1	Detailed Description	47
7.4.2	Function/Subroutine Documentation	47
7.4.2.1	equalrealsabsolute	47
7.4.2.2	equalrealsinvector	47
7.4.2.3	equalrealsrelative	48
7.4.2.4	equaltableswithreals	48
7.4.2.5	equalvectorswithintegers	49
7.4.2.6	equalvectorswithreals	49
7.4.2.7	equalvectorswithrealsweightedmargin	50
7.5	factormodulertovertopping Module Reference	50
7.5.1	Function/Subroutine Documentation	50
7.5.1.1	calculategammab	50
7.5.1.2	calculategammabeta	51

---

7.5.1.3	calculategammaf . . . . .	51
7.5.1.4	calculatetanalpha . . . . .	52
7.6	feedback Module Reference . . . . .	53
7.6.1	Variable Documentation . . . . .	55
7.6.1.1	dp . . . . .	55
7.6.1.2	id . . . . .	55
7.6.1.3	sp . . . . .	55
7.6.1.4	wp . . . . .	55
7.7	feedback_parameters Module Reference . . . . .	56
7.7.1	Variable Documentation . . . . .	56
7.7.1.1	onfatalerrorstop . . . . .	56
7.7.1.2	onfatalerrorthrowexception . . . . .	56
7.7.1.3	onfatalerrorunittest . . . . .	56
7.8	feedbackdll Module Reference . . . . .	56
7.8.1	Function/Subroutine Documentation . . . . .	58
7.8.1.1	closefile . . . . .	58
7.8.1.2	contexttxt . . . . .	58
7.8.1.3	dropcontext . . . . .	58
7.8.1.4	getallowlogging . . . . .	58
7.8.1.5	getfatalerrormsg . . . . .	58
7.8.1.6	getfeedbackerror . . . . .	59
7.8.1.7	getonfatalerroraction . . . . .	59
7.8.1.8	getsetfatalerrorexpected . . . . .	59
7.8.1.9	getsetfatalerroroccurred . . . . .	59
7.8.1.10	getstacklevel . . . . .	59
7.8.1.11	getunitnumber . . . . .	60
7.8.1.12	getverbosity . . . . .	61
7.8.1.13	getverbositlevel . . . . .	61
7.8.1.14	initialise . . . . .	61
7.8.1.15	outputtodll . . . . .	61
7.8.1.16	poproutine . . . . .	61
7.8.1.17	printcontext . . . . .	62
7.8.1.18	printmessage . . . . .	62
7.8.1.19	printstack . . . . .	62
7.8.1.20	pushroutine . . . . .	63
7.8.1.21	setallowlogging . . . . .	63
7.8.1.22	setfatalerrormessage . . . . .	63
7.8.1.23	setonfatalerroraction . . . . .	63
7.8.1.24	writetodll . . . . .	63
7.8.2	Variable Documentation . . . . .	64

7.8.2.1	allow_logging . . . . .	64
7.8.2.2	fatalerroraction . . . . .	64
7.8.2.3	fatalerrormessage . . . . .	64
7.8.2.4	fbdata . . . . .	64
7.8.2.5	id . . . . .	64
7.8.2.6	isfatalerrorexpected . . . . .	64
7.8.2.7	isfatalerroroccurred . . . . .	64
7.9	fileutilities Module Reference . . . . .	65
7.9.1	Function/Subroutine Documentation . . . . .	65
7.9.1.1	readtable . . . . .	65
7.9.1.2	removefile . . . . .	65
7.9.1.3	writetablevalues . . . . .	65
7.10	formulamodulertoovertopping Module Reference . . . . .	66
7.10.1	Function/Subroutine Documentation . . . . .	67
7.10.1.1	adjustinfluencefactors . . . . .	67
7.10.1.2	calculateanglewaveattack . . . . .	67
7.10.1.3	calculatebreakerlimit . . . . .	67
7.10.1.4	calculatebreakerparameter . . . . .	68
7.10.1.5	calculatewavelength . . . . .	68
7.10.1.6	calculatewaveovertoppingdischarge . . . . .	69
7.10.1.7	calculatewaverunup . . . . .	70
7.10.1.8	calculatwavesteeepness . . . . .	71
7.10.1.9	cubicroots . . . . .	71
7.10.1.10	isequalreal . . . . .	72
7.10.1.11	isequalzero . . . . .	72
7.10.1.12	realrootscubicfunction . . . . .	73
7.10.1.13	rootsdepressedcubic . . . . .	74
7.10.1.14	rootsgeneralcubic . . . . .	74
7.11	ftnunit Module Reference . . . . .	75
7.11.1	Function/Subroutine Documentation . . . . .	77
7.11.1.1	assert_comparable_double . . . . .	77
7.11.1.2	assert_comparable_double1d . . . . .	77
7.11.1.3	assert_comparable_double2d . . . . .	78
7.11.1.4	assert_comparable_real . . . . .	78
7.11.1.5	assert_comparable_real1d . . . . .	78
7.11.1.6	assert_comparable_real2d . . . . .	78
7.11.1.7	assert_equal_int . . . . .	79
7.11.1.8	assert_equal_int1d . . . . .	79
7.11.1.9	assert_equal_logical . . . . .	79
7.11.1.10	assert_equal_logical1d . . . . .	80

7.11.1.11 assert_equal_string . . . . .	80
7.11.1.12 assert_equal_string1d . . . . .	80
7.11.1.13 assert_false . . . . .	81
7.11.1.14 assert_files_comparable . . . . .	81
7.11.1.15 assert_inbetween_double . . . . .	81
7.11.1.16 assert_inbetween_real . . . . .	82
7.11.1.17 assert_true . . . . .	82
7.11.1.18 colour_number . . . . .	82
7.11.1.19 expect_program_stop . . . . .	83
7.11.1.20 ftnunit_file_exists . . . . .	83
7.11.1.21 ftnunit_make_empty_file . . . . .	83
7.11.1.22 ftnunit_remove_file . . . . .	84
7.11.1.23 ftnunit_write_html_close_row . . . . .	84
7.11.1.24 ftnunit_write_html_cpu . . . . .	86
7.11.1.25 ftnunit_write_html_failed_double . . . . .	86
7.11.1.26 ftnunit_write_html_failed_double1d . . . . .	87
7.11.1.27 ftnunit_write_html_failed_double2d . . . . .	87
7.11.1.28 ftnunit_write_html_failed_equivalent . . . . .	88
7.11.1.29 ftnunit_write_html_failed_equivalent1d . . . . .	88
7.11.1.30 ftnunit_write_html_failed_files . . . . .	89
7.11.1.31 ftnunit_write_html_failed_inbetween_double . . . . .	90
7.11.1.32 ftnunit_write_html_failed_inbetween_real . . . . .	90
7.11.1.33 ftnunit_write_html_failed_int . . . . .	91
7.11.1.34 ftnunit_write_html_failed_int1d . . . . .	91
7.11.1.35 ftnunit_write_html_failed_logic . . . . .	92
7.11.1.36 ftnunit_write_html_failed_real . . . . .	93
7.11.1.37 ftnunit_write_html_failed_real1d . . . . .	93
7.11.1.38 ftnunit_write_html_failed_real2d . . . . .	94
7.11.1.39 ftnunit_write_html_failed_string . . . . .	94
7.11.1.40 ftnunit_write_html_failed_string1d . . . . .	95
7.11.1.41 ftnunit_write_html_footer . . . . .	95
7.11.1.42 ftnunit_write_html_header . . . . .	96
7.11.1.43 ftnunit_write_html_ignored . . . . .	97
7.11.1.44 ftnunit_write_html_previous_failed . . . . .	97
7.11.1.45 ftnunit_write_html_previous_stopped . . . . .	98
7.11.1.46 ftnunit_write_html_test_begin . . . . .	99
7.11.1.47 issilent . . . . .	99
7.11.1.48 runtests . . . . .	99
7.11.1.49 runtests_final . . . . .	100
7.11.1.50 runtests_init . . . . .	101

7.11.1.51 runtests_init_priv . . . . .	101
7.11.1.52 setruntestlevel . . . . .	102
7.11.1.53 settesttitle . . . . .	102
7.11.1.54 networkingdir . . . . .	102
7.11.1.55 test . . . . .	102
7.11.1.56 testwithlevel . . . . .	103
7.11.2 Variable Documentation . . . . .	104
7.11.2.1 call_final . . . . .	104
7.11.2.2 dp . . . . .	104
7.11.2.3 failed_asserts . . . . .	104
7.11.2.4 has_run . . . . .	104
7.11.2.5 html_file . . . . .	104
7.11.2.6 ignore_previous_test . . . . .	104
7.11.2.7 ignore_reason . . . . .	104
7.11.2.8 ignore_reason_previous_test . . . . .	104
7.11.2.9 ignore_test . . . . .	104
7.11.2.10 last_test . . . . .	104
7.11.2.11 mode_all . . . . .	104
7.11.2.12 mode_list . . . . .	104
7.11.2.13 mode_single . . . . .	105
7.11.2.14nofails . . . . .	105
7.11.2.15nofails_prev . . . . .	105
7.11.2.16noruns . . . . .	105
7.11.2.17notests_failed . . . . .	105
7.11.2.18notests_ignored . . . . .	105
7.11.2.19notests_run . . . . .	105
7.11.2.20notests_work_in_progress . . . . .	105
7.11.2.21previous . . . . .	105
7.11.2.22run_test_level . . . . .	105
7.11.2.23single_test . . . . .	105
7.11.2.24subptrtestinit . . . . .	105
7.11.2.25subptrtestprograminit . . . . .	106
7.11.2.26test_mode . . . . .	106
7.11.2.27testname . . . . .	106
7.11.2.28testno . . . . .	106
7.11.2.29testtitle . . . . .	106
7.11.2.30work_in_progress . . . . .	106
7.12 ftunit_hooks Module Reference . . . . .	106
7.12.1 Function/Subroutine Documentation . . . . .	106
7.12.1.1 ftunit_hook_test_assertion_failed . . . . .	106

7.12.1.2 <code>ftnunit_hook_test_completed</code>	107
7.12.1.3 <code>ftnunit_hook_test_start</code>	107
7.12.1.4 <code>ftnunit_hook_test_stop</code>	108
7.13 general Module Reference	108
7.14 geometrymodulertoovertopping Module Reference	108
7.14.1 Function/Subroutine Documentation	109
7.14.1.1 <code>adjustnonhorizontalberms</code>	109
7.14.1.2 <code>allocatevectorsgeometry</code>	110
7.14.1.3 <code>calculatehorzdistance</code>	111
7.14.1.4 <code>calculatehorzlenghts</code>	111
7.14.1.5 <code>calculatesegmentsslopes</code>	112
7.14.1.6 <code>checkcrosssection</code>	112
7.14.1.7 <code>copygeometry</code>	113
7.14.1.8 <code>deallocategeometry</code>	114
7.14.1.9 <code>determinesegmenttypes</code>	114
7.14.1.10 <code>initializegeometry</code>	114
7.14.1.11 <code>isequalgeometry</code>	115
7.14.1.12 <code>mergesequentialberms</code>	116
7.14.1.13 <code>removeberms</code>	116
7.14.1.14 <code>removedikesegments</code>	117
7.14.1.15 <code>splitcrosssection</code>	118
7.14.1.16 <code>writecrosssection</code>	119
7.15 interpolationtriangular Module Reference	119
7.15.1 Detailed Description	119
7.15.2 Function/Subroutine Documentation	119
7.15.2.1 <code>datainterpolation</code>	119
7.15.2.2 <code>triangularinterpolation</code>	120
7.16 mainmodulertoovertopping Module Reference	121
7.16.1 Function/Subroutine Documentation	121
7.16.1.1 <code>calculateovertopping</code>	121
7.16.1.2 <code>calculateovertoppingnegativefreeboard</code>	122
7.16.1.3 <code>calculateovertoppingsection</code>	122
7.16.1.4 <code>calculatewaveovertopping</code>	123
7.16.1.5 <code>checkinputdata</code>	124
7.16.1.6 <code>checkmodelfactors</code>	125
7.16.1.7 <code>convertovertoppinginput</code>	125
7.16.1.8 <code>interpolateresultssections</code>	126
7.17 modulelogging Module Reference	126
7.17.1 Variable Documentation	126
7.17.1.1 <code>currentlogging</code>	126

---

7.17.1.2	maxfilenamelen gth	127
7.18	overtoppinginterface Module Reference	127
7.18.1	Variable Documentation	127
7.18.1.1	varmodelfactorcriticalovertopping	127
7.19	physics Module Reference	127
7.20	physics_breakwater Module Reference	127
7.20.1	Detailed Description	127
7.20.2	Function/Subroutine Documentation	127
7.20.2.1	calculatebreakwater	127
7.21	physics_breakwatertypesenumerations Module Reference	128
7.21.1	Detailed Description	128
7.21.2	Variable Documentation	128
7.21.2.1	breakwatercaisson	128
7.21.2.2	breakwaternotpresent	128
7.21.2.3	breakwaterrubblemound	129
7.21.2.4	breakwaterverticalwall	129
7.22	physics_bretschneider Module Reference	129
7.22.1	Detailed Description	129
7.22.2	Function/Subroutine Documentation	129
7.22.2.1	calculatewavebretschneider	129
7.23	physics_foreland Module Reference	129
7.23.1	Detailed Description	130
7.23.2	Function/Subroutine Documentation	130
7.23.2.1	physics_calculateforeland	130
7.23.3	Variable Documentation	131
7.23.3.1	message_length	131
7.24	physics_wavereduction Module Reference	131
7.24.1	Detailed Description	131
7.24.2	Function/Subroutine Documentation	131
7.24.2.1	calculatewavereduction	131
7.25	precision Module Reference	132
7.25.1	Detailed Description	132
7.25.2	Variable Documentation	132
7.25.2.1	almostinf	132
7.25.2.2	almostzero	132
7.25.2.3	betalimit	133
7.25.2.4	dp	133
7.25.2.5	emconstant	133
7.25.2.6	gravityconstant	133
7.25.2.7	longmax	133

7.25.2.8 pi . . . . .	133
7.25.2.9 rholimit . . . . .	133
7.25.2.10 shortmax . . . . .	133
7.25.2.11 sp . . . . .	133
7.25.2.12 wp . . . . .	133
7.26 typedefinitionsrtoovertopping Module Reference . . . . .	134
7.26.1 Variable Documentation . . . . .	135
7.26.1.1 berm_max . . . . .	135
7.26.1.2 berm_min . . . . .	135
7.26.1.3 fb_max . . . . .	135
7.26.1.4 fb_min . . . . .	135
7.26.1.5 fn_max . . . . .	135
7.26.1.6 fn_min . . . . .	136
7.26.1.7 foreshore_max . . . . .	136
7.26.1.8 foreshore_min . . . . .	136
7.26.1.9 frunup1_max . . . . .	136
7.26.1.10 frunup1_min . . . . .	136
7.26.1.11 frunup2_max . . . . .	136
7.26.1.12 frunup2_min . . . . .	136
7.26.1.13 frunup3_max . . . . .	136
7.26.1.14 frunup3_min . . . . .	136
7.26.1.15 fs_max . . . . .	137
7.26.1.16 fs_min . . . . .	137
7.26.1.17 margindiff . . . . .	137
7.26.1.18 margingrad . . . . .	137
7.26.1.19 mz2_max . . . . .	137
7.26.1.20 mz2_min . . . . .	137
7.26.1.21 rfactor_max . . . . .	137
7.26.1.22 rfactor_min . . . . .	137
7.26.1.23 slope_max . . . . .	137
7.26.1.24 slope_min . . . . .	138
7.26.1.25 xdiff_min . . . . .	138
7.26.1.26 z2_iter_max . . . . .	138
7.26.1.27 z2_margin . . . . .	138
7.27 utilities Module Reference . . . . .	138
7.27.1 Detailed Description . . . . .	138
7.27.2 Function/Subroutine Documentation . . . . .	138
7.27.2.1 getfreelunumber . . . . .	138
7.27.2.2 to_lower . . . . .	140
7.27.2.3 to_upper . . . . .	140

---

7.27.3	Variable Documentation	140
7.27.3.1	cap	140
7.27.3.2	low	140
7.28	vectorutilities Module Reference	140
7.28.1	Detailed Description	141
7.28.2	Function/Subroutine Documentation	141
7.28.2.1	interpolateline	141
7.28.2.2	intgratewithexponentcomponent	141
7.28.2.3	linearinterpolate	142
7.28.2.4	linearinterpolateangles	143
7.28.2.5	linearinterpolatecyclic	144
7.28.2.6	loglinearinterpolate	145
7.28.2.7	loglinearinterpolatecyclic	145
7.28.2.8	normalize	146
7.28.2.9	sortandlocate	146
7.28.2.10	ssort	147
7.28.2.11	stepsize	147
7.28.2.12	trapeziumrule	148
7.29	waveparametersutilities Module Reference	148
7.29.1	Detailed Description	149
7.29.2	Function/Subroutine Documentation	149
7.29.2.1	computewaveperiod	149
7.29.2.2	computewavesteeppness	149
7.30	waverunup Module Reference	149
7.30.1	Function/Subroutine Documentation	150
7.30.1.1	convergedwithresidu	150
7.30.1.2	determinestartingvalue	150
7.30.1.3	findsmallestresidu	150
7.30.1.4	iterationwaverunup	151
7.31	writeutilities Module Reference	151
7.31.1	Function/Subroutine Documentation	152
7.31.1.1	inttostr	152
7.31.1.2	writetodevice_line	152
7.31.1.3	writetodevice_text_real	152
7.31.1.4	writetodevicechars	153
7.32	zfunctionswtiovertopping Module Reference	153
7.32.1	Detailed Description	153
7.32.2	Function/Subroutine Documentation	154
7.32.2.1	adjustprofile	154
7.32.2.2	calculateqorto	154

7.32.2.3	profileinstructure . . . . .	155
7.32.2.4	zfunclogratios . . . . .	156
<b>8</b>	<b>Data Type Documentation</b>	<b>159</b>
8.1	ftnunit::assert_comparable Interface Reference . . . . .	159
8.1.1	Detailed Description . . . . .	159
8.1.2	Member Function/Subroutine Documentation . . . . .	160
8.1.2.1	assert_comparable_double . . . . .	160
8.1.2.2	assert_comparable_double1d . . . . .	160
8.1.2.3	assert_comparable_double2d . . . . .	160
8.1.2.4	assert_comparable_real . . . . .	160
8.1.2.5	assert_comparable_real1d . . . . .	160
8.1.2.6	assert_comparable_real2d . . . . .	160
8.2	ftnunit::assert_equal Interface Reference . . . . .	161
8.2.1	Detailed Description . . . . .	161
8.2.2	Member Function/Subroutine Documentation . . . . .	161
8.2.2.1	assert_equal_int . . . . .	161
8.2.2.2	assert_equal_int1d . . . . .	161
8.2.2.3	assert_equal_logical . . . . .	161
8.2.2.4	assert_equal_logical1d . . . . .	162
8.2.2.5	assert_equal_string . . . . .	162
8.2.2.6	assert_equal_string1d . . . . .	162
8.3	ftnunit::assert_inbetween Interface Reference . . . . .	162
8.3.1	Detailed Description . . . . .	162
8.3.2	Member Function/Subroutine Documentation . . . . .	162
8.3.2.1	assert_inbetween_double . . . . .	162
8.3.2.2	assert_inbetween_real . . . . .	163
8.4	feedback::errormessage Interface Reference . . . . .	163
8.4.1	Detailed Description . . . . .	163
8.4.2	Member Function/Subroutine Documentation . . . . .	163
8.4.2.1	errormessagedouble . . . . .	163
8.4.2.2	errormessageint . . . . .	164
8.4.2.3	errormessagereal . . . . .	164
8.4.2.4	errormessagetxt . . . . .	164
8.5	feedback::fatalerror Interface Reference . . . . .	165
8.5.1	Detailed Description . . . . .	165
8.5.2	Member Function/Subroutine Documentation . . . . .	165
8.5.2.1	fatalerrordouble . . . . .	165
8.5.2.2	fatalerrorint . . . . .	165
8.5.2.3	fatalerrorreal . . . . .	166

8.5.2.4	<code>fatalerrortxt</code>	166
8.6	<code>feedback::feedbackcontext</code> Interface Reference	166
8.6.1	Detailed Description	167
8.6.2	Member Function/Subroutine Documentation	167
8.6.2.1	<code>feedbackcontextdouble</code>	167
8.6.2.2	<code>feedbackcontextint</code>	167
8.6.2.3	<code>feedbackcontextreal</code>	167
8.6.2.4	<code>feedbackcontexttxt</code>	167
8.7	<code>feedbackdll::feedbackdata</code> Type Reference	168
8.7.1	Detailed Description	168
8.7.2	Member Data Documentation	169
8.7.2.1	<code>context</code>	169
8.7.2.2	<code>contextnumber</code>	169
8.7.2.3	<code>error</code>	169
8.7.2.4	<code>lun</code>	169
8.7.2.5	<code>output</code>	169
8.7.2.6	<code>routine</code>	169
8.7.2.7	<code>stacklevel</code>	169
8.7.2.8	<code>verbositylevel</code>	169
8.8	<code>overtoppinginterface::overtoppinggeometrytype</code> Type Reference	170
8.8.1	Detailed Description	170
8.8.2	Member Data Documentation	170
8.8.2.1	<code>normal</code>	170
8.8.2.2	<code>npoints</code>	170
8.8.2.3	<code>roughness</code>	171
8.8.2.4	<code>xcoords</code>	171
8.8.2.5	<code>ycoords</code>	171
8.9	<code>overtoppinginterface::overtoppinggeometrytypef</code> Type Reference	171
8.9.1	Detailed Description	172
8.9.2	Member Data Documentation	172
8.9.2.1	<code>normal</code>	172
8.9.2.2	<code>npoints</code>	172
8.9.2.3	<code>roughness</code>	172
8.9.2.4	<code>xcoords</code>	172
8.9.2.5	<code>ycoords</code>	172
8.10	<code>ftnunit::proc</code> Interface Reference	172
8.10.1	Detailed Description	172
8.10.2	Constructor & Destructor Documentation	173
8.10.2.1	<code>proc</code>	173
8.11	<code>feedback::progressreport</code> Interface Reference	173

8.11.1	Detailed Description	173
8.11.2	Member Function/Subroutine Documentation	174
8.11.2.1	progressreportarray	174
8.11.2.2	progressreportdouble	175
8.11.2.3	progressreportint	175
8.11.2.4	progressreportreal	175
8.11.2.5	progressreporttxt	175
8.11.2.6	progressreportvalueandarray	176
8.12	modulelogging::logging Type Reference	176
8.12.1	Detailed Description	177
8.12.2	Member Data Documentation	177
8.12.2.1	filename	177
8.12.2.2	verbosity	177
8.13	physics_bretschneider::tpbretschneider Type Reference	177
8.13.1	Detailed Description	178
8.13.2	Member Data Documentation	178
8.13.2.1	d	178
8.13.2.2	f	178
8.13.2.3	mgh	178
8.13.2.4	mgt	178
8.13.2.5	v	178
8.14	typedefinitionsrovertopping::tpgeometry Type Reference	179
8.14.1	Detailed Description	180
8.14.2	Member Data Documentation	180
8.14.2.1	nbermsegments	180
8.14.2.2	ncoordinates	180
8.14.2.3	psi	180
8.14.2.4	roughnessfactors	180
8.14.2.5	segmentslopes	180
8.14.2.6	segmenttypes	180
8.14.2.7	xcoorddiff	180
8.14.2.8	xcoordinates	181
8.14.2.9	ycoorddiff	181
8.14.2.10	ycoordinates	181
8.15	typedefinitionsrovertopping::tpload Type Reference	181
8.15.1	Detailed Description	182
8.15.2	Member Data Documentation	182
8.15.2.1	h	182
8.15.2.2	hm0	182
8.15.2.3	phi	182

---

8.15.2.4	tm_10	182
8.16	typedefinitionsrtodovertopping::tpovertopping Type Reference	182
8.16.1	Detailed Description	183
8.16.2	Member Data Documentation	183
8.16.2.1	qo	183
8.16.2.2	z2	183
8.17	typedefinitionsrtodovertopping::tpovertoppinginput Type Reference	184
8.17.1	Detailed Description	185
8.17.2	Member Data Documentation	185
8.17.2.1	computedovertopping	185
8.17.2.2	criticalovertopping	185
8.17.2.3	factordeterminationq_b_f_b	185
8.17.2.4	factordeterminationq_b_f_n	185
8.17.2.5	frunup1	185
8.17.2.6	frunup2	186
8.17.2.7	frunup3	186
8.17.2.8	fshallow	186
8.17.2.9	m_z2	186
8.17.2.10	reductionfactorforeshore	186
8.17.2.11	relaxationfactor	186
8.17.2.12	typerunup	186
8.18	overtoppinginterface::tpprofilecoordinate Type Reference	187
8.18.1	Detailed Description	187
8.18.2	Member Data Documentation	187
8.18.2.1	roughness	187
8.18.2.2	xcoordinate	188
8.18.2.3	zcoordinate	188
8.19	physics_wavereduction::tpwavereduction Type Reference	188
8.19.1	Detailed Description	189
8.19.2	Member Data Documentation	189
8.19.2.1	h	189
8.19.2.2	hs	189
8.19.2.3	ht	189
8.20	feedback::warningmessage Interface Reference	189
8.20.1	Detailed Description	190
8.20.2	Member Function/Subroutine Documentation	190
8.20.2.1	warningmessagedouble	190
8.20.2.2	warningmessageint	190
8.20.2.3	warningmessagereal	190
8.20.2.4	warningmessagetext	190

8.21 writeutilities::writetodevice Interface Reference . . . . .	191
8.21.1 Detailed Description . . . . .	191
8.21.2 Member Function/Subroutine Documentation . . . . .	191
8.21.2.1 writetodevice_line . . . . .	191
8.21.2.2 writetodevice_text_real . . . . .	191
<b>9 File Documentation</b> . . . . .	<b>193</b>
9.1 angleUtilities.f90 File Reference . . . . .	193
9.1.1 Detailed Description . . . . .	193
9.2 breakwater.f90 File Reference . . . . .	193
9.2.1 Detailed Description . . . . .	194
9.3 breakwaterEnumerations.f90 File Reference . . . . .	194
9.3.1 Detailed Description . . . . .	194
9.4 bretschneider.f90 File Reference . . . . .	194
9.4.1 Detailed Description . . . . .	195
9.5 conversionFunctions.f90 File Reference . . . . .	195
9.6 dllOvertopping.f90 File Reference . . . . .	195
9.6.1 Detailed Description . . . . .	196
9.7 equalReals.f90 File Reference . . . . .	196
9.7.1 Detailed Description . . . . .	197
9.8 factorModuleRTOovertopping.f90 File Reference . . . . .	197
9.8.1 Detailed Description . . . . .	197
9.9 feedback.f90 File Reference . . . . .	198
9.9.1 Detailed Description . . . . .	200
9.10 feedback_parameters.f90 File Reference . . . . .	200
9.10.1 Detailed Description . . . . .	201
9.11 feedbackDLL.f90 File Reference . . . . .	201
9.11.1 Detailed Description . . . . .	202
9.12 fileUtilities.f90 File Reference . . . . .	202
9.12.1 Detailed Description . . . . .	203
9.13 foreland.f90 File Reference . . . . .	203
9.14 formulaModuleRTOovertopping.f90 File Reference . . . . .	203
9.14.1 Detailed Description . . . . .	204
9.15 ftnunit.f90 File Reference . . . . .	204
9.15.1 Function/Subroutine Documentation . . . . .	206
9.15.1.1 write_header . . . . .	206
9.16 ftnunit_hooks.f90 File Reference . . . . .	207
9.17 ftnunit_hooks_teamcity.f90 File Reference . . . . .	207
9.18 general.f90 File Reference . . . . .	207
9.18.1 Detailed Description . . . . .	208

---

9.19 geometryModuleRTOvertopping.f90 File Reference . . . . .	208
9.19.1 Detailed Description . . . . .	209
9.20 interpolationTriangular.f90 File Reference . . . . .	209
9.20.1 Detailed Description . . . . .	209
9.21 mainModuleRTOvertopping.f90 File Reference . . . . .	209
9.21.1 Detailed Description . . . . .	210
9.22 ModuleLogging.f90 File Reference . . . . .	210
9.22.1 Detailed Description . . . . .	210
9.23 overtoppingInterface.f90 File Reference . . . . .	210
9.23.1 Detailed Description . . . . .	211
9.24 physics.f90 File Reference . . . . .	211
9.24.1 Detailed Description . . . . .	211
9.25 precision.f90 File Reference . . . . .	211
9.25.1 Detailed Description . . . . .	212
9.26 typeDefinitionsRTOvertopping.f90 File Reference . . . . .	212
9.26.1 Detailed Description . . . . .	214
9.27 utilities.f90 File Reference . . . . .	214
9.27.1 Detailed Description . . . . .	214
9.28 vectorUtilities.f90 File Reference . . . . .	214
9.28.1 Detailed Description . . . . .	215
9.29 waveParametersUtilities.f90 File Reference . . . . .	215
9.29.1 Detailed Description . . . . .	216
9.30 waveReduction.f90 File Reference . . . . .	216
9.30.1 Detailed Description . . . . .	216
9.31 waveRunup.f90 File Reference . . . . .	216
9.31.1 Detailed Description . . . . .	216
9.32 writeUtilities.f90 File Reference . . . . .	217
9.32.1 Detailed Description . . . . .	217
9.33 zFunctionsWTIOvertopping.f90 File Reference . . . . .	217
9.33.1 Detailed Description . . . . .	218
<b>Index</b>	<b>219</b>



# Chapter 1

## Todo List

### Subprogram **conversionfunctions::betafromq** (q, beta)

Is it necessary to extend the routine betaFromQ with a check on the probability of non-exceedance (q) and a error message? Q has to be a between 0 and 1.

This routine is extreemly often called! Therefore with this check the speed of the computation is decreased. There is another argument not to implement this check: Sometimes by combining the computed probabilities the combined probability is slightly more than 1 or slightly less than 0 (errors of rounding), then the computation will end eith this error message. Now this error of rounding is repaired in this routine betaFromQ. If a probability is outside the allowed area the probability is changed to the lower or upper bound. The disadvantage of this method is that programming errors are no longer visible: a probability of for intance two doesn't generate an error!

### Module **Feedback**

Add multilingual support



## **Chapter 2**

# **Module Index**

### **2.1 Libraries**

Here is a list of all modules:

Feedback library . . . . .	11
----------------------------	----



# Chapter 3

## Modules Index

### 3.1 Modules List

Here is a list of all modules with brief descriptions:

<a href="#">angleutilities</a>	Module with direction utility functions . . . . .	35
<a href="#">conversionfunctions</a>		36
<a href="#">dllovertopping</a>	Calculate one type of overtopping . . . . .	42
<a href="#">equalreals</a>	Module with functions to test quality of reals . . . . .	46
<a href="#">factormodulertoovertopping</a>		50
<a href="#">feedback</a>		53
<a href="#">feedback_parameters</a>		56
<a href="#">feedbackdll</a>		56
<a href="#">fileutilities</a>		65
<a href="#">formulamodulertoovertopping</a>		66
<a href="#">ftnunit</a>		75
<a href="#">ftnunit_hooks</a>		106
<a href="#">general</a>		108
<a href="#">geometrymodulertoovertopping</a>		108
<a href="#">interpolationtriangular</a>	Module with function for triangular interpolation . . . . .	119
<a href="#">mainmodulertoovertopping</a>		121
<a href="#">modulelogging</a>		126
<a href="#">overtoppinginterface</a>		127
<a href="#">physics</a>		127
<a href="#">physics_breakwater</a>	Module with routines for breakwater . . . . .	127
<a href="#">physics_breakwatertypesenumerations</a>	Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRingEnumerations.CS . . . . .	128
<a href="#">physics_bretschneider</a>	Module with routines for bretschneider . . . . .	129
<a href="#">physics_foreland</a>	Wrapper for foreland dll !! . . . . .	129
<a href="#">physics_wavereduction</a>	Module with routines for wave reduction . . . . .	131
<a href="#">precision</a>	Module with parameters for KINDs and mathematical constants . . . . .	132
<a href="#">typedefinitionsrovertopping</a>		134

<b>utilities</b>	
Module with general utility routines . . . . .	138
<b>vectorutilities</b>	
Module with vector utility functions . . . . .	140
<b>waveparametersutilities</b>	
Module to calculate the wave steepness or the wave period . . . . .	148
<b>waverunup</b>	
. . . . .	149
<b>writeutilities</b>	
. . . . .	151
<b>zfunctionswtiovertopping</b>	
Module for the Limit State Functions (Z-functions) for wave overtopping . . . . .	153

## Chapter 4

# Data Type Index

### 4.1 Class List

Here are the data types with brief descriptions:

ftnunit::assert_comparable . . . . .	159
ftnunit::assert_equal . . . . .	161
ftnunit::assert_inbetween . . . . .	162
feedback::errormessage . . . . .	163
feedback::fatalerror . . . . .	165
feedback::feedbackcontext . . . . .	166
feedbackdll::feedbackdata . . . . .	168
overtoppinginterface::overtoppinggeometrytype . . . . .	170
overtoppinginterface::overtoppinggeometrytypeef . . . . .	171
ftnunit::proc . . . . .	172
feedback::progressreport . . . . .	173
modulelogging::tlogging TLogging: structure for steering the logging . . . . .	176
physics_bretschneider::tpbretschneider . . . . .	177
typedefinitionsrovertopping::tpgeometry TpGeometry: structure with geometry data . . . . .	179
typedefinitionsrovertopping::tupload TpLoad: structure with load parameters . . . . .	181
typedefinitionsrovertopping::tpovertopping TpOvertopping: structure with overtopping results . . . . .	182
typedefinitionsrovertopping::tpovertoppinginput OvertoppingModelFactors: C-structure with model factors . . . . .	184
overtoppinginterface::tpprofilecoordinate . . . . .	187
physics_wavereduction::tpwavereduction . . . . .	188
feedback::warningmessage . . . . .	189
writeutilities::writetodevice . . . . .	191



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

angleUtilities.f90	Implement various utility functions dealing with directions . . . . .	193
breakwater.f90	Breakwater formulas . . . . .	193
breakwaterEnumerations.f90	Module for enumerating breakwater types . . . . .	194
bretschneider.f90	Compute wave parameters using Bretschneider formulas . . . . .	194
conversionFunctions.f90		195
dllOvertopping.f90	Main entry for the dll DikesOvertopping FUNCTIONS/SUBROUTINES exported from dll← Overtopping.dll: . . . . .	195
equalReals.f90	This file contains a module with functions to test quality of reals . . . . .	196
factorModuleRTOovertopping.f90	This file contains a module with functions for the slope angle and influence factors . . . . .	197
feedback.f90	Feedback library: error messaging and progress reporting . . . . .	198
feedback_parameters.f90	Feedback parameters: parameters needed bij feedback and feedback_dll . . . . .	200
feedbackDLL.f90	Feedback dll: error messaging and progress reporting . . . . .	201
fileUtilities.f90	Module for reading information from files . . . . .	202
foreland.f90		203
formulaModuleRTOovertopping.f90	This file contains a module with the core computations for Dikes Overtopping . . . . .	203
ftnunit.f90		204
ftnunit_hooks.f90		207
ftnunit_hooks_teamcity.f90		207
general.f90	Overall module for general library . . . . .	207
geometryModuleRTOovertopping.f90	This file contains a module with the core computations for Dikes Overtopping related to the geometry . . . . .	208
interpolationTriangular.f90	Module containing the function for triangular interpolation . . . . .	209

<a href="#">mainModuleRTOvertopping.f90</a>	
This file contains a module with the core computations for Dikes Overtopping . . . . .	209
<a href="#">ModuleLogging.f90</a>	
Module for steering the extra logging . . . . .	210
<a href="#">overtoppingInterface.f90</a>	
This file contains the parameters and types (structs) as part of the interface to and from dll← Overtopping . . . . .	210
<a href="#">physics.f90</a>	
Overall module for physics library . . . . .	211
<a href="#">precision.f90</a>	
Define KINDs for controlling the precision of the REAL variables and define parameters for math- ematical constants . . . . .	211
<a href="#">typeDefinitionsRTOvertopping.f90</a>	
This file contains a module with the type definitions for Dikes Overtopping . . . . .	212
<a href="#">utilities.f90</a>	
General utility routines . . . . .	214
<a href="#">vectorUtilities.f90</a>	
Implement various utility functions for vectors . . . . .	214
<a href="#">waveParametersUtilities.f90</a>	
Module waveParametersUtilities for wave parameter computations . . . . .	215
<a href="#">waveReduction.f90</a>	
Wave reduction routines . . . . .	216
<a href="#">waveRunup.f90</a>	
This file contains a module with the core computations for Dikes Overtopping . . . . .	216
<a href="#">writeUtilities.f90</a>	
General write utilities . . . . .	217
<a href="#">zFunctionsWTIOvertopping.f90</a>	
This file contains the limit state functions for wave overtopping within WTI . . . . .	217

# Chapter 6

## Module Documentation

### 6.1 Feedback library

The feedback library provides a mechanism for reporting errors and progress.

#### Functions/Subroutines

- integer function **feedback::p ()**  
*Returns the pointer to the feedback dll, avoiding many loadlibrary calls.*
- logical function, public **feedback::todll ()**  
*Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?*
- logical function, public **feedback::fatalerroroccurred (value)**  
*Has a fatal error occured? Mainly used in unit tests.*
- subroutine, public **feedback::resetfatalerroroccurred ()**  
*Reset fatal error occurrence Mainly used in unit tests.*
- subroutine **feedback::setfatalerroroccurred (value)**  
*Set fatal error occurrence Mainly used in unit tests.*
- logical function, public **feedback::getfatalerrorexpected (value)**  
*Get value of FatalErrorExpected Mainly used in unit tests.*
- subroutine, public **feedback::setfatalerrorexpected (value)**  
*Set the value of FatalErrorExpected Mainly used in unit tests.*
- logical function, public **feedback::allow\_log ()**  
*Is logging active?*
- subroutine, public **feedback::setallowlogging (value)**  
*Set logging on/off.*
- subroutine, public **feedback::getthreadid (threadId)**  
*Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.*
- subroutine, public **feedback::setfatalerroraction (action)**  
*Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program  
onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.*
- subroutine, public **feedback::getfatalerrormessage (message)**  
*Get the last of FatalErrorMessage.*
- subroutine, public **feedback::feedbackerror (error)**  
*Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.*
- subroutine, public **feedback::feedbackclose**

- subroutine, public **feedback::feedbackinitialise** (output, logfile, verbosity)
 

*Initialise the feedback subsystem.*
- subroutine **feedback::progressreporttxt** (level, msgtext)
 

*Write a single message to the output for the benefit of the user.*
- subroutine **feedback::progressreportint** (level, msgtext, value)
 

*Write a single message with an integer value.*
- subroutine **feedback::progressreportreal** (level, msgtext, value)
 

*Write a single message with a real value.*
- subroutine **feedback::progressreportdouble** (level, msgtext, value)
 

*Write a single message with a double precision real value.*
- subroutine **feedback::progressreportarray** (level, msgText, x, size)
 

*Print array values to report file.*
- subroutine **feedback::progressreportvalueandarray** (level, msgText, z, x, size)
 

*Print z-value and array values to report file.*
- subroutine, public **feedback::feedbackenter** (routine)
 

*Register a new routine level.*
- subroutine, public **feedback::feedbackleave**

*Drop a routine level from the stack trace.*
- subroutine, public **feedback::feedbackprintstack**

*Print the stack trace as known via feedbackEnter and feedbackLeave.*
- subroutine **feedback::errormessagetxt** (msgtext)
 

*Write an error message.*
- subroutine **feedback::errormessageint** (msgtext, value)
 

*Write an error message with an integer value.*
- subroutine **feedback::errormessagereal** (msgtext, value)
 

*Write an error message with a real value.*
- subroutine **feedback::errormessagedouble** (msgtext, value)
 

*Write an error message with a double precision real value.*
- subroutine **feedback::warningmessagetxt** (msgtext)
 

*Write a warning message.*
- subroutine **feedback::warningmessageint** (msgtext, value)
 

*Write a warning message with an integer value.*
- subroutine **feedback::warningmessagereal** (msgtext, value)
 

*Write a warning message with a real value.*
- subroutine **feedback::warningmessagedouble** (msgtext, value)
 

*Write a warning message with a double precision real value.*
- subroutine **feedback::fatalerrortxt** (msgtext)
 

*Write a fatal error message.*
- subroutine **feedback::fatalerrorint** (msgtext, value)
 

*Write a fatal error message with an integer value.*
- subroutine **feedback::fatalerrorreal** (msgtext, value)
 

*Write a fatal error message with a real value.*
- subroutine **feedback::fatalerrordouble** (msgtext, value)
 

*Write a fatal error message with a double precision real value.*
- subroutine, public **feedback::feedbackdropcontext**

*Drop the entire context.*
- subroutine, public **feedback::feedbackprintcontext**

*Register a context for error messages.*
- subroutine **feedback::feedbackcontexttxt** (text)
 

*Register a context for error messages.*

- subroutine `feedback::feedbackcontextint` (text, value)  
*Register a context for error messages - with integer value.*
- subroutine `feedback::feedbackcontextreal` (text, value)  
*Register a context for error messages - with real value.*
- subroutine `feedback::feedbackcontextdouble` (text, value)  
*Register a context for error messages - with double precision real value.*
- subroutine, public `feedback::throw` (msgText)  
*Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.*

## Variables

- integer, parameter `feedback_parameters::onlyscreen` = 0  
*Mode: write to screen only.*
- integer, parameter `feedback_parameters::onlyfile` = 1  
*Mode: write to file only.*
- integer, parameter `feedback_parameters::screenandfile` = 2  
*Mode: write to both screen and file.*
- integer, parameter `feedback_parameters::dll` = 3  
*Mode: write to dll.*
- integer, parameter `feedback_parameters::verbosenone` = -1  
*Reporting: NO messages.*
- integer, parameter `feedback_parameters::verbosebasic` = 0  
*Reporting: basic messages only.*
- integer, parameter `feedback_parameters::verbosetailed` = 1  
*Reporting: detailed messages (including traceback)*
- integer, parameter `feedback_parameters::verbosedebugging` = 2  
*Reporting: all messages.*

### 6.1.1 Detailed Description

The feedback library provides a mechanism for reporting errors and progress.

#### Short documentation of intended usage:

The library consists of the following types of routines:

- Managing the library itself
- Write progress messages (detailed or basic)
- Write error messages
- Define the context of the computation

**Managing the library:** The routine `feedbackInitialise()` initialises the library with:

- Whether to write to screen only or to a file or both
- Name of the log file
- Detailed output or not (parameters: `verboseNone` to `verboseDebugging`)

For performance reasons, you can set and examine the variable `traceback`. This variable indicates whether keeping track of the stack is required. Typical usage in tight loops:

```
use feedback
...
if ( traceback ) call feedbackenter( "section or routine" )
...
if ( traceback ) call feedbackleave
```

See below for more details

(Note: initialisation has to be done for each thread and each thread should have its own logfile to avoid interlacing the messages)

**Write progress messages:** Progress messages tell the user what the program is doing. For users who are interested in the details of the computation the level of detail can be set high, for others a minimum amount of information is printed.

Use the routine `progressReport` to print a short message indicating the stage in the computation.

To communicate with the user-interface, use the `progressBarPercent` or `progressBarIncrement` routines.

**Write error messages:** Error messages are written with the context in which the error occurred and if details are required with the known stack.

An error message is written with the subroutine `errorMessage`, a warning with `warningMessage` and a fatal error (which stops the program) with `fatalError`

To indicate where in the program the error occurs, you can use the subroutine `feedbackEnter` - this routine registers where in the program (source) you are, this may be a routine you enter but also a section within the computation that is of interest. If you leave the subroutine or section, use `feedbackLeave` to indicate this. You probably should not use it inside a tight loop, but here is a sketch of typical use:

```
subroutine examplecomputation( ... )
use feedback
...
if ( traceback ) call feedbackenter( "exampleComputation" )
... some computation ...

if ( traceback ) call feedbackenter( "Main do-loop" )
do i = 1,1000000
    ... some computation ...
enddo
if ( traceback ) call feedbackleave
if ( traceback ) call feedbackleave
```

**Note:** `feedbackEnter` and `feedbackLeave` must be used in matching pairs!

**Define the context of the computation:** A bare error message may not be enough to understand what went wrong. For the purpose of standardising the extra information that should accompany an error message, the library offers the routine `feedbackContext`. You can use it to define relevant text to be shown in conjunction with the error message. For instance: the name of the computation method that is being run or the case that you are dealing with.

To define a completely new context, use `feedbackDropContext`.

**Todo** Add multilingual support

## 6.1.2 Function/Subroutine Documentation

### 6.1.2.1 logical function, public feedback::allow\_log( )

Is logging active?

Definition at line 273 of file `feedback.f90`.

Here is the call graph for this function:



**6.1.2.2 subroutine feedback::errormessagedouble ( character(len=\*), intent(in) msgtext, real(kind=dp), intent(in) value ) [private]**

Write an error message with a double precision real value.

#### Parameters

in	msgtext	Error message to write
in	value	Integer value to add

Definition at line 661 of file feedback.f90.

**6.1.2.3 subroutine feedback::errormessageint ( character(len=\*), intent(in) msgtext, integer, intent(in) value ) [private]**

Write an error message with an integer value.

#### Parameters

in	msgtext	Error message to write
in	value	Integer value to add

Definition at line 631 of file feedback.f90.

**6.1.2.4 subroutine feedback::errormessagereal ( character(len=\*), intent(in) msgtext, real, intent(in) value ) [private]**

Write an error message with a real value.

#### Parameters

in	msgtext	Error message to write
in	value	Integer value to add

Definition at line 646 of file feedback.f90.

**6.1.2.5 subroutine feedback::errormessagetxt ( character(len=\*), intent(in) msgtext ) [private]**

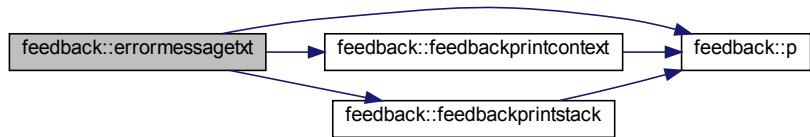
Write an error message.

#### Parameters

in	msgtext	Error message to write
----	---------	------------------------

Definition at line 608 of file feedback.f90.

Here is the call graph for this function:



**6.1.2.6 subroutine `feedback::fatalerrordouble`** ( `character(len=*) intent(in) msgtext, real(kind=dp), intent(in) value` )  
[private]

Write a fatal error message with a double precision real value.

#### Parameters

in	<code>msgtext</code>	Error message to write
in	<code>value</code>	Integer value to add

Definition at line 815 of file `feedback.f90`.

**6.1.2.7 subroutine `feedback::fatalerrorint`** ( `character(len=*) intent(in) msgtext, integer, intent(in) value` ) [private]

Write a fatal error message with an integer value.

#### Parameters

in	<code>msgtext</code>	Error message to write
in	<code>value</code>	Integer value to add

Definition at line 785 of file `feedback.f90`.

**6.1.2.8 logical function, public `feedback::fatalerroroccurred`** ( `result, value` )

Has a fatal error occured? Mainly used in unit tests.

Definition at line 212 of file `feedback.f90`.

Here is the call graph for this function:



**6.1.2.9 subroutine `feedback::fatalerrorreal`** ( `character(len=*) intent(in) msgtext, real, intent(in) value` ) [private]

Write a fatal error message with a real value.

## Parameters

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 800 of file feedback.f90.

6.1.2.10 subroutine `feedback::fatalerrortxt` ( character(len=\*)*msgtext* ) [private]

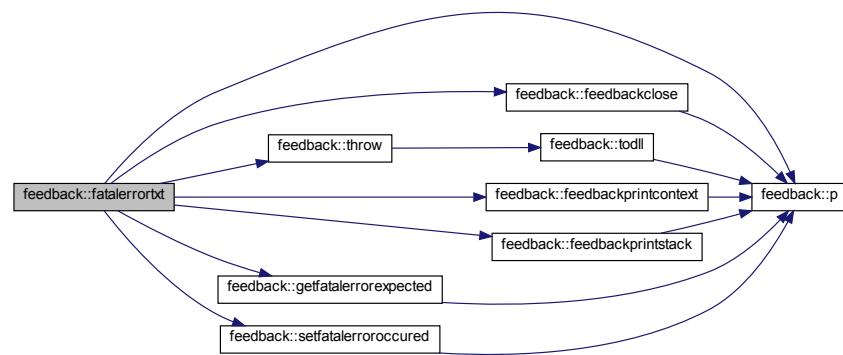
Write a fatal error message.

## Parameters

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 744 of file feedback.f90.

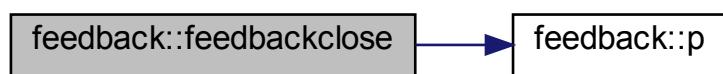
Here is the call graph for this function:

6.1.2.11 subroutine, public `feedback::feedbackclose` ( )

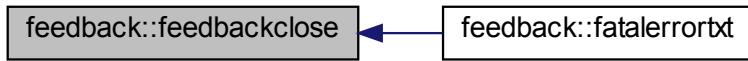
Close the log file.

Definition at line 355 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**6.1.2.12 subroutine `feedback::feedbackcontextdouble`** ( `character(len=*) intent(in) text, real(kind=dp), intent(in) value` )  
[private]

Register a context for error messages - with double precision real value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 900 of file `feedback.f90`.

**6.1.2.13 subroutine `feedback::feedbackcontextint`** ( `character(len=*) intent(in) text, integer, intent(in) value` ) [private]

Register a context for error messages - with integer value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 870 of file `feedback.f90`.

**6.1.2.14 subroutine `feedback::feedbackcontextreal`** ( `character(len=*) intent(in) text, real(kind=sp), intent(in) value` )  
[private]

Register a context for error messages - with real value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 885 of file `feedback.f90`.

**6.1.2.15 subroutine `feedback::feedbackcontexttxt`** ( `character(len=*) intent(in) text` ) [private]

Register a context for error messages.

#### Parameters

in	<i>text</i>	text to be added to the context stack
----	-------------	---------------------------------------

Definition at line 852 of file `feedback.f90`.

Here is the call graph for this function:



#### 6.1.2.16 subroutine, public feedback::feedbackdropcontext( )

Drop the entire context.

Definition at line 830 of file feedback.f90.

Here is the call graph for this function:



#### 6.1.2.17 subroutine, public feedback::feedbackenter( character(len=\*) routine )

Register a new routine level.

##### Parameters

<i>routine</i>	Name of the routine to add
----------------	----------------------------

Definition at line 563 of file feedback.f90.

Here is the call graph for this function:



#### 6.1.2.18 subroutine, public feedback::feedbackerror( logical error )

Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.

### Parameters

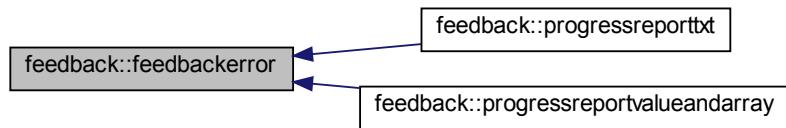
<i>error</i>	True if there has been some error reported
--------------	--

Definition at line 341 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**6.1.2.19 subroutine, public feedback::feedbackinitialise ( integer, intent(in) *output*, character(len=\*) , intent(in) *logfile*, integer, intent(in) *verbosity* )**

Initialise the feedback subsystem.

### Parameters

in	<i>output</i>	option: output to screen or file
in	<i>logfile</i>	name of the log file to be used
in	<i>verbosity</i>	level for displaying the messages

Definition at line 382 of file feedback.f90.

Here is the call graph for this function:



## 6.1.2.20 subroutine, public feedback::feedbackleave( )

Drop a routine level from the stack trace.

Definition at line 580 of file feedback.f90.

Here is the call graph for this function:



## 6.1.2.21 subroutine, public feedback::feedbackprintcontext( )

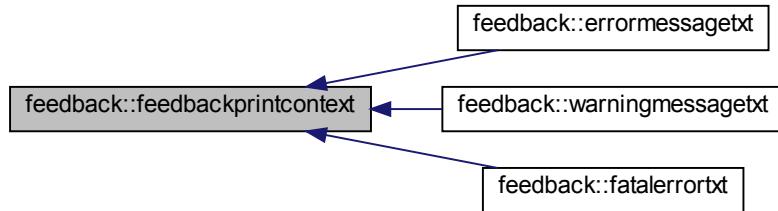
Register a context for error messages.

Definition at line 841 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.1.2.22 subroutine, public feedback::feedbackprintstack( )

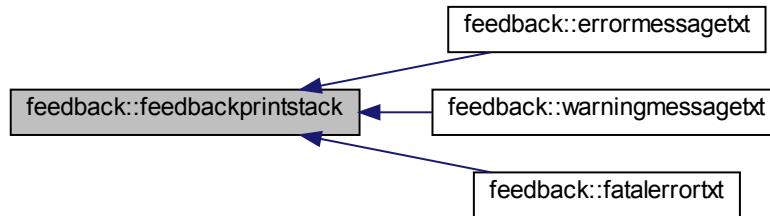
Print the stack trace as known via feedbackEnter and feedbackLeave.

Definition at line 597 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.1.2.23 logical function, public `feedback::getfatalerrorexpected ( result, value )`

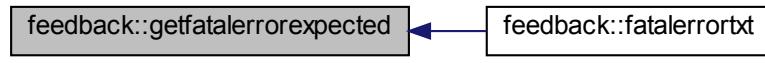
Get value of FatalErrorExpected Mainly used in unit tests.

Definition at line 249 of file `feedback.f90`.

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.2.24 subroutine, public feedback::getfatalerrormessage ( character(len=\*)*message* )

Get the last of FatalErrorMessage.

**Parameters**

<code>out</code>	<code>message</code>	the last error message
------------------	----------------------	------------------------

Definition at line 326 of file feedback.f90.

Here is the call graph for this function:

**6.1.2.25 subroutine, public `feedback::getthreadid` ( `integer, intent(out) threadid` )**

Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.

**Parameters**

<code>out</code>	<code>threadid</code>	the current thread ID (always 0)
------------------	-----------------------	----------------------------------

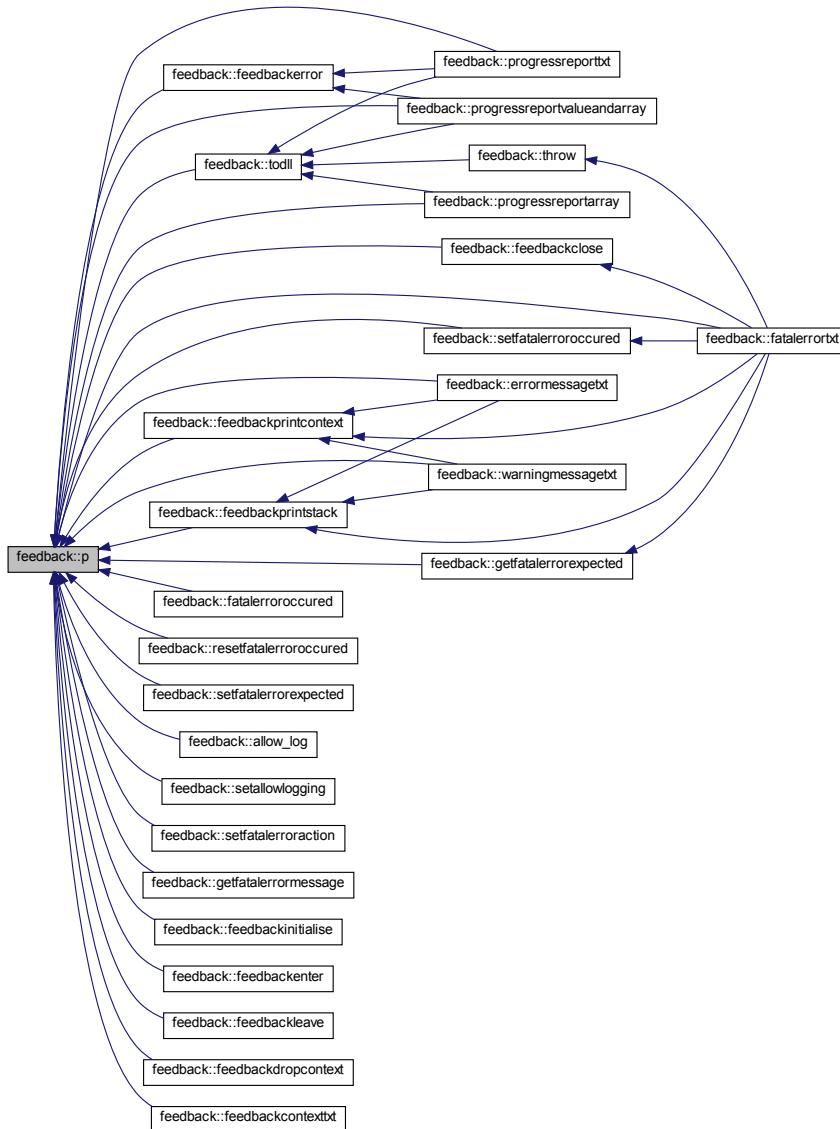
Definition at line 300 of file feedback.f90.

**6.1.2.26 integer function `feedback::p` ( ) [private]**

Returns the pointer to the feedback dll, avoiding many loadlibrary calls.

Definition at line 186 of file feedback.f90.

Here is the caller graph for this function:



**6.1.2.27 subroutine `feedback::progressreportarray`** ( integer, intent(in) *level*, character(len=\*) , intent(in) *msgText*, real(kind=wp), dimension(:), intent(in) *x*, integer, intent(in) *size* ) [private]

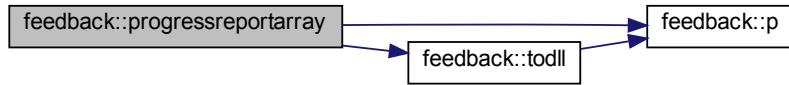
Print array values to report file.

#### Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>x</i>	x vector to write
in	<i>size</i>	the length of x

Definition at line 489 of file `feedback.f90`.

Here is the call graph for this function:



**6.1.2.28 subroutine `feedback::progressreportdouble`** ( `integer, intent(in) level`, `character(len=*) intent(in) msgtext, real(kind=dp), intent(in) value` ) [private]

Write a single message with a double precision real value.

#### Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Double value to add

Definition at line 472 of file `feedback.f90`.

**6.1.2.29 subroutine `feedback::progressreportint`** ( `integer, intent(in) level`, `character(len=*) intent(in) msgtext, integer, intent(in) value` ) [private]

Write a single message with an integer value.

#### Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 440 of file `feedback.f90`.

**6.1.2.30 subroutine `feedback::progressreportreal`** ( `integer, intent(in) level`, `character(len=*) intent(in) msgtext, real, intent(in) value` ) [private]

Write a single message with a real value.

#### Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Real value to add

Definition at line 456 of file `feedback.f90`.

**6.1.2.31 subroutine `feedback::progressreporttxt`** ( `integer, intent(in) level`, `character(len=*) intent(in) msgtext` ) [private]

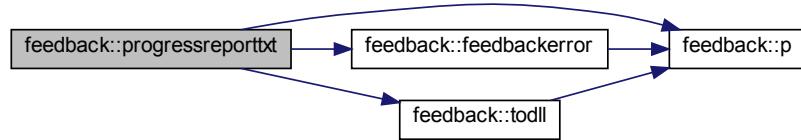
Write a single message to the output for the benefit of the user.

## Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Text to write

Definition at line 400 of file feedback.f90.

Here is the call graph for this function:



**6.1.2.32 subroutine `feedback::progressreportvalueandarray`** ( integer, intent(in) *level*, character(len=\*)*msgText*, real(kind=wp), intent(in) *z*, real(kind=wp), dimension(:), intent(in) *x*, integer, intent(in) *size* ) [private]

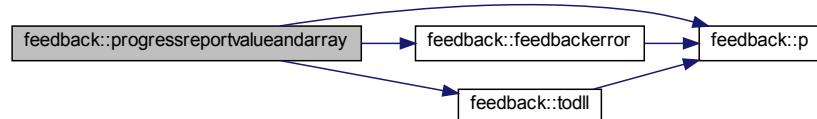
Print *z*-value and array values to report file.

## Parameters

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>z</i>	<i>z</i> value to write
in	<i>x</i>	<i>x</i> vector to write
in	<i>size</i>	the length of <i>x</i>

Definition at line 525 of file feedback.f90.

Here is the call graph for this function:



**6.1.2.33 subroutine, public `feedback::resetfatalerroroccurred`** ( )

Reset fatal error occurrence Mainly used in unit tests.

Definition at line 224 of file feedback.f90.

Here is the call graph for this function:



#### 6.1.2.34 subroutine, public feedback::setallowlogging ( logical, intent(in) value )

Set logging on/off.

##### Parameters

in	value	the value of allow_logging to be set
----	-------	--------------------------------------

Definition at line 286 of file feedback.f90.

Here is the call graph for this function:



#### 6.1.2.35 subroutine, public feedback::setfatalerroraction ( integer, intent(in) action )

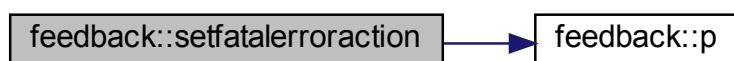
Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.

##### Parameters

in	action	the value of FatalErrorAction to be set (1, 2 or 3)
----	--------	---

Definition at line 313 of file feedback.f90.

Here is the call graph for this function:



6.1.2.36 subroutine, public `feedback::setfatalerrorexpected ( logical, intent(in) value )`

Set the value of FatalErrorExpected Mainly used in unit tests.

## Parameters

in	value	the value of FatalErrorExpected to be set
----	-------	---

Definition at line 261 of file feedback.f90.

Here is the call graph for this function:

6.1.2.37 subroutine `feedback::setfatalerroroccured ( logical, intent(in) value ) [private]`

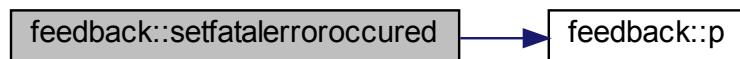
Set fatal error occurrence Mainly used in unit tests.

## Parameters

in	value	the value of FatalErrorOccured to be set
----	-------	--

Definition at line 236 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

6.1.2.38 subroutine, public `feedback::throw ( character(len=*), intent(in) msgText )`

Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.

**Parameters**

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 917 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

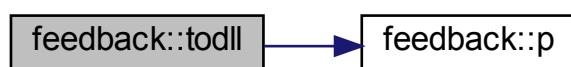


#### 6.1.2.39 logical function, public feedback::todll ( )

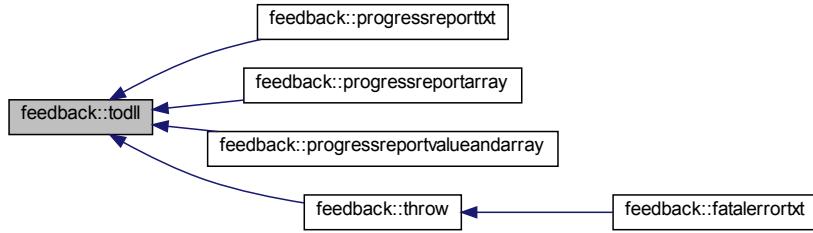
Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?

Definition at line 200 of file feedback.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.1.2.40 subroutine `feedback::warningmessagedouble` ( `character(len=*) intent(in) msgtext, real(kind=dp), intent(in) value` ) [private]

Write a warning message with a double precision real value.

##### Parameters

in	<code>msgtext</code>	Error message to write
in	<code>value</code>	Real (double) value to add

Definition at line 729 of file `feedback.f90`.

#### 6.1.2.41 subroutine `feedback::warningmessageint` ( `character(len=*) intent(in) msgtext, integer, intent(in) value` ) [private]

Write a warning message with an integer value.

##### Parameters

in	<code>msgtext</code>	Error message to write
in	<code>value</code>	Integer value to add

Definition at line 699 of file `feedback.f90`.

#### 6.1.2.42 subroutine `feedback::warningmessagereal` ( `character(len=*) intent(in) msgtext, real, intent(in) value` ) [private]

Write a warning message with a real value.

##### Parameters

in	<code>msgtext</code>	Error message to write
in	<code>value</code>	Real value to add

Definition at line 714 of file `feedback.f90`.

#### 6.1.2.43 subroutine `feedback::warningmessagetxt` ( `character(len=*) intent(in) msgtext` ) [private]

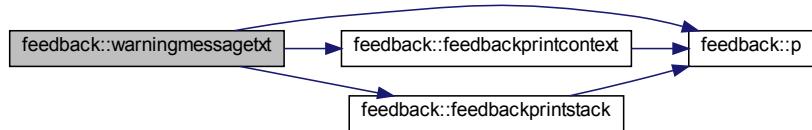
Write a warning message.

**Parameters**

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 676 of file feedback.f90.

Here is the call graph for this function:



### 6.1.3 Variable Documentation

#### 6.1.3.1 integer, parameter feedback\_parameters::dll = 3

Mode: write to dll.

Definition at line 17 of file feedback\_parameters.f90.

#### 6.1.3.2 integer, parameter feedback\_parameters::onlyfile = 1

Mode: write to file only.

Definition at line 15 of file feedback\_parameters.f90.

#### 6.1.3.3 integer, parameter feedback\_parameters::onlyscreen = 0

Mode: write to screen only.

Definition at line 14 of file feedback\_parameters.f90.

#### 6.1.3.4 integer, parameter feedback\_parameters::screenandfile = 2

Mode: write to both screen and file.

Definition at line 16 of file feedback\_parameters.f90.

#### 6.1.3.5 integer, parameter feedback\_parameters::verbosebasic = 0

Reporting: basic messages only.

Definition at line 19 of file feedback\_parameters.f90.

#### 6.1.3.6 integer, parameter feedback\_parameters::verbosedebugging = 2

Reporting: all messages.

Definition at line 21 of file feedback\_parameters.f90.

6.1.3.7 integer, parameter feedback\_parameters::verbosedetailed = 1

Reporting: detailed messages (including traceback)

Definition at line 20 of file feedback\_parameters.f90.

6.1.3.8 integer, parameter feedback\_parameters::verbosenone = -1

Reporting: NO messages.

Definition at line 18 of file feedback\_parameters.f90.



# Chapter 7

## Module Documentation

### 7.1 angleutilities Module Reference

Module with direction utility functions.

#### Functions/Subroutines

- real(kind=wp) function `anglebetween2directions` (`direction1, direction2`)

*Function angleBetween2Directions() for the computation of the angle between two directions.*

*The two directions must be given in degrees and within [0, 360]*

*The computed angle between these two directions is also in degrees, and within [0, 180]*

#### 7.1.1 Detailed Description

Module with direction utility functions.

#### 7.1.2 Function/Subroutine Documentation

##### 7.1.2.1 real( kind= wp) function `angleutilities::anglebetween2directions` ( `real( kind= wp), intent(in) direction1, real( kind= wp), intent(in) direction2` )

Function angleBetween2Directions() for the computation of the angle between two directions.

The two directions must be given in degrees and within [0, 360]

The computed angle between these two directions is also in degrees, and within [0, 180]

#### Parameters

in	<code>direction1</code>	first direction (an angle in degrees)
in	<code>direction2</code>	second direction (an angle in degrees)

#### Returns

angle between the two directions

Definition at line 24 of file angleUtilities.f90.

## 7.2 conversionfunctions Module Reference

### Functions/Subroutines

- subroutine, public [pqfrombeta](#) (beta, p, q)
 

*Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.*
- real(kind=wp) function, public [pfrombeta](#) (beta)
 

*Function for computing the non-exceedance probability (P) for a given reliability index (beta), assuming a standard normal distribution.*
- real(kind=wp) function, public [qfrombeta](#) (beta)
 

*Function for computing the exceedance probability (Q) for a given reliability index (beta), assuming a standard normal distribution.*
- subroutine, public [returntimefrombeta](#) (beta, returnTime)
 

*Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.*
- subroutine, public [freqfrombeta](#) (beta, freq)
 

*Subroutine for computing the frequency for a given reliability index (beta) assuming a standard normal distribution.*
- subroutine, public [logqfrombeta](#) (beta, logQ)
 

*Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (beta) assuming a standard normal distribution.*
- subroutine, public [betafromq](#) (q, beta)
 

*Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.*

### Variables

- real(kind=wp), parameter, private [qmin](#) = 1.0d-300
- real(kind=wp), parameter, private [upperlog](#) = 700.0d0
- real(kind=wp), parameter, private [ulimit](#) = 5.6d0

#### 7.2.1 Function/Subroutine Documentation

##### 7.2.1.1 subroutine, public conversionfunctions::betafromq ( real(kind=wp), intent(in) q, real(kind=wp), intent(out) beta )

Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.

This routine is essentially the code for a standard normal distribution, in which a probability is input and a u-value (beta) is returned.

*Note:* Theoretically the probability q has to be between 0.0 and 1.0

Values outside this range can be used in this routine but are set equal to the bounds of the theoretical range:

- values larger than 1.0 are set equal to 1.0 (or, to be precisely: 1 - 1e-300), and
  - values smaller than 0.0 are set equal to 0.0 (or, to be precisely: 1e-300)
- This is done with the construction "qsr = max(qsr, 1.0d-300)"

**Todo** Is it necessary to extend the routine betaFromQ with a check on the probability of non-exceedance ( $q$ ) and a error message?  $Q$  has to be a between 0 and 1.

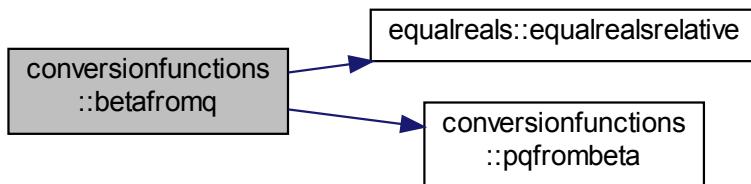
This routine is extreemly often called! Therefore with this check the speed of the computation is decreased. There is another argument not to implement this check: Sometimes by combining the computed probabilities the combined probability is slightly more than 1 or slightly less than 0 (errors of rounding), then the computation will end eith this error message. Now this error of rounding is repaired in this routine betaFromQ. If a probability is outside the allowed area the probability is changed to the lower or upper bound. The disadvantage of this method is that programming errors are no longer visible: a probability of for intance two doesn't generate an error!

#### Parameters

in	$q$	Probability of exceedance
out	$\beta$	Reliability index

Definition at line 234 of file conversionFunctions.f90.

Here is the call graph for this function:



#### 7.2.1.2 subroutine, public conversionfunctions::freqfrommbeta ( real(kind=wp), intent(in) $\beta$ , real(kind=wp), intent(out) freq )

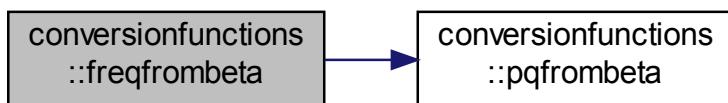
Subroutine for computing the frequency for a given reliability index ( $\beta$ ) assuming a standard normal distribution.

#### Parameters

in	$\beta$	Reliability index
out	freq	Return time

Definition at line 165 of file conversionFunctions.f90.

Here is the call graph for this function:



### 7.2.1.3 subroutine, public conversionfunctions::logqfrombeta ( real(kind=wp), intent(in) *beta*, real(kind=wp), intent(out) *logQ* )

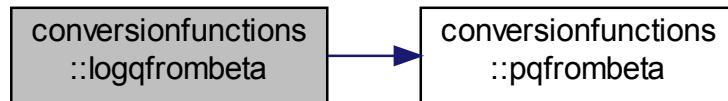
Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (*beta*) assuming a standard normal distribution.

## Parameters

in	<i>beta</i>	Reliability index
out	<i>logq</i>	Log q

Definition at line 188 of file conversionFunctions.f90.

Here is the call graph for this function:



#### 7.2.1.4 real(kind = wp) function, public conversionfunctions::pfrombeta ( real(kind = wp), intent(in) beta )

Function for computing the non-exceedance probability (P) for a given reliability index (beta), assuming a standard normal distribution.

## Parameters

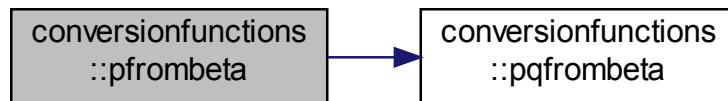
in	<i>beta</i>	Reliability index
----	-------------	-------------------

## Returns

Probability of non-exceedance

Definition at line 90 of file conversionFunctions.f90.

Here is the call graph for this function:



#### 7.2.1.5 subroutine, public conversionfunctions::pqfrombeta ( real(kind=wp), intent(in) beta, real(kind=wp), intent(out) p, real(kind=wp), intent(out) q )

Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.

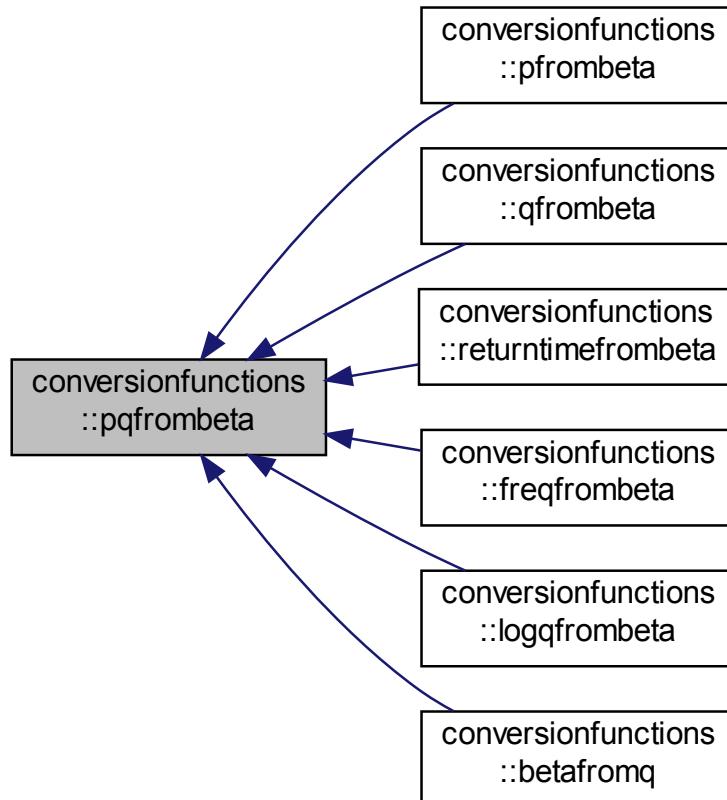
Approximation method is taken from: Approximation Formula 26.2.17 for Normal Probability Function, Handbook of Mathematical Functions, Abramowitz & Stegun, page 932

### Parameters

in	<i>beta</i>	Reliability index
out	<i>p</i>	Probability of non-exceedance
out	<i>q</i>	Probability of exceedance

Definition at line 44 of file conversionFunctions.f90.

Here is the caller graph for this function:



### 7.2.1.6 real(kind = wp) function, public conversionfunctions::qfrombeta ( real(kind = wp), intent(in) *beta* )

Function for computing the exceedance probability (Q) for a given reliability index (*beta*), assuming a standard normal distribution.

### Parameters

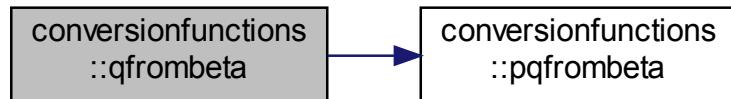
in	<i>beta</i>	Reliability index
----	-------------	-------------------

### Returns

Probability of exceedance

Definition at line 116 of file conversionFunctions.f90.

Here is the call graph for this function:



**7.2.1.7 subroutine, public conversionfunctions::returntimefrombeta ( real(kind=wp), intent(in) beta, real(kind=wp), intent(out) returnTime )**

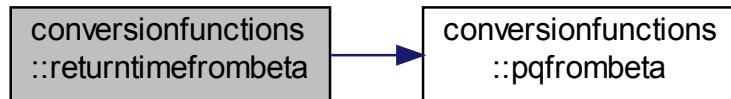
Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.

#### Parameters

in	<i>beta</i>	Reliability index
out	<i>returntime</i>	Return time

Definition at line 142 of file conversionFunctions.f90.

Here is the call graph for this function:



## 7.2.2 Variable Documentation

**7.2.2.1 real(kind = wp), parameter, private conversionfunctions::qmin = 1.0d-300**

Definition at line 23 of file conversionFunctions.f90.

**7.2.2.2 real(kind = wp), parameter, private conversionfunctions::ulimit = 5.6d0**

Definition at line 31 of file conversionFunctions.f90.

**7.2.2.3 real(kind = wp), parameter, private conversionfunctions::upperlog = 700.0d0**

Definition at line 27 of file conversionFunctions.f90.

## 7.3 dliovertopping Module Reference

Calculate one type of overtopping.

### Functions/Subroutines

- subroutine, public **calculateqo** (load, geometryInput, dikeHeight, modelFactors, overtopping, success, errorText, verbosity, logFile)
 

*Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.*
- subroutine, public **calculateqof** (load, geometryF, dikeHeight, modelFactors, overtopping, success, errorText, logging)
 

*Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.*
- subroutine, public **calcZvalue** (criticalOvertoppingRate, modelFactors, Qo, z, success, errorMessage)
 

*Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.*
- subroutine, public **validateinputc** (geometryInput, dikeHeight, modelFactors, success, errorText)
 

*Subroutine that validates the geometry Wrapper for ValidateInputC: convert C-like input structures to Fortran input structures.*
- subroutine, public **validateinputf** (geometryF, dikeHeight, modelFactors, success, errorText)
 

*Subroutine that validates the geometry.*
- subroutine, public **versionnumber** (version)
 

*Subroutine that delivers the version number.*
- type(overtoppinggeometrytypef) function **geometry\_c\_f** (geometryInput)
 

*Private subroutine that converts geometry from c-pointer to fortran struct.*

### 7.3.1 Detailed Description

Calculate one type of overtopping.

### 7.3.2 Function/Subroutine Documentation

- 7.3.2.1 subroutine, public dliovertopping::calculateqo ( type(tload), intent(in) *load*, type(overtoppinggeometrytype), intent(in) *geometryInput*, real(kind=wp), intent(in) *dikeHeight*, type(tpovertoppinginput), intent(inout) *modelFactors*, type (tpovertopping), intent(out) *overtopping*, logical, intent(out) *success*, character(len=\*), intent(out) *errorText*, integer, intent(in) *verbosity*, character(len=\*), intent(in) *logFile* )

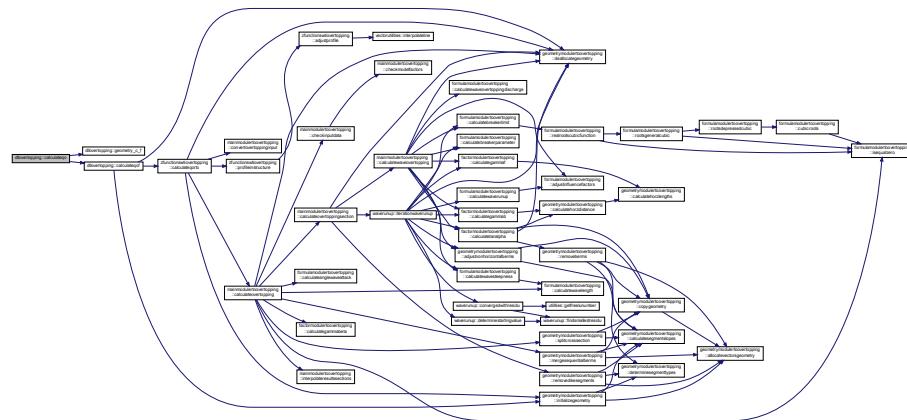
Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.

#### Parameters

in	<i>geometryinput</i>	struct with geometry and roughness as c-pointers
in	<i>load</i>	struct with waterlevel and wave parameters
in,out	<i>modelfactors</i>	struct with modelfactors
out	<i>overtopping</i>	structure with overtopping results
out	<i>success</i>	flag for success
out	<i>errortext</i>	error message (only set if not successful)
in	<i>verbosity</i>	level of verbosity
in	<i>logfile</i>	filename of logfile

Definition at line 40 of file dliovertopping.f90.

Here is the call graph for this function:



**7.3.2.2 subroutine, public dliovertopping::calculateqof ( type(tpload), intent(in) *load*, type(overtoppinggeometrytypef), intent(in) *geometryF*, real(kind=wp), intent(in) *dikeHeight*, type(tpovertoppinginput), intent(inout) *modelFactors*, type (tpovertopping), intent(out) *overtopping*, logical, intent(out) *success*, character(len=\*), intent(out) *errorText*, type(tlogging), intent(in) *logging* )**

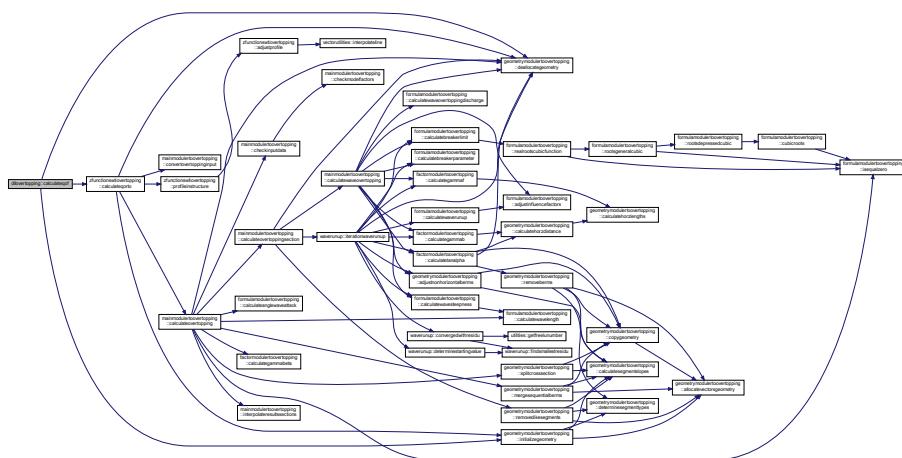
Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.

#### Parameters

in	<i>geometryf</i>	struct with geometry and roughness
in	<i>load</i>	struct with waterlevel and wave parameters
in, out	<i>modelFactors</i>	struct with modelFactors
out	<i>overtopping</i>	structure with overtopping results
out	<i>success</i>	flag for success
out	<i>errortext</i>	error message (only set if not successful)
in	<i>logging</i>	logging struct

Definition at line 71 of file dliovertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.3.2.3 subroutine, public** `dllovertopping::calcZvalue` ( `real(kind=wp), intent(in) criticalOvertoppingRate,`  
`type(tpovertoppinginput), intent(inout) modelFactors, real(kind=wp), intent(in) Qo, real(kind=wp), intent(out) z, logical,`  
`intent(out) success, character(len=*), intent(out) errorMessage` )

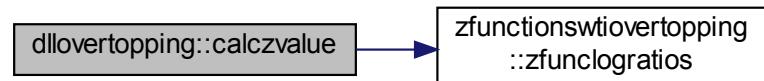
Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.

#### Parameters

in	<i>criticalovertoppingrate</i>	critical overtoppingrate
in,out	<i>modelfactors</i>	struct with modelfactors
in	<i>qo</i>	calculated discharge
out	<i>z</i>	z value
out	<i>errormessage</i>	error message (only if not successful)
out	<i>success</i>	flag for success

Definition at line 104 of file `dllOvertopping.f90`.

Here is the call graph for this function:



**7.3.2.4 type(overtoppinggeometrytype) function** `dllovertopping::geometry_c_f` ( `type(overtoppinggeometrytype), intent(in) geometryInput ) [private]`

Private subroutine that converts geometry from c-pointer to fortran struct.

#### Parameters

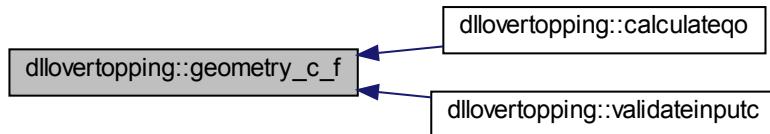
in	<i>geometryinput</i>	struct with geometry and roughness as c-pointers
----	----------------------	--

#### Returns

fortran struct with geometry and roughness

Definition at line 225 of file `dllOvertopping.f90`.

Here is the caller graph for this function:



**7.3.2.5 subroutine, public dlovertopping::validateinputc ( type(overtoppinggeometrytype), intent(in) geometryInput, real(kind=wp), intent(in) dikeHeight, type(tpovertoppinginput), intent(inout) modelFactors, logical, intent(out) success, character(len=\*), intent(out) errorText )**

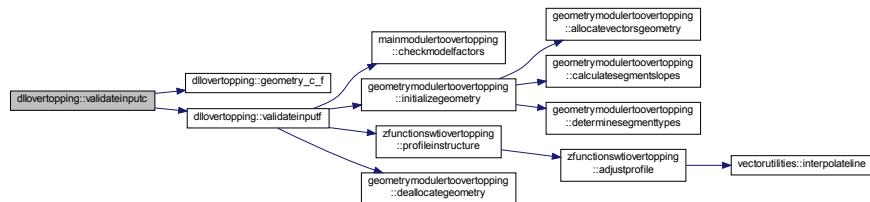
Subroutine that validates the geometry Wrapper for ValidateInputF: convert C-like input structures to Fortran input structures.

#### Parameters

in	<i>geometryinput</i>	struct with geometry and roughness as c-pointers
in, out	<i>modelfactors</i>	struct with modelfactors
out	<i>success</i>	flag for success
out	<i>errortext</i>	error message (only set if not successful)

Definition at line 124 of file dllOvertopping.f90.

Here is the call graph for this function:



**7.3.2.6 subroutine, public dlovertopping::validateinputf ( type(overtoppinggeometrytypef), intent(in) geometryF, real(kind=wp), intent(in) dikeHeight, type(tpovertoppinginput), intent(inout) modelFactors, logical, intent(out) success, character(len=\*), intent(out) errorText )**

Subroutine that validates the geometry.

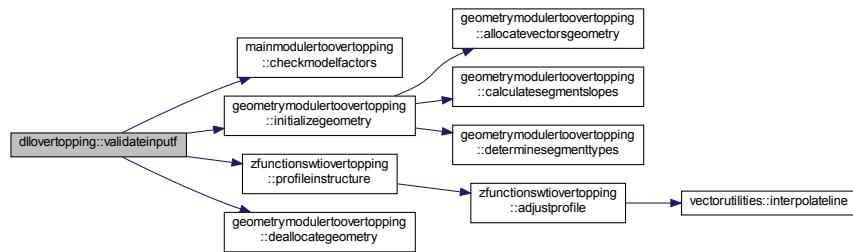
#### Parameters

in	<i>geometryf</i>	struct with geometry and roughness
in, out	<i>modelfactors</i>	struct with modelFactors
out	<i>success</i>	flag for success

out	errortext	error message (only set if not successful)
-----	-----------	--

Definition at line 146 of file dllOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.3.2.7 subroutine, public dllovertopping::versionnumber ( character(len=\*), intent(out) version )

Subroutine that delivers the version number.

#### Parameters

out	version	version number
-----	---------	----------------

Definition at line 203 of file dllOvertopping.f90.

## 7.4 equalreals Module Reference

Module with functions to test quality of reals.

### Functions/Subroutines

- logical function [equalrealsrelative](#) (x, y, margin)
 

*equalRealRelative Function to test for equality of two reals with a relative value for the margin equalRealRelative =  $|x-y| \leqslant \text{margin} * (|x|+|y|)/2$*
- logical function [equalrealsabsolute](#) (x, y, margin)
 

*equalRealAbsolute Function to test for equality of two reals with an absolute value for the margin equalRealAbsolute =  $|x-y| \leqslant \text{margin}$*
- logical function [equalrealsinvector](#) (n, vector, margin)
 

*Function to test for equality of reals in a vector with a relative value of the margin.*
- logical function [equalvectorswithreals](#) (n, vector1, vector2, margin, equality)
 

*Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.*

- logical function `equaltableswithreals` (matrix1, matrix2, margin)

*Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.*

- logical function `equalvectorswithrealsweightedmargin` (n, vector1, vector2, margin, equality)

*Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal Here, margin is weighted according to tested numbers.*

- logical function `equalvectorswithintegers` (n, vector1, vector2, equality)

#### 7.4.1 Detailed Description

Module with functions to test quality of reals.

#### 7.4.2 Function/Subroutine Documentation

##### 7.4.2.1 logical function `equalreals::equalrealsabsolute` ( `real(kind=wp), intent(in) x`, `real(kind=wp), intent(in) y`, `real(kind=wp), intent(in) margin` )

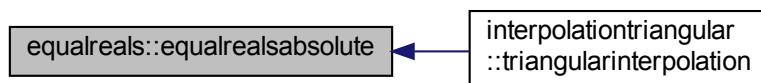
`equalRealsAbsolute` Function to test for equality of two reals with an absolute value for the margin  $\text{equalReals} \leftarrow \text{Absolute} = |x-y| \leq \text{margin}$

###### Parameters

in	x	First real for the comparison
in	y	Second real for the comparison
in	margin	Absolute value for the margin

Definition at line 40 of file `equalReals.f90`.

Here is the caller graph for this function:



##### 7.4.2.2 logical function `equalreals::equalrealsinvector` ( `integer, intent(in) n`, `real(kind=wp), dimension(n), intent(in) vector`, `real(kind=wp), intent(in) margin` )

Function to test for equality of reals in a vector with a relative value of the margin.

###### Parameters

in	n	Vector length
in	vector	Vector with reals
in	margin	Relative value for the margin

Definition at line 55 of file `equalReals.f90`.

Here is the call graph for this function:



#### 7.4.2.3 logical function equalreals::equalrealsrelative ( real(kind=wp), intent(in) x, real(kind=wp), intent(in) y, real(kind=wp), intent(in) margin )

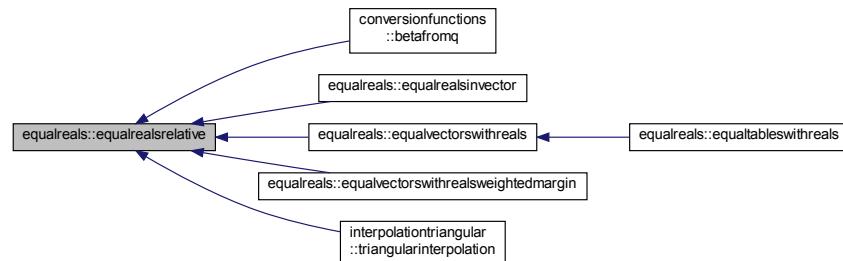
equalRealRelative Function to test for equality of two reals with a relative value for the margin  $\text{equalRealRelative} = |x-y| \leq \text{margin} * (|x|+|y|)/2$

##### Parameters

in	x	First real for the comparison
in	y	Second real for the comparison
in	margin	Relative value for the margin

Definition at line 24 of file equalReal.f90.

Here is the caller graph for this function:



#### 7.4.2.4 logical function equalreals::equaltableswithreals ( real(kind=wp), dimension(:, :), intent(in) matrix1, real(kind=wp), dimension(:, :), intent(in) matrix2, real(kind=wp), intent(in) margin )

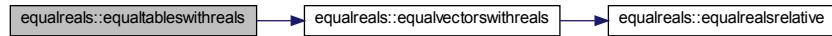
Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

##### Parameters

in	matrix1	Matrix 1 with reals
in	matrix2	Matrix 2 with reals
in	margin	Relative value for the margin

Definition at line 110 of file equalReal.f90.

Here is the call graph for this function:



**7.4.2.5 logical function equalreals::equalvectorswithintegers** ( integer, intent(in) *n*, integer, dimension(*n*), intent(in) *vector1*, integer, dimension(*n*), intent(in) *vector2*, logical, dimension(*n*), intent(out) *equality* )

#### Parameters

in	<i>n</i>	Vector length
in	<i>vector1</i>	Vector1 with reals (size <i>n</i> )
in	<i>vector2</i>	Vector2 with reals (size <i>n</i> )
out	<i>equality</i>	Vector (size <i>n</i> ) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 200 of file equalReals.f90.

**7.4.2.6 logical function equalreals::equalvectorswithreals** ( integer, intent(in) *n*, real(kind=wp), dimension(*n*), intent(in) *vector1*, real(kind=wp), dimension(*n*), intent(in) *vector2*, real(kind=wp), intent(in) *margin*, logical, dimension(*n*), intent(out) *equality* )

Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.

#### Parameters

in	<i>n</i>	Vector length
in	<i>vector1</i>	Vector1 with reals (size <i>n</i> )
in	<i>vector2</i>	Vector2 with reals (size <i>n</i> )
in	<i>margin</i>	Relative value for the margin
out	<i>equality</i>	Vector (size <i>n</i> ) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 77 of file equalReals.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.2.7 logical function equalreals::equalvectorswithrealsweightedmargin ( integer, intent(in) *n*, real(kind=wp), dimension(*n*), intent(in) *vector1*, real(kind=wp), dimension(*n*), intent(in) *vector2*, real(kind=wp), intent(in) *margin*, logical, dimension(*n*), intent(out) *equality* )

Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal. Here, margin is weighted according to tested numbers.

#### Parameters

in	<i>n</i>	Vector length
in	<i>vector1</i>	Vector1 with reals (size <i>n</i> )
in	<i>vector2</i>	Vector2 with reals (size <i>n</i> )
in	<i>margin</i>	Relative value for the margin
out	<i>equality</i>	Vector (size <i>n</i> ) with elements which indicates if the corresponding elements of the two input vectors are the same

Definition at line 159 of file equalReals.f90.

Here is the call graph for this function:



## 7.5 factormodulertoovertopping Module Reference

### Functions/Subroutines

- subroutine, public [calculatetanalpha](#) (*h*, *Hm0*, *z2*, *geometry*, *tanAlpha*, *succes*, *errorMessage*)
   
*calculateTanAlpha representative slope angle*
- subroutine, public [calculategammabeta](#) (*Hm0*, *Tm\_10*, *beta*, *gammaBeta\_z*, *gammaBeta\_o*)
   
*calculateGammaBeta influence factor angle of wave attack*
- subroutine, public [calculategammaf](#) (*h*, *ksi0*, *ksi0Limit*, *gammaB*, *z2*, *geometry*, *gammaF*, *succes*, *errorMessage*)
   
*calculateGammaF influence factor roughness*
- subroutine, public [calculategammab](#) (*h*, *Hm0*, *z2*, *geometry*, *NbermSegments*, *gammaB*, *succes*, *errorMessage*)
   
*calculateGammaB influence factor berms*

### 7.5.1 Function/Subroutine Documentation

7.5.1.1 subroutine, public factormodulertoovertopping::calculategammab ( real(wp), intent(in) *h*, real(wp), intent(in) *Hm0*, real(wp), intent(in) *z2*, type(tpgeometry), intent(in) *geometry*, integer, intent(in) *NbermSegments*, real(wp), intent(out) *gammaB*, logical, intent(out) *succes*, character(len=\*) intent(out) *errorMessage* )

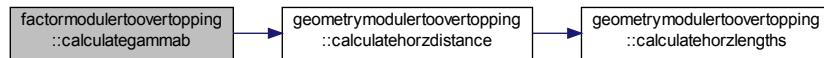
*calculateGammaB influence factor berms*

#### Parameters

in	<i>h</i>	local water level (m+NAP)
in	<i>hm0</i>	significant wave height (m)
in	<i>z2</i>	2% wave run-up (m)
in	<i>geometry</i>	structure with geometry data
in	<i>nbermsegments</i>	number of berm segments
out	<i>gammab</i>	influence factor berms
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 271 of file factorModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.5.1.2 subroutine, public factormodulertoovertopping::calculategammabeta ( real(wp), intent(inout) *Hm0*, real(wp), intent(inout) *Tm\_10*, real(wp), intent(in) *beta*, real(wp), intent(out) *gammaBeta\_z*, real(wp), intent(out) *gammaBeta\_o* )

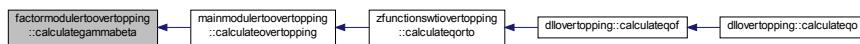
calculateGammaBeta influence factor angle of wave attack

#### Parameters

in,out	<i>hm0</i>	significant wave height (m)
in,out	<i>tm_10</i>	spectral wave period (s)
in	<i>beta</i>	angle of wave attack (degree)
out	<i>gammabeta_z</i>	influence factor angle of wave attack 2% wave run-up
out	<i>gammabeta_o</i>	influence factor angle of wave attack overtopping

Definition at line 114 of file factorModuleRTOvertopping.f90.

Here is the caller graph for this function:



### 7.5.1.3 subroutine, public factormodulertoovertopping::calculategammaf ( real(wp), intent(in) *h*, real(wp), intent(in) *ksi0*, real(wp), intent(in) *ksi0Limit*, real(wp), intent(in) *gammaB*, real(wp), intent(in) *z2*, type(tpgeometry), intent(in) *geometry*, real(wp), intent(out) *gammaF*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

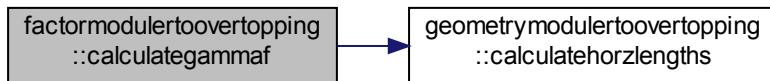
calculateGammaF influence factor roughness

### Parameters

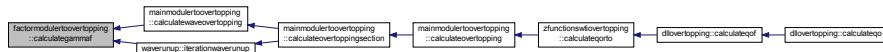
in	<i>h</i>	local water level (m+NAP)
in	<i>ksi0</i>	breaker parameter
in	<i>ksi0limit</i>	limit value breaker parameter
in	<i>gammab</i>	influence factor berms
in	<i>z2</i>	2% wave run-up (m)
in	<i>geometry</i>	structure with geometry data
out	<i>gammaf</i>	influence factor roughness
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 154 of file factorModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.5.1.4 subroutine, public factormodulertoovertopping::calculatetanalpha ( real(wp), intent(in) *h*, real(wp), intent(in) *Hm0*, real(wp), intent(in) *z2*, type(tpgeometry), intent(in) *geometry*, real(wp), intent(out) *tanAlpha*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )**

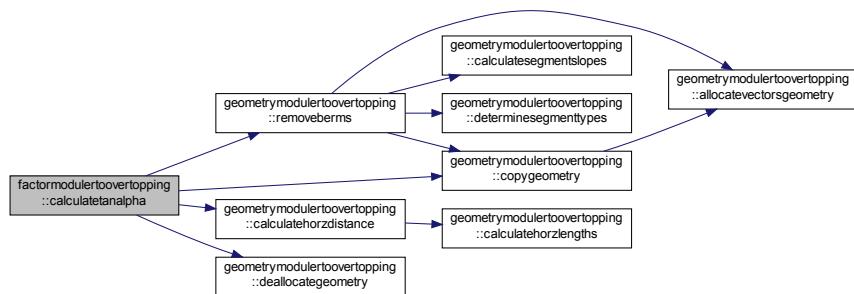
*calculateTanAlpha* representative slope angle

### Parameters

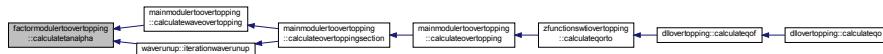
in	<i>h</i>	local water level (m+NAP)
in	<i>hm0</i>	significant wave height (m)
in	<i>z2</i>	2% wave run-up (m)
in	<i>geometry</i>	structure with geometry data
out	<i>tanalpha</i>	representative slope angle
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 35 of file factorModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.6 feedback Module Reference

### Data Types

- interface `errormessage`
- interface `fatalerror`
- interface `feedbackcontext`
- interface `progressreport`
- interface `warningmessage`

### Functions/Subroutines

- integer function `p ()`

*Returns the pointer to the feedback dll, avoiding many loadlibrary calls.*
- logical function, public `tolll ()`

*Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?*
- logical function, public `fatalerroroccurred (value)`

*Has a fatal error occurred? Mainly used in unit tests.*
- subroutine, public `resetfatalerroroccurred ()`

*Reset fatal error occurrence Mainly used in unit tests.*
- subroutine `setfatalerroroccurred (value)`

*Set fatal error occurrence Mainly used in unit tests.*
- logical function, public `getfatalerrorexpected (value)`

*Get value of FatalErrorExpected Mainly used in unit tests.*
- subroutine, public `setfatalerrorexpected (value)`

*Set the value of FatalErrorExpected Mainly used in unit tests.*
- logical function, public `allow_log ()`

*Is logging active?*
- subroutine, public `setallowlogging (value)`

- subroutine, public **getthreadid** (threadId)
 

*Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.*
- subroutine, public **setfatalerroraction** (action)
 

*Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.*
- subroutine, public **getfatalerrormessage** (message)
 

*Get the last of FatalErrorMessage.*
- subroutine, public **feedbackerror** (error)
 

*Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.*
- subroutine, public **feedbackclose**

*Close the log file.*
- subroutine, public **feedbackinitialise** (output, logfile, verbosity)
 

*Initialise the feedback subsystem.*
- subroutine **progressreporttxt** (level, msgtext)
 

*Write a single message to the output for the benefit of the user.*
- subroutine **progressreportint** (level, msgtext, value)
 

*Write a single message with an integer value.*
- subroutine **progressreportreal** (level, msgtext, value)
 

*Write a single message with a real value.*
- subroutine **progressreportdouble** (level, msgtext, value)
 

*Write a single message with a double precision real value.*
- subroutine **progressreportarray** (level, msgText, x, size)
 

*Print array values to report file.*
- subroutine **progressreportvalueandarray** (level, msgText, z, x, size)
 

*Print z-value and array values to report file.*
- subroutine, public **feedbackenter** (routine)
 

*Register a new routine level.*
- subroutine, public **feedbackleave**

*Drop a routine level from the stack trace.*
- subroutine, public **feedbackprintstack**

*Print the stack trace as known via feedbackEnter and feedbackLeave.*
- subroutine **errormessagetxt** (msgtext)
 

*Write an error message.*
- subroutine **errormessageint** (msgtext, value)
 

*Write an error message with an integer value.*
- subroutine **errormessagereal** (msgtext, value)
 

*Write an error message with a real value.*
- subroutine **errormessagedouble** (msgtext, value)
 

*Write an error message with a double precision real value.*
- subroutine **warningmessagetxt** (msgtext)
 

*Write a warning message.*
- subroutine **warningmessageint** (msgtext, value)
 

*Write a warning message with an integer value.*
- subroutine **warningmessagereal** (msgtext, value)
 

*Write a warning message with a real value.*
- subroutine **warningmessagedouble** (msgtext, value)
 

*Write a warning message with a double precision real value.*
- subroutine **fatalerrortxt** (msgtext)

- subroutine **fatalerrorint** (msgtext, value)
 

*Write a fatal error message.*
- subroutine **fatalerrorreal** (msgtext, value)
 

*Write a fatal error message with an integer value.*
- subroutine **fatalerrordouble** (msgtext, value)
 

*Write a fatal error message with a real value.*
- subroutine, public **feedbackdropcontext**

*Drop the entire context.*
- subroutine, public **feedbackprintcontext**

*Register a context for error messages.*
- subroutine **feedbackcontexttxt** (text)
 

*Register a context for error messages.*
- subroutine **feedbackcontextint** (text, value)
 

*Register a context for error messages - with integer value.*
- subroutine **feedbackcontextreal** (text, value)
 

*Register a context for error messages - with real value.*
- subroutine **feedbackcontextdouble** (text, value)
 

*Register a context for error messages - with double precision real value.*
- subroutine, public **throw** (msgText)
 

*Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.*

## Variables

- integer, parameter **id** = 0
 

*Thread ID; multi threading is not supported yet!*
- integer, parameter **sp** = kind(1.0)
- integer, parameter **dp** = kind(1.0d0)
- integer, parameter **wp** = kind(1.0d0)

### 7.6.1 Variable Documentation

#### 7.6.1.1 integer, parameter **feedback::dp** = kind(1.0d0)

Definition at line 114 of file feedback.f90.

#### 7.6.1.2 integer, parameter **feedback::id** = 0

Thread ID; multi threading is not supported yet!

Definition at line 111 of file feedback.f90.

#### 7.6.1.3 integer, parameter **feedback::sp** = kind(1.0)

Definition at line 113 of file feedback.f90.

#### 7.6.1.4 integer, parameter **feedback::wp** = kind(1.0d0)

Definition at line 115 of file feedback.f90.

## 7.7 feedback\_parameters Module Reference

### Variables

- integer, parameter `onfatalerrorstop` = 1  
*Action on fatal error: exit the program.*
- integer, parameter `onfatalerrorthrowexception` = 2  
*Action on fatal error: throw exception (for use as dll)*
- integer, parameter `onfatalerrorunittest` = 3  
*Action on fatal error: handle as unit test.*
- integer, parameter `onlyscreen` = 0  
*Mode: write to screen only.*
- integer, parameter `onlyfile` = 1  
*Mode: write to file only.*
- integer, parameter `screenandfile` = 2  
*Mode: write to both screen and file.*
- integer, parameter `dll` = 3  
*Mode: write to dll.*
- integer, parameter `verbosenone` = -1  
*Reporting: NO messages.*
- integer, parameter `verbosebasic` = 0  
*Reporting: basic messages only.*
- integer, parameter `verbosetailed` = 1  
*Reporting: detailed messages (including traceback)*
- integer, parameter `verbosedebugging` = 2  
*Reporting: all messages.*

### 7.7.1 Variable Documentation

#### 7.7.1.1 integer, parameter `feedback_parameters::onfatalerrorstop` = 1

Action on fatal error: exit the program.

Definition at line 10 of file `feedback_parameters.f90`.

#### 7.7.1.2 integer, parameter `feedback_parameters::onfatalerrorthrowexception` = 2

Action on fatal error: throw exception (for use as dll)

Definition at line 11 of file `feedback_parameters.f90`.

#### 7.7.1.3 integer, parameter `feedback_parameters::onfatalerrorunittest` = 3

Action on fatal error: handle as unit test.

Definition at line 12 of file `feedback_parameters.f90`.

## 7.8 feedbackdll Module Reference

### Data Types

- type `feedbackdata`

## Functions/Subroutines

- subroutine **initialise** (output, logfile, verbosity)  
*Initialise the feedback dll.*
- subroutine **printmessage** (msgtext)  
*Write a message to output device (as selected in Initialise)*
- subroutine **outputtodll** (active)  
*Is output is to another dll?*
- subroutine **getallowlogging** (value)  
*Is logging active.*
- subroutine **setallowlogging** (value)  
*Set logging on/off.*
- subroutine **getverbosity** (level)  
*Get the current verbosity level.*
- subroutine **setonfatalerroraction** (action)  
*Set the OnFatalErrorAction.*
- subroutine **getonfatalerroraction** (onfatalerrorAction)  
*Get the current OnFatalErrorAction.*
- subroutine **getfeedbackerror** (error)  
*Get the error status (true/false)*
- subroutine **setfatalerrormessage** (message)  
*Set the errormessage.*
- subroutine **getfatalerrormsg** (message)  
*Get the last error message.*
- subroutine **getverbosylevel** (verbosityLevel)  
*Get the verbosity level.*
- subroutine **pushroutine** (routine, success)  
*Add a routine name to the call stack.*
- subroutine **poproutine** (success)  
*Removes and get a routine name from the call stack.*
- subroutine **printstack**  
*Print the call stack.*
- subroutine **getstacklevel** (level)  
*Get the number of routines in the call stack.*
- subroutine **closefile**  
*Closes the output file.*
- subroutine **getunitnumber** (lun)  
*Get the unit number of the logfile.*
- subroutine **dropcontext**  
*Reset the context stack.*
- subroutine **printcontext**  
*Print the context stack.*
- subroutine **contexttxt** (text, success)  
*Add a line of text to the context stack.*
- subroutine **getsetfatalerrorexpected** (func, value)  
*Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.*
- subroutine **getsetfatalerroroccurred** (func, value)  
*Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.*
- subroutine **writetodll** (msgText)  
*Write message to another dll.*

## Variables

- type([feedbackdata](#)) **fbdata**  
*Array holding the actual information per thread (multiple threads not yet implemented)*
- integer **fatalerroraction** = 1
- character(len=255) **fatalerrormessage**
- integer, parameter **id** = 0
- logical **allow\_logging** = .true.
- logical **isfatalerrorexpected** = .false.  
*if isFatalErrorExpected = .true. then log error message in fatalErrorMsgText and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop*
- logical **isfatalerroroccurred** = .false.

### 7.8.1 Function/Subroutine Documentation

#### 7.8.1.1 subroutine feedbackdll::closefile( ) [private]

Closes the output file.

Definition at line 288 of file feedbackDLL.f90.

#### 7.8.1.2 subroutine feedbackdll::contexttxt( character(len=\*), intent(in) **text**, logical, intent(out) **success** ) [private]

Add a line of text to the context stack.

##### Parameters

in	<b>text</b>	line of text to be pushed on the context stack
out	<b>success</b>	.false., if the stack already reached its maximum size

Definition at line 335 of file feedbackDLL.f90.

#### 7.8.1.3 subroutine feedbackdll::dropcontext( ) [private]

Reset the context stack.

Definition at line 307 of file feedbackDLL.f90.

#### 7.8.1.4 subroutine feedbackdll::getallowlogging( logical, intent(out) **value** ) [private]

Is logging active.

##### Parameters

out	<b>value</b>	the current value of allow_logging
-----	--------------	------------------------------------

Definition at line 133 of file feedbackDLL.f90.

#### 7.8.1.5 subroutine feedbackdll::getfatalerrormsg( character(len=\*), intent(out) **message** ) [private]

Get the last error message.

##### Parameters

<code>out</code>	<code>message</code>	current error message
------------------	----------------------	-----------------------

Definition at line 203 of file feedbackDLL.f90.

#### 7.8.1.6 subroutine `feedbackdll::getfeedbackerror` ( `logical, intent(out) error` ) [private]

Get the error status (true/false)

Parameters

<code>out</code>	<code>error</code>	last error status
------------------	--------------------	-------------------

Definition at line 183 of file feedbackDLL.f90.

#### 7.8.1.7 subroutine `feedbackdll::getonfatalerroraction` ( `integer, intent(out) onfatalerrorAction` ) [private]

Get the current OnFatalErrorAction.

Parameters

<code>out</code>	<code>onfatalerroraction</code>	current FatalErrorAction: 1, 2 or 3
------------------	---------------------------------	-------------------------------------

Definition at line 172 of file feedbackDLL.f90.

#### 7.8.1.8 subroutine `feedbackdll::getsetfatalerrorexpected` ( `character(len=3), intent(in) func, logical, intent(inout) value` ) [private]

Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.

Parameters

<code>in,out</code>	<code>value</code>	the value of FatalErrorExpected
<code>in</code>	<code>func</code>	the action: 'get' or 'set' (case sensitive!)

Definition at line 357 of file feedbackDLL.f90.

#### 7.8.1.9 subroutine `feedbackdll::getsetfatalerroroccurred` ( `character(len=3), intent(in) func, logical, intent(inout) value` ) [private]

Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.

Parameters

<code>in,out</code>	<code>value</code>	the value of FatalErrorOccured
<code>in</code>	<code>func</code>	the action: 'get' or 'set' (case sensitive!)

Definition at line 374 of file feedbackDLL.f90.

#### 7.8.1.10 subroutine `feedbackdll::getstacklevel` ( `integer, intent(out) level` ) [private]

Get the number of routines in the call stack.

Parameters

<code>out</code>	<code>level</code>	number of routines in the call stack
------------------	--------------------	--------------------------------------

Definition at line 278 of file feedbackDLL.f90.

**7.8.1.11 subroutine feedbackdll::getunitnumber ( integer, intent(out) *lun* ) [private]**

Get the unit number of the logfile.

## Parameters

out	<i>lun</i>	integer unit number of logfile
-----	------------	--------------------------------

Definition at line 297 of file feedbackDLL.f90.

7.8.1.12 subroutine **feedbackdll::getverbosity** ( integer, intent(out) *level* ) [private]

Get the current verbosity level.

## Parameters

out	<i>level</i>	the current verbosity level: -1, .. , 2
-----	--------------	---

Definition at line 151 of file feedbackDLL.f90.

7.8.1.13 subroutine **feedbackdll::getverbositylevel** ( integer, intent(out) *verbosityLevel* ) [private]

Get the verbosity level.

## Parameters

out	<i>verbosityLevel</i>	current verbosityLevel: -1, .., 2
-----	-----------------------	-----------------------------------

Definition at line 213 of file feedbackDLL.f90.

7.8.1.14 subroutine **feedbackdll::initialise** ( integer, intent(in) *output*, character(len=\*) *logfile*, integer, intent(in) *verbosity* ) [private]

Initialise the feedback dll.

## Parameters

in	<i>output</i>	option: output to screen, file, both, or another dll
in	<i>logfile</i>	name of the log file to be used (only used if writing to file)
in	<i>verbosity</i>	level for displaying the messages

Definition at line 51 of file feedbackDLL.f90.

7.8.1.15 subroutine **feedbackdll::outputtodll** ( logical, intent(out) *active* ) [private]

Is output is to another dll?

## Parameters

out	<i>active</i>	.true., if output is to another dll
-----	---------------	-------------------------------------

Definition at line 123 of file feedbackDLL.f90.

7.8.1.16 subroutine **feedbackdll::poproutine** ( logical, intent(out) *success* ) [private]

Removes and get a routine name from the call stack.

## Parameters

out	<i>success</i>	if .false., the stack is already empty
-----	----------------	--

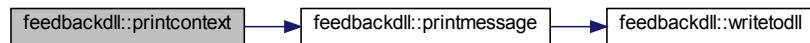
Definition at line 240 of file feedbackDLL.f90.

### 7.8.1.17 subroutine feedbackdll::printcontext ( ) [private]

Print the context stack.

Definition at line 317 of file feedbackDLL.f90.

Here is the call graph for this function:



### 7.8.1.18 subroutine feedbackdll::printmessage ( character(len=\*) msgtext, intent(in) msgtext ) [private]

Write a message to output device (as selected in Initialise)

Parameters

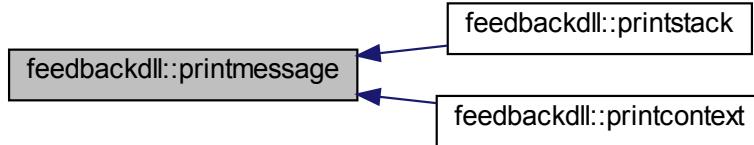
in	msgtext	text to be printed to screen, file or dll
----	---------	---

Definition at line 96 of file feedbackDLL.f90.

Here is the call graph for this function:



Here is the caller graph for this function:

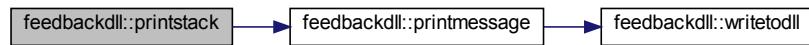


### 7.8.1.19 subroutine feedbackdll::printstack ( ) [private]

Print the call stack.

Definition at line 255 of file feedbackDLL.f90.

Here is the call graph for this function:



**7.8.1.20 subroutine feedbackdll::pushroutine ( character(len=\*), intent(in) routine, logical, intent(out) success ) [private]**

Add a routine name to the call stack.

#### Parameters

in	<i>routine</i>	routine name to be added to the stack
out	<i>success</i>	.false., if the stack already reached its maximum size

Definition at line 223 of file feedbackDLL.f90.

**7.8.1.21 subroutine feedbackdll::setallowlogging ( logical, intent(in) value ) [private]**

Set logging on/off.

#### Parameters

in	<i>value</i>	the value for allow_logging to be set
----	--------------	---------------------------------------

Definition at line 142 of file feedbackDLL.f90.

**7.8.1.22 subroutine feedbackdll::setfatalerrormessage ( character(len=\*), intent(in) message ) [private]**

Set the errormessage.

#### Parameters

in	<i>message</i>	error message to be set
----	----------------	-------------------------

Definition at line 193 of file feedbackDLL.f90.

**7.8.1.23 subroutine feedbackdll::setonfatalerroraction ( integer, intent(in) action ) [private]**

Set the OnFatalErrorAction.

#### Parameters

in	<i>action</i>	FatalErrorAction to be set. Must be in range: 1, 2 or 3
----	---------------	---

Definition at line 162 of file feedbackDLL.f90.

**7.8.1.24 subroutine feedbackdll::writetodll ( character(len=\*), intent(in) msgText ) [private]**

Write message to another dll.

### Parameters

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 391 of file feedbackDLL.f90.

Here is the caller graph for this function:



## 7.8.2 Variable Documentation

### 7.8.2.1 logical feedbackdll::allow\_logging = .true.

Definition at line 40 of file feedbackDLL.f90.

### 7.8.2.2 integer feedbackdll::fatalerroraction = 1

Definition at line 37 of file feedbackDLL.f90.

### 7.8.2.3 character(len=255) feedbackdll::fatalerrormessage

Definition at line 38 of file feedbackDLL.f90.

### 7.8.2.4 type(feedbackdata) feedbackdll::fbdata

Array holding the actual information per thread (multiple threads not yet implemented)

Definition at line 35 of file feedbackDLL.f90.

### 7.8.2.5 integer, parameter feedbackdll::id = 0

Definition at line 39 of file feedbackDLL.f90.

### 7.8.2.6 logical feedbackdll::isfatalerrorexpected = .false.

if isFatalErrorExpected = .true. then log error message in fatalErrorMsgtext and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop

Definition at line 41 of file feedbackDLL.f90.

### 7.8.2.7 logical feedbackdll::isfatalerroroccurred = .false.

Definition at line 43 of file feedbackDLL.f90.

## 7.9 fileutilities Module Reference

### Functions/Subroutines

- subroutine `readtable` (*filename*, *nColumns*, *table*)
- subroutine `writetablevalues` (*filename*, *table*)
- subroutine `removefile` (*filename*)

*Routine to throw away a file.*

#### 7.9.1 Function/Subroutine Documentation

7.9.1.1 subroutine `fileutilities::readtable` ( character(len=\*)*intent(in) filename*, integer, intent(in) *nColumns*, real(wp), dimension(:, :), allocatable *table* )

##### Parameters

<i>in</i>	<i>filename</i>	Name of the file
<i>in</i>	<i>ncolumns</i>	Number of columns
	<i>table</i>	Table with contents of file

Definition at line 20 of file `fileUtilities.f90`.

Here is the call graph for this function:



7.9.1.2 subroutine `fileutilities::removefile` ( character(len=\*)*intent(in) filename* )

*Routine to throw away a file.*

##### Parameters

<i>in</i>	<i>filename</i>	file to delete
-----------	-----------------	----------------

Definition at line 78 of file `fileUtilities.f90`.

Here is the call graph for this function:



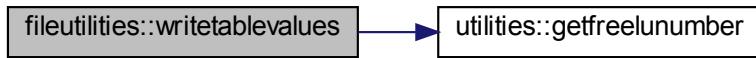
7.9.1.3 subroutine `fileutilities::writetablevalues` ( character(len=\*)*intent(in) filename*, real(wp), dimension(:, :), intent(in) *table* )

### Parameters

in	<i>filename</i>	Name of the file to write
in	<i>table</i>	Table with contents to write in file

Definition at line 51 of file fileUtilities.f90.

Here is the call graph for this function:



## 7.10 formulamodulertoovertopping Module Reference

### Functions/Subroutines

- subroutine, public **calculatewaverunup** (Hm0, ksi0, ksi0Limit, gammaB, gammaF, gammaBeta, modelFactors, z2, succes, errorMessage)
   
*calculateWaveRunup: calculate wave runup*
- subroutine, public **calculatewaveovertoppingdischarge** (h, Hm0, tanAlpha, gammaB, gammaF, gammaBeta, ksi0, hCrest, modelFactors, Qo, succes, errorMessage)
   
*calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge*
- subroutine, public **calculatewavelength** (Tm\_10, L0)
   
*calculateWaveLength: calculate the wave length*
- subroutine, public **calculatwavesteeepness** (Hm0, Tm\_10, s0, succes, errorMessage)
   
*calculateWaveSteepness: calculate the wave steepness*
- subroutine, public **calculatebreakerparameter** (tanAlpha, s0, ksi0, succes, errorMessage)
   
*calculateBreakerParameter: calculate the breaker parameter*
- subroutine, public **calculateanglewaveattack** (phi, psi, beta)
   
*calculateAngleWaveAttack: calculate the angle of wave attack*
- subroutine, public **calculatebreakerlimit** (modelFactors, gammaB, ksi0Limit, succes, errorMessage)
   
*calculateBreakerLimit: calculate the breaker limit*
- subroutine, public **adjustinfluencefactors** (gammaB, gammaF, gammaBeta, gammaBetaType, ksi0, ksi0Limit, succes, errorMessage)
   
*adjustInfluenceFactors: adjust the influence factors*
- subroutine, public **realrootscubicfunction** (a, b, c, d, N, x, succes, errorMessage)
   
*realRootsCubicFunction: calculate the roots of a cubic function*
- subroutine, public **rootsgeneralcubic** (a, b, c, d, z, succes, errorMessage)
   
*rootsGeneralCubic: calculate the roots of a generic cubic function*
- subroutine, public **rootsdepressedcubic** (p, q, z)
   
*rootsDepressedCubic: calculate the roots of a depressed cubic function*
- subroutine, public **cubicroots** (z, roots)
   
*cubicRoots: calculate the roots of a cubic function*
- logical function, public **isequalreal** (x1, x2)
   
*isEqualReal: are two reals (almost) equal*
- logical function, public **isequalzero** (x)
   
*isEqualZero: is a real (almost) zero*

### 7.10.1 Function/Subroutine Documentation

7.10.1.1 subroutine, public formulamodulertoovertopping::adjustinfluencefactors ( real(wp), intent(inout) *gammaB*, real(wp), intent(inout) *gammaF*, real(wp), intent(inout) *gammaBeta*, integer, intent(in) *gammaBetaType*, real(wp), intent(in) *ksi0*, real(wp), intent(in) *ksi0Limit*, logical, intent(out) *succes*, character(len=\*) intent(out) *errorMessage* )

*adjustInfluenceFactors*: adjust the influence factors

#### Parameters

in,out	<i>gammab</i>	influence factor berms
in,out	<i>gammaf</i>	influence factor roughness
in,out	<i>gammabeta</i>	influence factor angle of wave attack
in	<i>gammabetatype</i>	type influence factor angle of wave attack: 1 = wave run-up, 2 = overtopping
in	<i>ksi0</i>	breaker parameter
in	<i>ksi0limit</i>	limit value breaker parameter
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 382 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.10.1.2 subroutine, public formulamodulertoovertopping::calculateanglewaveattack ( real(wp), intent(in) *phi*, real(wp), intent(in) *psi*, real(wp), intent(out) *beta* )

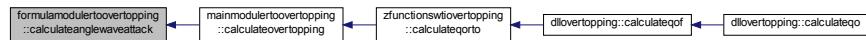
*calculateAngleWaveAttack*: calculate the angle of wave attack

#### Parameters

in	<i>phi</i>	wave direction (degree)
in	<i>psi</i>	dike normal (degree)
out	<i>beta</i>	angle of wave attack (degree)

Definition at line 285 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:

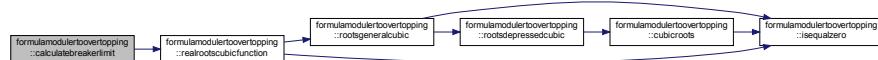


7.10.1.3 subroutine, public formulamodulertoovertopping::calculatebreakerlimit ( type (tpovertoppinginput), intent(in) *modelFactors*, real(wp), intent(in) *gammaB*, real(wp), intent(out) *ksi0Limit*, logical, intent(out) *succes*, character(len=\*) intent(out) *errorMessage* )

*calculateBreakerLimit*: calculate the breaker limit

Definition at line 308 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.4 subroutine, public formulanodulerootovertopping::calculatebreakerparameter ( real(wp), intent(in) tanAlpha, real(wp), intent(in) s0, real(wp), intent(out) ksl0, logical, intent(out) succes, character(len=\*) , intent(out) errorMessage )

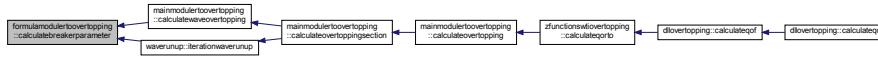
`calculateBreakerParameter`: calculate the breaker parameter

## Parameters

in	<i>tanalpha</i>	representative slope angle
in	<i>s0</i>	wave steepness
out	<i>ksi0</i>	breaker parameter
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 244 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.10.1.5 subroutine, public formulaModulertoOvertopping::calculateWavelength ( real(wp), intent(in) Tm\_10, real(wp), intent(out) L0 )

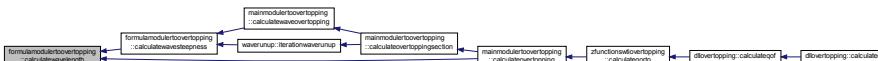
`calculateWaveLength`: calculate the wave length

## Parameters

in	<i>tm_10</i>	spectral wave period (s)
out	<i>l0</i>	wave length (m)

Definition at line 181 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.10.1.6 subroutine, public formulamodulertoovertopping::calculatewaveovertoppingdischarge ( real(wp), intent(in) *h*, real(wp), intent(in) *Hm0*, real(wp), intent(in) *tanAlpha*, real(wp), intent(in) *gammaB*, real(wp), intent(in) *gammaF*, real(wp), intent(in) *gammaBeta*, real(wp), intent(in) *ksi0*, real(wp), intent(in) *hCrest*, type(tpovertoppinginput), intent(in) *modelFactors*, real(wp), intent(out) *Qo*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge

## Parameters

in	<i>h</i>	local water level (m+NAP)
in	<i>hm0</i>	significant wave height (m)
in	<i>tanalpha</i>	representative slope angle
in	<i>gammab</i>	influence factor berms
in	<i>gammaf</i>	influence factor roughness
in	<i>gammabeta</i>	influence factor angle of wave attack
in	<i>ksi0</i>	breaker parameter
in	<i>hcrest</i>	crest level (m+NAP)
in	<i>modelfactors</i>	structure with model factors
out	<i>qo</i>	wave overtopping discharge (l/m per s)
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 83 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.10.1.7 subroutine, public formulaModulerToOvertopping::calculateWaverunup ( real(wp), intent(in) *Hm0*, real(wp), intent(in) *ksi0*, real(wp), intent(in) *ksi0Limit*, real(wp), intent(inout) *gammaB*, real(wp), intent(inout) *gammaF*, real(wp), intent(inout) *gammaBeta*, type (tpovertoppinginput), intent(in) *modelFactors*, real(wp), intent(out) *z2*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

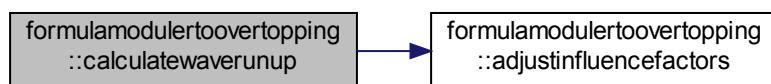
calculateWaveRunup: calculate wave runup

## Parameters

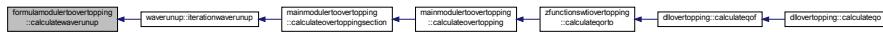
in	<i>hm0</i>	significant wave height (m)
in	<i>ksi0</i>	breaker parameter
in	<i>ksi0limit</i>	limit value breaker parameter
in,out	<i>gammab</i>	influence factor berms
in,out	<i>gammaf</i>	influence factor roughness
in,out	<i>gammabeta</i>	influence factor angle of wave attack
in	<i>modelfactors</i>	structure with model factors
out	<i>z2</i>	2% wave run-up (m)
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 34 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.10.1.8 subroutine, public formulamodulertoovertopping::calculatewavestepness ( real(wp), intent(in)  $Hm0$ , real(wp), intent(in)  $Tm\_10$ , real(wp), intent(out)  $s0$ , logical, intent(out)  $succes$ , character(len=\*) intent(out)  $errorMessage$  )**

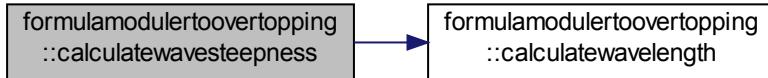
**calculateWaveSteepness:** calculate the wave steepness

#### Parameters

in	<i>hm0</i>	significant wave height (m)
in	<i>tm_10</i>	spectral wave period (s)
out	<i>s0</i>	wave steepness
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 202 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.10.1.9 subroutine, public formulamodulertoovertopping::cubicroots ( double complex, intent(in)  $z$ , double complex, dimension(3), intent(out)  $roots$  )**

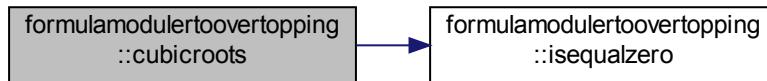
**cubicRoots:** calculate the roots of a cubic function

#### Parameters

in	<i>z</i>	complex number
out	<i>roots</i>	cubic roots

Definition at line 628 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.10.1.10 logical function, public formulaModulertoOvertopping::isEqualReal ( real(wp), intent(in) x1, real(wp), intent(in) x2 )

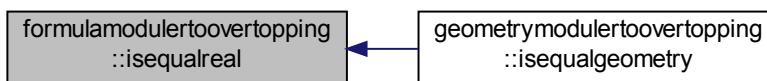
isEqualReal: are two reals (almost) equal

Parameters

in	x1	first real
in	x2	second real

Definition at line 669 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



#### 7.10.1.11 logical function, public formulaModulertoOvertopping::isEqualZero ( real(wp), intent(in) x )

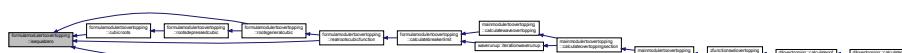
isEqualZero: is a real (almost) zero

Parameters

in	x	real number
----	---	-------------

Definition at line 693 of file formulaModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.10.1.12 subroutine, public formulamodulertoovertopping::realrootsCubicFunction ( real(wp), intent(in) *a*, real(wp), intent(in) *b*, real(wp), intent(in) *c*, real(wp), intent(in) *d*, integer, intent(out) *N*, real(wp), dimension(3), intent(out) *x*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

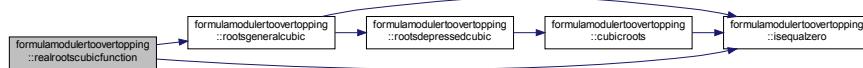
realRootsCubicFunction: calculate the roots of a cubic function

### Parameters

in	<i>d</i>	coefficients cubic function
out	<i>n</i>	number of real roots cubic function
out	<i>x</i>	real roots cubic function
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 480 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.13 subroutine, public formulaModulerToOvertopping::rootsDepressedCubic ( real(wp), intent(in) *p*, real(wp), intent(in) *q*, double complex, dimension(3), intent(out) *z* )

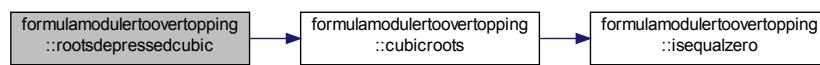
rootsDepressedCubic: calculate the roots of a depressed cubic function

### Parameters

in	<i>q</i>	coefficients depressed cubic
out	<i>z</i>	roots depressed cubic

Definition at line 588 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.10.1.14 subroutine, public formulaModulerToOvertopping::rootGeneralCubic ( real(wp), intent(in) *a*, real(wp), intent(in) *b*, real(wp), intent(in) *c*, real(wp), intent(in) *d*, double complex, dimension(3), intent(out) *z*, logical, intent(out) *succes*, character(len=\*), intent(out) *errormessage* )

rootsGeneralCubic: calculate the roots of a generic cubic function

### Parameters

in	<i>d</i>	coefficients cubic function
out	<i>z</i>	roots cubic function
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 538 of file formulaModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.11 ftnunit Module Reference

### Data Types

- interface [assert\\_comparable](#)
- interface [assert\\_equal](#)
- interface [assert\\_inbetween](#)
- interface [proc](#)

### Functions/Subroutines

- subroutine, public [setruntestlevel](#) (runTestLevel)
 

*Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.*
- subroutine, public [settesttitle](#) (message)
 

*Set the title that is displayed in the report of the test run.*
- subroutine, public [test](#) (proc, text, ignore)
- subroutine, public [testwithlevel](#) (proc, text, testlevel, ignore)
- subroutine, public [runtests\\_init](#)
- subroutine [runtests\\_init\\_priv](#)
- subroutine, public [runtests\\_final](#) (stop)
- logical function [issilent](#) ()
- subroutine [networkingdir](#)
- subroutine, public [runtests](#) (testproc)
- subroutine [expect\\_program\\_stop](#)
- subroutine, public [assert\\_true](#) (cond, text)
- subroutine, public [assert\\_false](#) (cond, text)
- subroutine [assert\\_equal\\_logical](#) (value1, value2, text)
- subroutine [assert\\_equal\\_logical1d](#) (array1, array2, text)
- subroutine [assert\\_equal\\_string](#) (value1, value2, text)
- subroutine [assert\\_equal\\_string1d](#) (array1, array2, text)

- subroutine `assert_equal_int` (value1, value2, text)
- subroutine `assert_equal_int1d` (array1, array2, text)
- subroutine `assert_comparable_real` (value1, value2, margin, text)
- subroutine `assert_comparable_real1d` (array1, array2, margin, text)
- subroutine `assert_comparable_real2d` (array1, array2, margin, text)
- subroutine `assert_comparable_double` (value1, value2, margin, text)
- subroutine `assert_comparable_double1d` (array1, array2, margin, text)
- subroutine `assert_comparable_double2d` (array1, array2, margin, text)
- subroutine `assert_inbetween_real` (value, vmin, vmax, text)
- subroutine `assert_inbetween_double` (value, vmin, vmax, text)
- subroutine, public `assert_files_comparable` (filename1, filename2, text, tolerance)
- logical function `ftnunit_file_exists` (filename)
- subroutine `ftnunit_remove_file` (filename)
- subroutine `ftnunit_make_empty_file` (filename)
- subroutine `ftnunit_write_html_header`
- subroutine `ftnunit_write_html_footer`
- subroutine `colour_number` (number, coloured, colour1, colour2, allowed)
- subroutine `ftnunit_write_html_test_begin` (text)
- subroutine `ftnunit_write_html_ignored` (lun)
- subroutine `ftnunit_write_html_cpu` (cpu)
- subroutine `ftnunit_write_html_previous_failed`
- subroutine `ftnunit_write_html_previous_stopped`
- subroutine `ftnunit_write_html_close_row` (lun)
- subroutine `ftnunit_write_html_failed_logic` (text, expected)
- subroutine `ftnunit_write_html_failed_equivalent` (text)
- subroutine `ftnunit_write_html_failed_equivalent1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_string` (text, value1, value2)
- subroutine `ftnunit_write_html_failed_string1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_int` (text, value1, value2)
- subroutine `ftnunit_write_html_failed_int1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_real` (text, value1, value2)
- subroutine `ftnunit_write_html_failed_real1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_real2d` (text, idx1, idx2, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_double` (text, value1, value2)
- subroutine `ftnunit_write_html_failed_double1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_double2d` (text, idx1, idx2, value1, value2, addtext)
- subroutine `ftnunit_write_html_failed_inbetween_real` (text, value, vmin, vmax)
- subroutine `ftnunit_write_html_failed_inbetween_double` (text, value, vmin, vmax)
- subroutine `ftnunit_write_html_failed_files` (text, string1, string2, string3)

## Variables

- procedure(`proc`), pointer, public `subptrtestprograminit` => null()
- procedure(`proc`), pointer, public `subptrtestinit` => null()
- integer, parameter `mode_all` = 0
- integer, parameter `mode_single` = 1
- integer, parameter `mode_list` = 2
- integer, parameter `dp` = kind(1.0d0)
- integer, save `run_test_level` = 99
- integer, save `single_test` = -1
- integer, save `test_mode` = `mode_all`
- integer, save `last_test`
- integer, save `testno`
- integer, save `nofails`

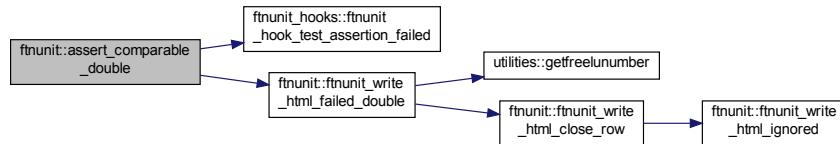
- integer, save `nofails_prev`
- integer, save `notests_run`
- integer, save `notests_failed`
- integer, save `notests_ignored`
- integer, save `notests_work_in_progress`
- integer, save `noruns`
- logical, save `call_final` = .true.
- logical, save `previous` = .false.
- integer, save `failed_asserts` = 0
- logical, save `has_run` = .false.
- logical, save `ignore_test` = .false.
- logical, save `ignore_previous_test` = .false.
- character(len=20), save `html_file` = 'ftnunit.html'
- character(len=80), save `testname`
- character(len=80), save `testtitle` = 'Result of unit tests'
- character(len=32), save `ignore_reason`
- character(len=32), save `ignore_reason_previous_test` = ''
- character(len=\*), parameter `work_in_progress` = 'work-in-progress'

### 7.11.1 Function/Subroutine Documentation

7.11.1.1 subroutine `ftnunit::assert_comparable_double` ( `real(kind=dp), intent(in) value1`, `real(kind=dp), intent(in) value2`, `real(kind=dp), intent(in) margin`, `character(len=*), intent(in) text` ) [private]

Definition at line 966 of file ftnunit.f90.

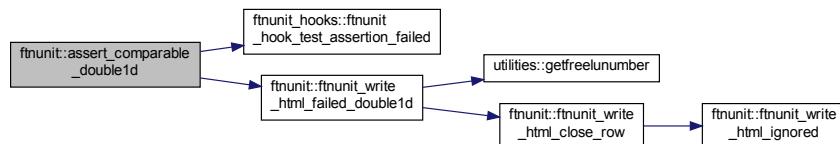
Here is the call graph for this function:



7.11.1.2 subroutine `ftnunit::assert_comparable_double1d` ( `real(kind=dp), dimension(:), intent(in) array1`, `real(kind=dp), dimension(:), intent(in) array2`, `real(kind=dp), intent(in) margin`, `character(len=*), intent(in) text` ) [private]

Definition at line 997 of file ftnunit.f90.

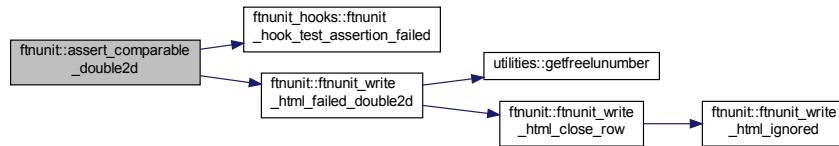
Here is the call graph for this function:



7.11.1.3 subroutine `ftnunit::assert_comparable_double2d` ( `real(kind=dp), dimension(:, :), intent(in) array1`, `real(kind=dp), dimension(:, :), intent(in) array2`, `real(kind=dp), intent(in) margin`, `character(len=*) intent(in) text` ) [private]

Definition at line 1061 of file ftnunit.f90.

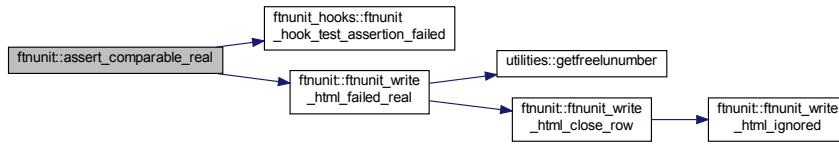
Here is the call graph for this function:



7.11.1.4 subroutine `ftnunit::assert_comparable_real` ( `real, intent(in) value1`, `real, intent(in) value2`, `real, intent(in) margin`, `character(len=*) intent(in) text` ) [private]

Definition at line 802 of file ftnunit.f90.

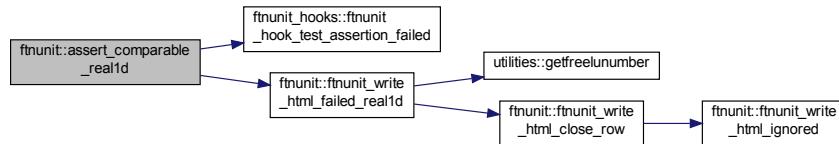
Here is the call graph for this function:



7.11.1.5 subroutine `ftnunit::assert_comparable_real1d` ( `real, dimension(:), intent(in) array1`, `real, dimension(:), intent(in) array2`, `real, intent(in) margin`, `character(len=*) intent(in) text` ) [private]

Definition at line 834 of file ftnunit.f90.

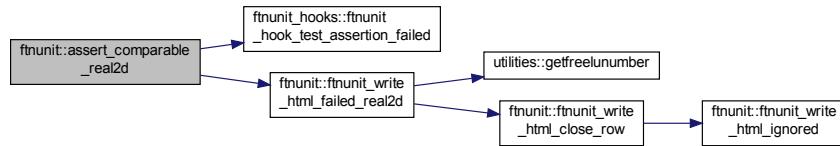
Here is the call graph for this function:



7.11.1.6 subroutine `ftnunit::assert_comparable_real2d` ( `real, dimension(:, :), intent(in) array1`, `real, dimension(:, :), intent(in) array2`, `real, intent(in) margin`, `character(len=*) intent(in) text` ) [private]

Definition at line 898 of file ftnunit.f90.

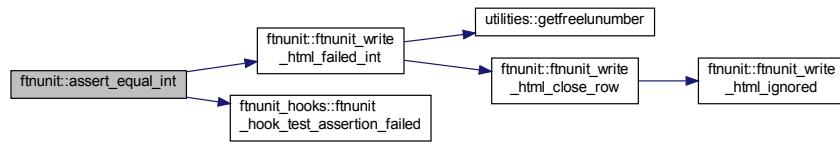
Here is the call graph for this function:



**7.11.1.7 subroutine ftnunit::assert\_equal\_int ( integer, intent(in) value1, integer, intent(in) value2, character(len=\*), intent(in) text ) [private]**

Definition at line 724 of file ftnunit.f90.

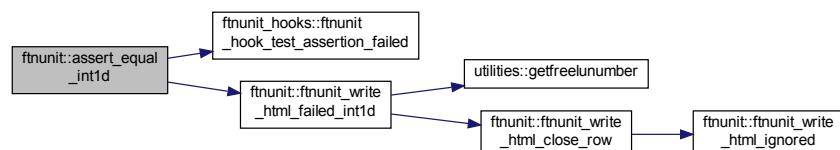
Here is the call graph for this function:



**7.11.1.8 subroutine ftnunit::assert\_equal\_int1d ( integer, dimension(:), intent(in) array1, integer, dimension(:), intent(in) array2, character(len=\*), intent(in) text ) [private]**

Definition at line 748 of file ftnunit.f90.

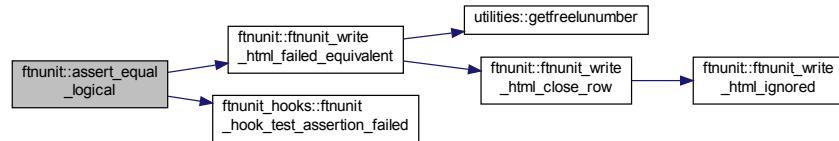
Here is the call graph for this function:



**7.11.1.9 subroutine ftnunit::assert\_equal\_logical ( logical, intent(in) value1, logical, intent(in) value2, character(len=\*), intent(in) text ) [private]**

Definition at line 575 of file ftnunit.f90.

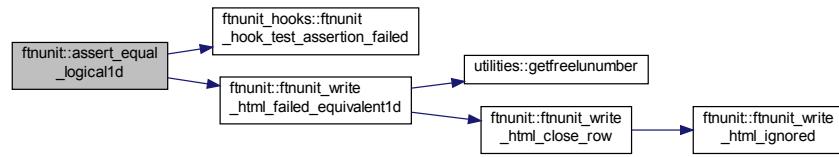
Here is the call graph for this function:



**7.11.1.10 subroutine ftunit::assert\_equal\_logical1d ( logical, dimension(:), intent(in) array1, logical, dimension(:), intent(in) array2, character(len=\*), intent(in) text ) [private]**

Definition at line 599 of file ftnunit.f90.

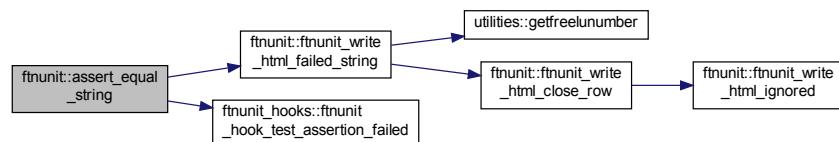
Here is the call graph for this function:



**7.11.1.11 subroutine ftunit::assert\_equal\_string ( character(len=\*), intent(in) value1, character(len=\*), intent(in) value2, character(len=\*), intent(in) text ) [private]**

Definition at line 649 of file ftnunit.f90.

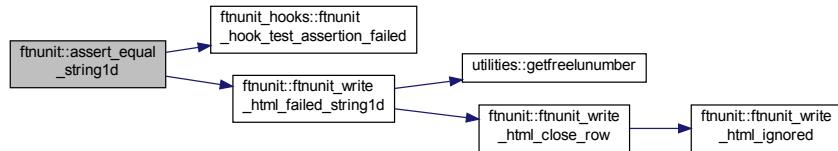
Here is the call graph for this function:



**7.11.1.12 subroutine ftunit::assert\_equal\_string1d ( character(len=\*), dimension(:), intent(in) array1, character(len=\*), dimension(:), intent(in) array2, character(len=\*), intent(in) text ) [private]**

Definition at line 675 of file ftnunit.f90.

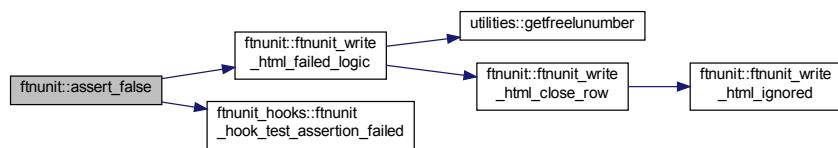
Here is the call graph for this function:



#### 7.11.1.13 subroutine, public ftnunit::assert\_false ( logical, intent(in) cond, character(len=\*) intent(in) text )

Definition at line 552 of file ftnunit.f90.

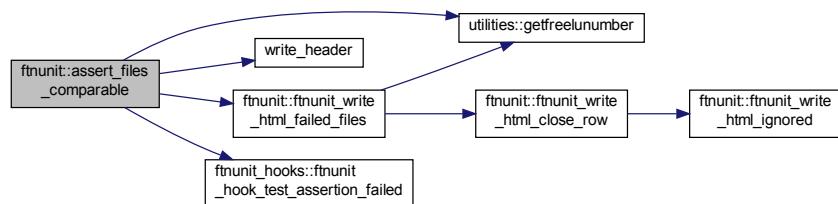
Here is the call graph for this function:



#### 7.11.1.14 subroutine, public ftnunit::assert\_files\_comparable ( character(len=\*) intent(in) filename1, character(len=\*) intent(in) filename2, character(len=\*) intent(in) text, real, intent(in), optional tolerance )

Definition at line 1213 of file ftnunit.f90.

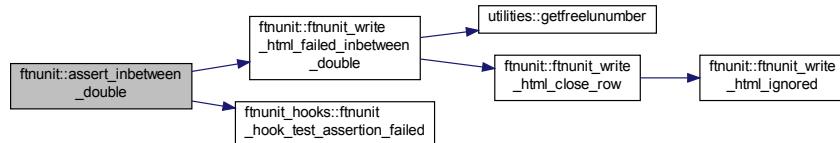
Here is the call graph for this function:



#### 7.11.1.15 subroutine ftnunit::assert\_inbetween\_double ( real(kind=dp), intent(in) value, real(kind=dp), intent(in) vmin, real(kind=dp), intent(in) vmax, character(len=\*) intent(in) text ) [private]

Definition at line 1168 of file ftnunit.f90.

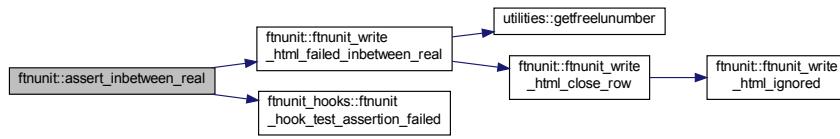
Here is the call graph for this function:



**7.11.1.16 subroutine ftnunit::assert\_inbetween\_real ( real, intent(in) value, real, intent(in) vmin, real, intent(in) vmax, character(len=\*), intent(in) text ) [private]**

Definition at line 1129 of file ftnunit.f90.

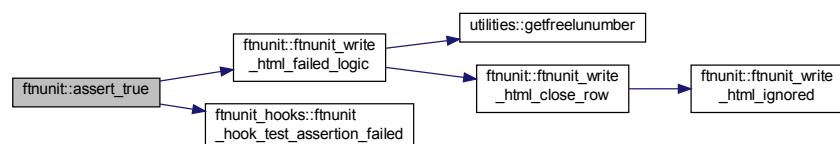
Here is the call graph for this function:



**7.11.1.17 subroutine, public ftnunit::assert\_true ( logical, intent(in) cond, character(len=\*), intent(in) text )**

Definition at line 530 of file ftnunit.f90.

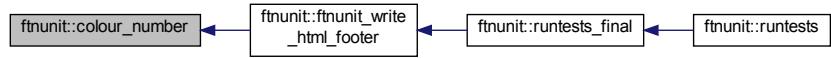
Here is the call graph for this function:



**7.11.1.18 subroutine ftnunit::colour\_number ( integer, intent(in) number, character(len=\*), intent(out) coloured, character(len=\*), intent(in) colour1, character(len=\*), intent(in), optional colour2, integer, intent(in), optional allowed ) [private]**

Definition at line 1567 of file ftnunit.f90.

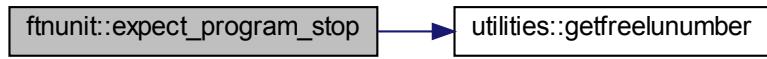
Here is the caller graph for this function:



#### 7.11.1.19 subroutine ftnunit::expect\_program\_stop( ) [private]

Definition at line 509 of file ftnunit.f90.

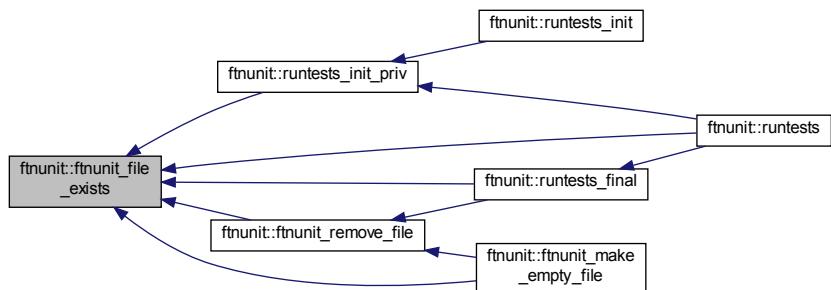
Here is the call graph for this function:



#### 7.11.1.20 logical function ftnunit::ftnunit\_file\_exists( character(len=\*) intent(in) filename ) [private]

Definition at line 1424 of file ftnunit.f90.

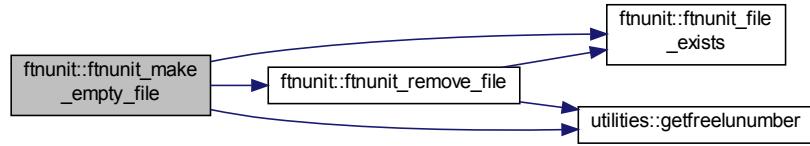
Here is the caller graph for this function:



#### 7.11.1.21 subroutine ftnunit::ftnunit\_make\_empty\_file( character(len=\*) intent(in) filename ) [private]

Definition at line 1461 of file ftnunit.f90.

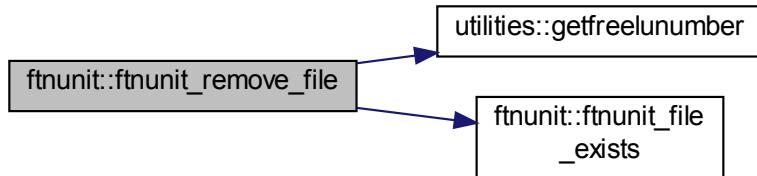
Here is the call graph for this function:



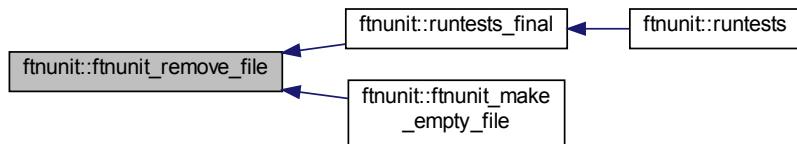
#### 7.11.1.22 subroutine `ftnunit::ftnunit_remove_file` ( character(len=\*)*filename* ) [private]

Definition at line 1435 of file `ftnunit.f90`.

Here is the call graph for this function:



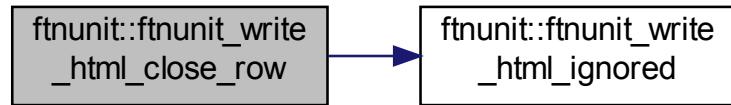
Here is the caller graph for this function:



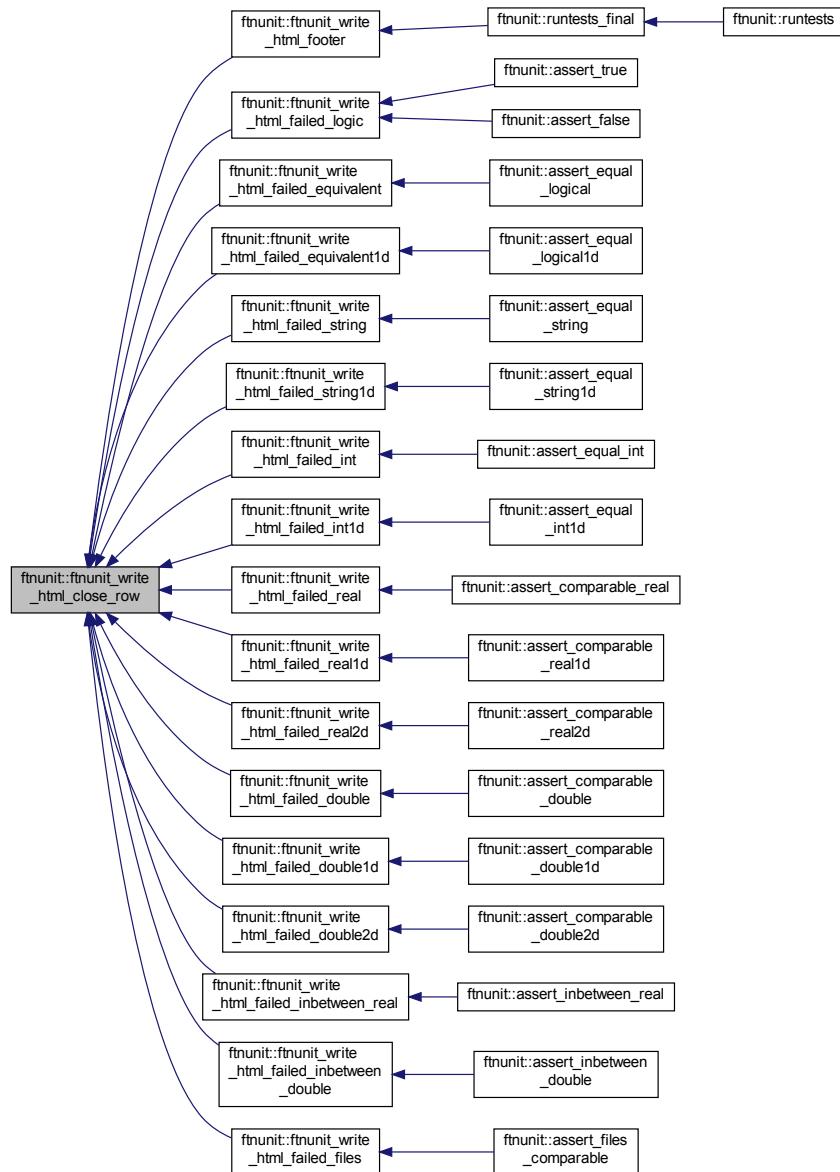
#### 7.11.1.23 subroutine `ftnunit::ftnunit_write_html_close_row` ( integer *lun* ) [private]

Definition at line 1708 of file `ftnunit.f90`.

Here is the call graph for this function:



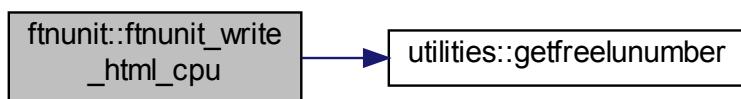
Here is the caller graph for this function:



7.11.1.24 subroutine `ftnunit::ftnunit_write_html_cpu ( real(kind=wp) cpu ) [private]`

Definition at line 1636 of file ftnunit.f90.

Here is the call graph for this function:



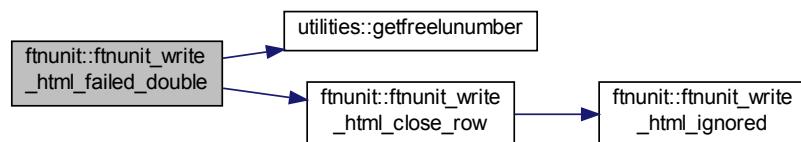
Here is the caller graph for this function:



7.11.1.25 subroutine `ftnunit::ftnunit_write_html_failed_double ( character(len=*) text, real(kind=dp) value1, real(kind=dp) value2 ) [private]`

Definition at line 2086 of file ftnunit.f90.

Here is the call graph for this function:



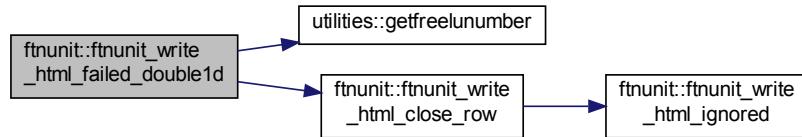
Here is the caller graph for this function:



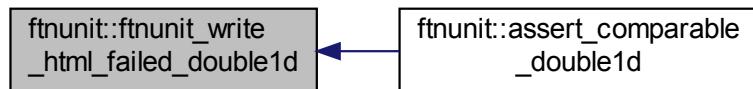
7.11.1.26 subroutine `ftnunit::ftnunit_write_html_failed_double1d` ( `character(len=*) text`, `integer idx`, `real(kind=dp) value1`,  
`real(kind=dp) value2`, `logical addtext` ) [private]

Definition at line 2120 of file ftnunit.f90.

Here is the call graph for this function:



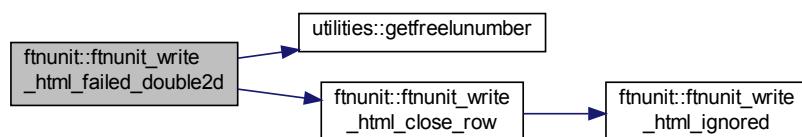
Here is the caller graph for this function:



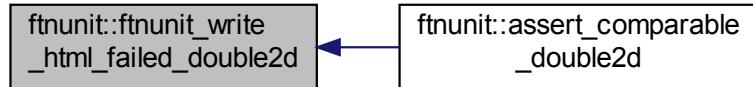
7.11.1.27 subroutine `ftnunit::ftnunit_write_html_failed_double2d` ( `character(len=*) text`, `integer idx1`, `integer idx2`,  
`real(kind=dp) value1`, `real(kind=dp) value2`, `logical addtext` ) [private]

Definition at line 2162 of file ftnunit.f90.

Here is the call graph for this function:



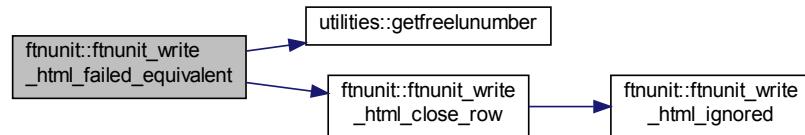
Here is the caller graph for this function:



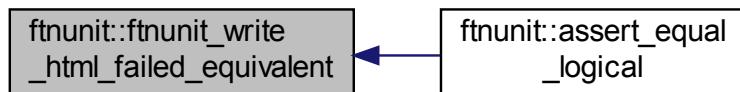
#### 7.11.1.28 subroutine ftnunit::ftnunit\_write\_html\_failed\_equivalent ( character(len=\*) text ) [private]

Definition at line 1759 of file ftnunit.f90.

Here is the call graph for this function:



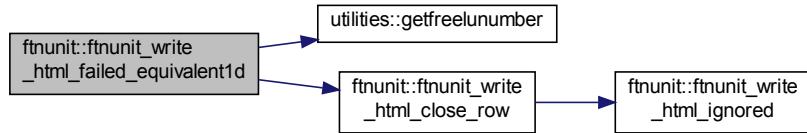
Here is the caller graph for this function:



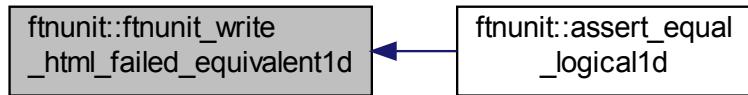
#### 7.11.1.29 subroutine ftnunit::ftnunit\_write\_html\_failed\_equivalent1d ( character(len=\*) text, integer idx, logical value1, logical value2, logical addtext ) [private]

Definition at line 1788 of file ftnunit.f90.

Here is the call graph for this function:



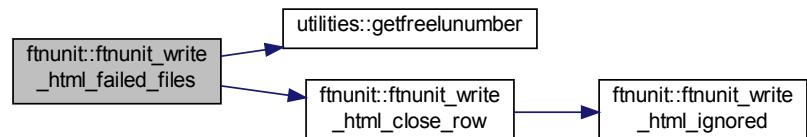
Here is the caller graph for this function:



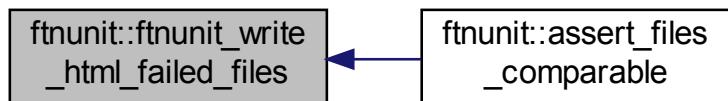
**7.11.1.30 subroutine ftnunit::ftnunit\_write\_html\_failed\_files ( character(len=\*) text, character(len=\*) string1, character(len=\*) string2, character(len=\*) string3 ) [private]**

Definition at line 2267 of file ftnunit.f90.

Here is the call graph for this function:



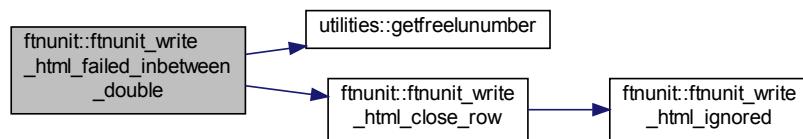
Here is the caller graph for this function:



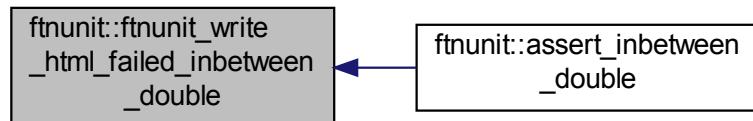
7.11.1.31 subroutine ftnunit::ftnunit\_write\_html\_failed\_inbetween\_double ( character(len=\*) text, real(kind=dp) value,  
real(kind=dp) vmin, real(kind=dp) vmax ) [private]

Definition at line 2236 of file ftnunit.f90.

Here is the call graph for this function:



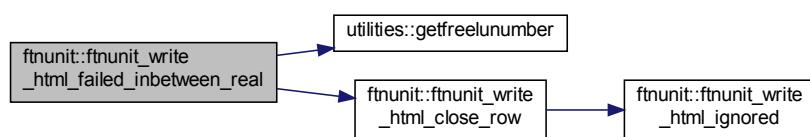
Here is the caller graph for this function:



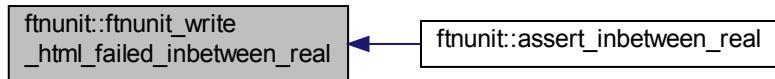
7.11.1.32 subroutine ftnunit::ftnunit\_write\_html\_failed\_inbetween\_real ( character(len=\*) text, real value, real vmin, real vmax  
) [private]

Definition at line 2204 of file ftnunit.f90.

Here is the call graph for this function:



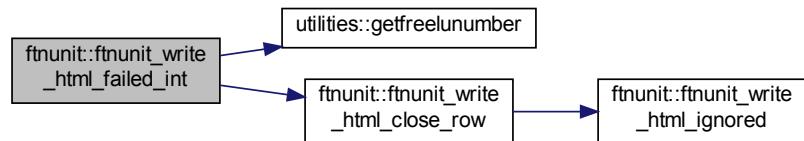
Here is the caller graph for this function:



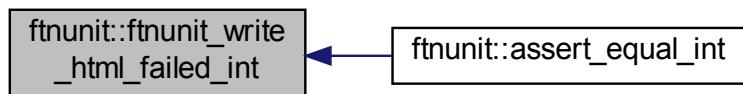
7.11.1.33 subroutine ftnunit::ftnunit\_write\_html\_failed\_int ( character(len=\*) text, integer value1, integer value2 )  
[private]

Definition at line 1899 of file ftnunit.f90.

Here is the call graph for this function:



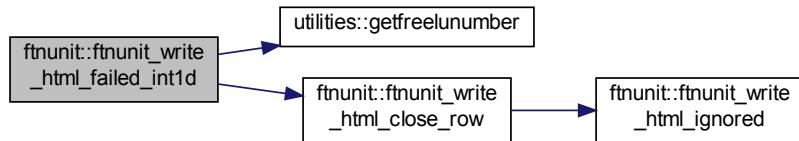
Here is the caller graph for this function:



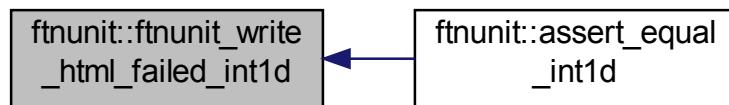
7.11.1.34 subroutine ftnunit::ftnunit\_write\_html\_failed\_int1d ( character(len=\*) text, integer idx, integer value1, integer value2, logical addtext ) [private]

Definition at line 1932 of file ftnunit.f90.

Here is the call graph for this function:



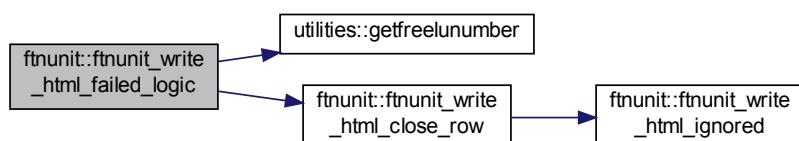
Here is the caller graph for this function:



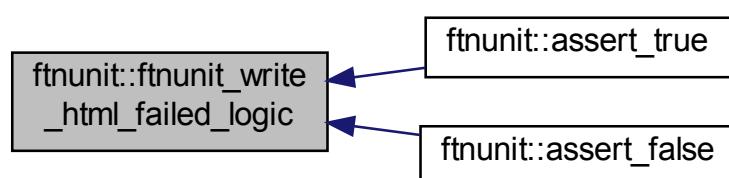
#### 7.11.1.35 subroutine ftnunit::ftnunit\_write\_html\_failed\_logic ( character(len=\*) text, logical expected ) [private]

Definition at line 1733 of file ftnunit.f90.

Here is the call graph for this function:



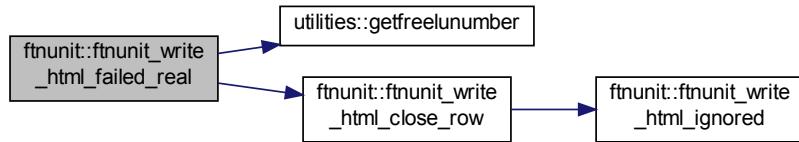
Here is the caller graph for this function:



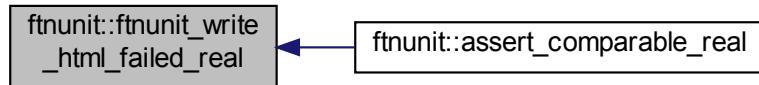
7.11.1.36 subroutine ftnunit::ftnunit\_write\_html\_failed\_real ( character(len=\*) *text*, real *value1*, real *value2* ) [private]

Definition at line 1972 of file ftnunit.f90.

Here is the call graph for this function:



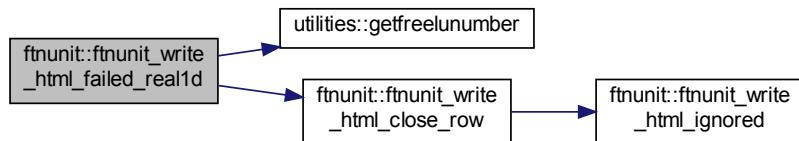
Here is the caller graph for this function:



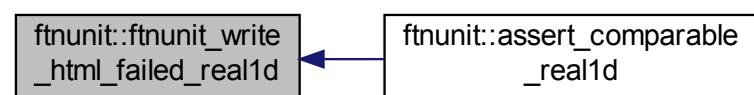
7.11.1.37 subroutine ftnunit::ftnunit\_write\_html\_failed\_real1d ( character(len=\*) *text*, integer *idx*, real *value1*, real *value2*, logical *addtext* ) [private]

Definition at line 2004 of file ftnunit.f90.

Here is the call graph for this function:



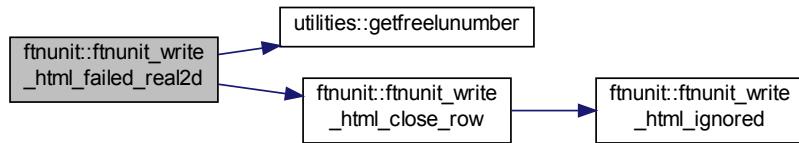
Here is the caller graph for this function:



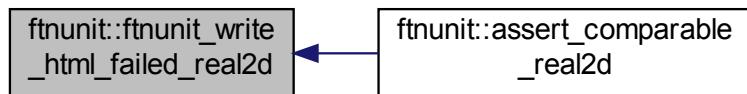
7.11.1.38 subroutine ftnunit::ftnunit\_write\_html\_failed\_real2d ( character(len=\*) *text*, integer *idx1*, integer *idx2*, real *value1*, real *value2*, logical *addtext* ) [private]

Definition at line 2046 of file ftnunit.f90.

Here is the call graph for this function:



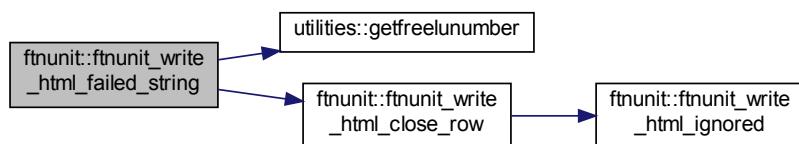
Here is the caller graph for this function:



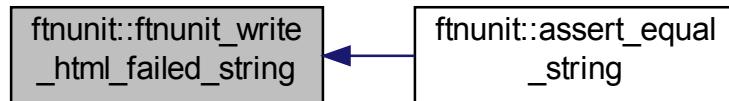
7.11.1.39 subroutine ftnunit::ftnunit\_write\_html\_failed\_string ( character(len=\*) *text*, character(len=\*) *value1*, character(len=\*) *value2* ) [private]

Definition at line 1827 of file ftnunit.f90.

Here is the call graph for this function:



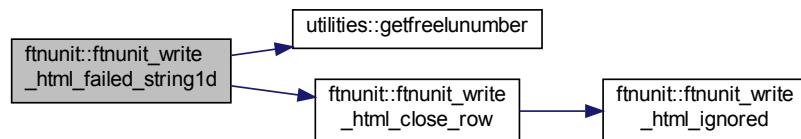
Here is the caller graph for this function:



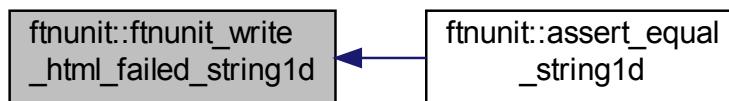
**7.11.1.40 subroutine ftnunit::ftnunit\_write\_html\_failed\_string1d ( character(len=\*) text, integer idx, character(len=\*) value1, character(len=\*) value2, logical addtext ) [private]**

Definition at line 1859 of file ftnunit.f90.

Here is the call graph for this function:



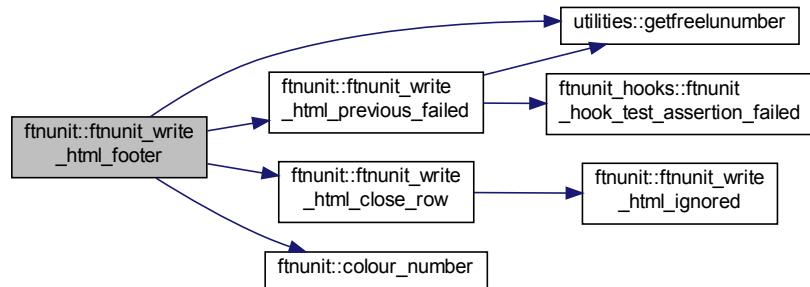
Here is the caller graph for this function:



**7.11.1.41 subroutine ftnunit::ftnunit\_write\_html\_footer ( ) [private]**

Definition at line 1525 of file ftnunit.f90.

Here is the call graph for this function:



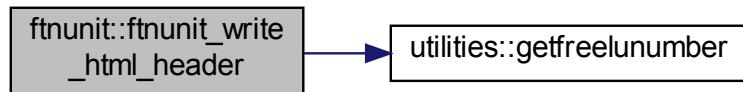
Here is the caller graph for this function:



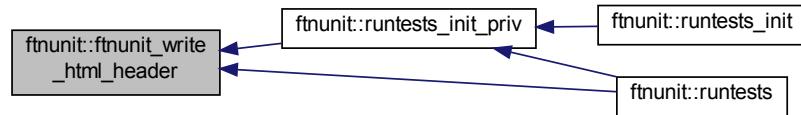
#### 7.11.1.42 subroutine `ftnunit::ftnunit_write_html_header( )` [private]

Definition at line 1486 of file `ftnunit.f90`.

Here is the call graph for this function:



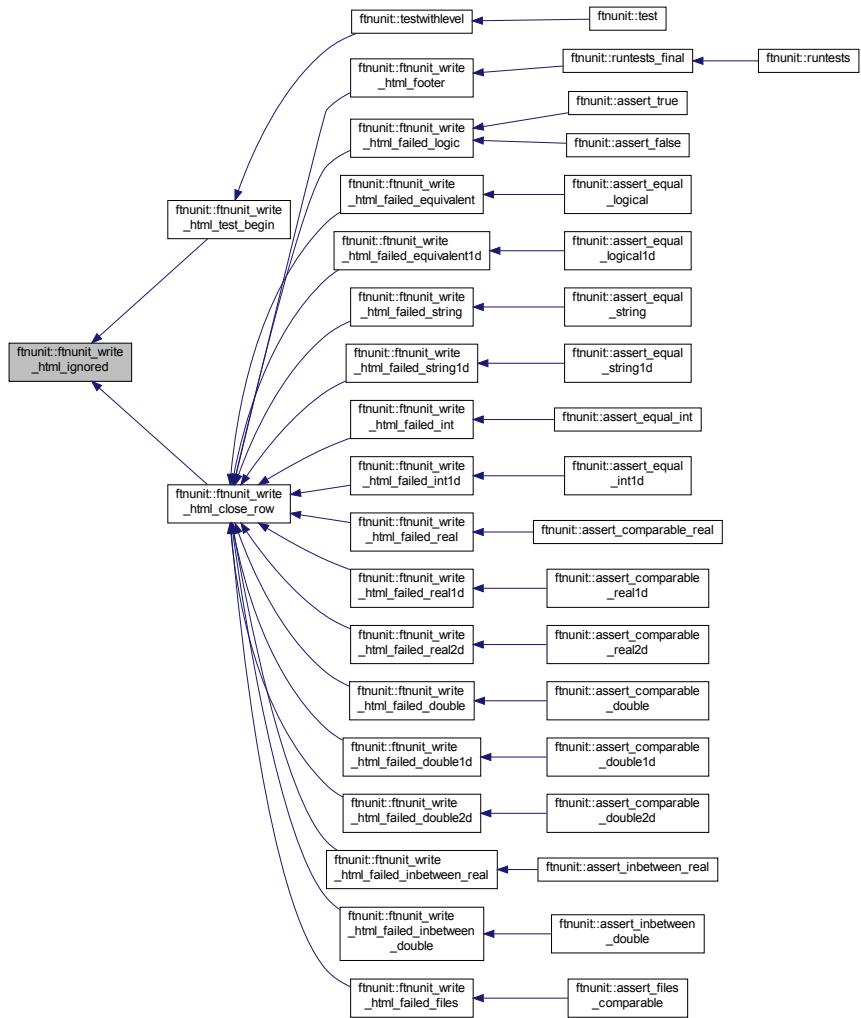
Here is the caller graph for this function:



## 7.11.1.43 subroutine ftnunit::ftnunit\_write\_html\_ignored ( integer, intent(in) lun ) [private]

Definition at line 1622 of file ftnunit.f90.

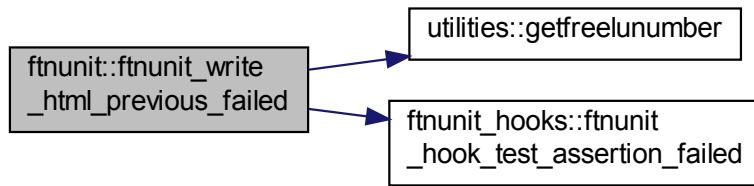
Here is the caller graph for this function:



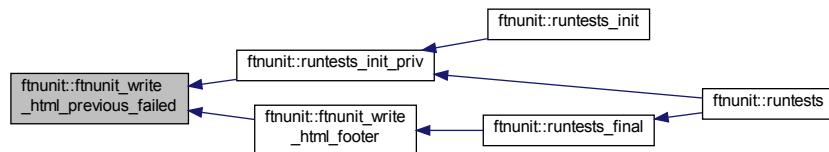
## 7.11.1.44 subroutine ftnunit::ftnunit\_write\_html\_previous\_failed ( )

Definition at line 1661 of file ftnunit.f90.

Here is the call graph for this function:



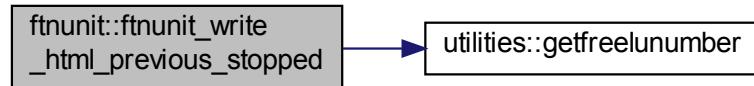
Here is the caller graph for this function:



#### 7.11.1.45 subroutine ftnunit::ftnunit\_write\_html\_previous\_stopped( ) [private]

Definition at line 1688 of file ftnunit.f90.

Here is the call graph for this function:



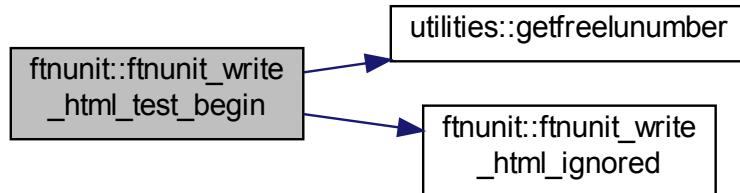
Here is the caller graph for this function:



7.11.1.46 subroutine `ftnunit::ftnunit_write_html_test_begin( character(len=*) text )` [private]

Definition at line 1593 of file ftnunit.f90.

Here is the call graph for this function:



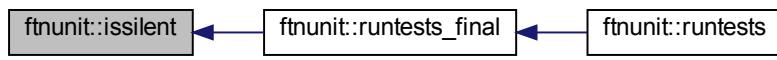
Here is the caller graph for this function:



7.11.1.47 logical function `ftnunit::issilent( )` [private]

Definition at line 411 of file ftnunit.f90.

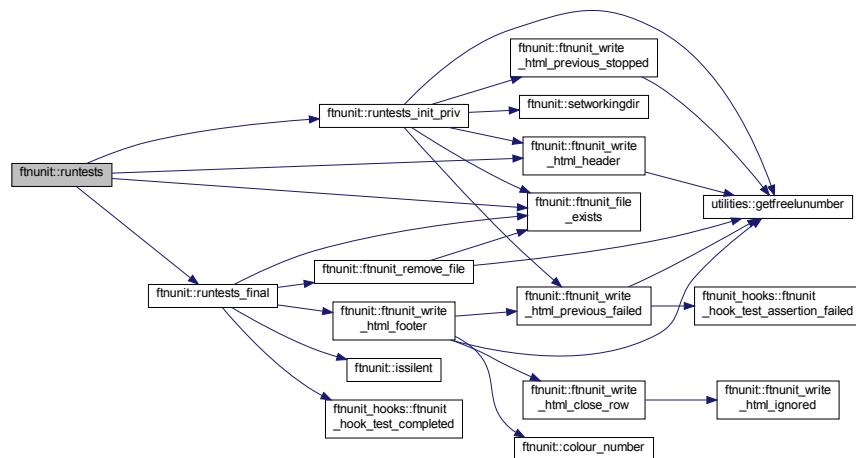
Here is the caller graph for this function:



7.11.1.48 subroutine, public `ftnunit::runtests( testproc )`

Definition at line 463 of file ftnunit.f90.

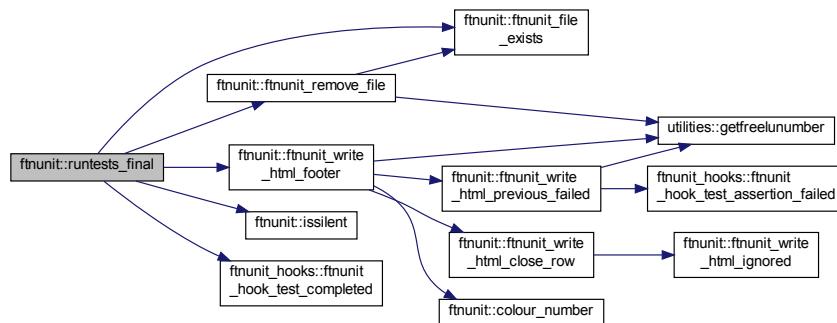
Here is the call graph for this function:



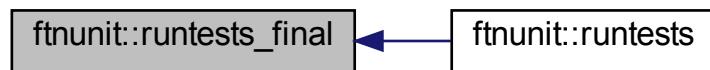
#### 7.11.1.49 subroutine, public ftnunit::runtests\_final ( logical, intent(in), optional stop )

Definition at line 383 of file ftnunit.f90.

Here is the call graph for this function:



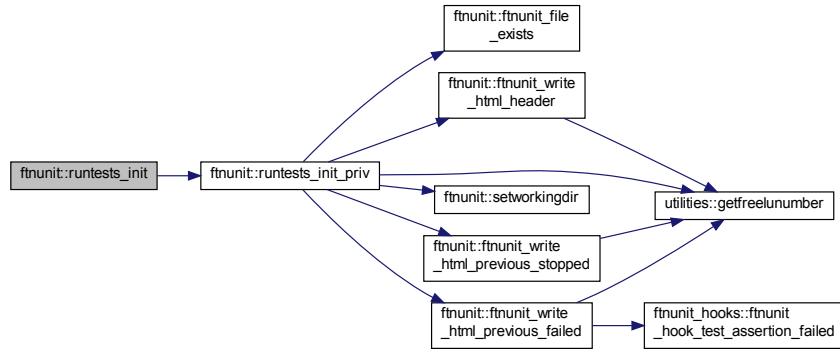
Here is the caller graph for this function:



## 7.11.1.50 subroutine, public ftnunit::runtests\_init( )

Definition at line 266 of file ftnunit.f90.

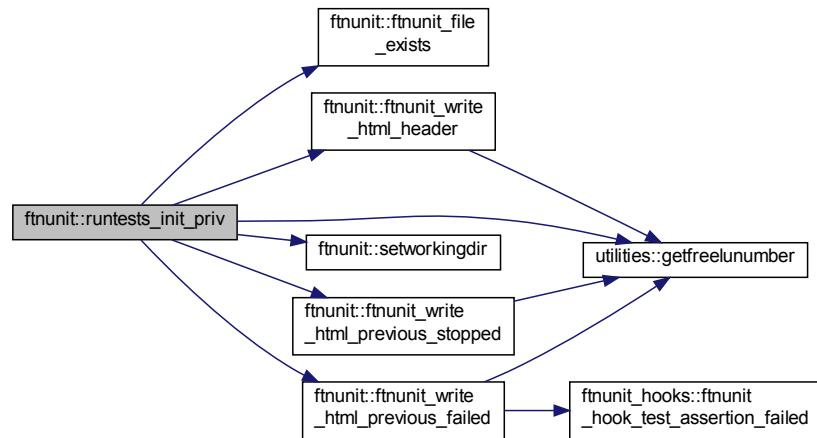
Here is the call graph for this function:



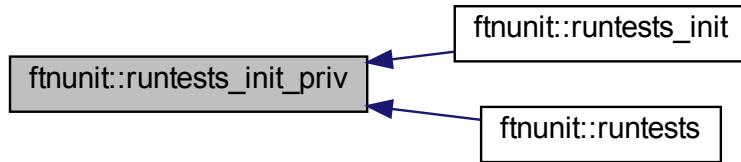
## 7.11.1.51 subroutine ftnunit::runtests\_init\_priv( ) [private]

Definition at line 278 of file ftnunit.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.11.1.52 subroutine, public ftnunit::setruntestlevel ( integer, intent(in) runtTestLevel )

Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.

##### Parameters

in	runttestlevel	set the new run testlevel to this value
----	---------------	---

Definition at line 112 of file ftnunit.f90.

#### 7.11.1.53 subroutine, public ftnunit::settesttitle ( character(len=\*), intent(in) message )

Set the title that is displayed in the report of the test run.

##### Parameters

in	message	the title to be set
----	---------	---------------------

Definition at line 120 of file ftnunit.f90.

#### 7.11.1.54 subroutine ftnunit::setworkingdir ( ) [private]

Definition at line 430 of file ftnunit.f90.

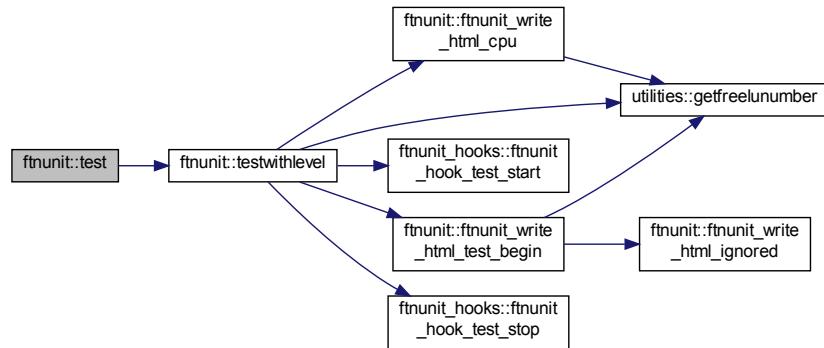
Here is the caller graph for this function:



#### 7.11.1.55 subroutine, public ftnunit::test ( external proc, character(len=\*) text, logical, intent(in), optional ignore )

Definition at line 131 of file ftnunit.f90.

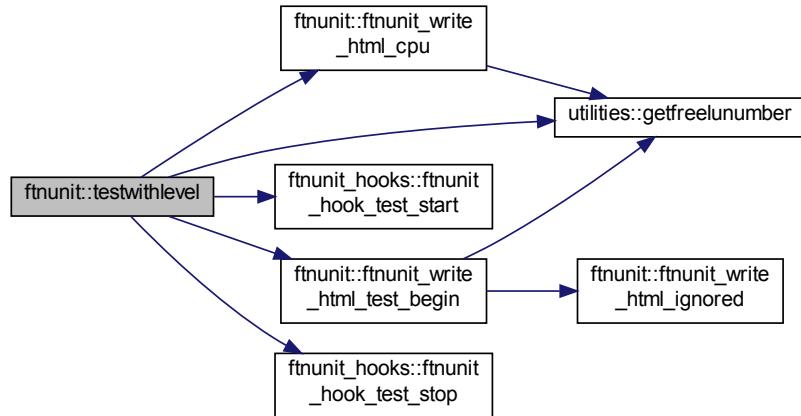
Here is the call graph for this function:



**7.11.1.56 subroutine, public ftnunit::testwithlevel ( external proc, character(len=\*) text, integer, intent(in) testlevel, logical, intent(in), optional ignore )**

Definition at line 145 of file ftnunit.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.11.2 Variable Documentation

7.11.2.1 logical, save ftnunit::call\_final = .true.

Definition at line 68 of file ftnunit.f90.

7.11.2.2 integer, parameter ftnunit::dp = kind(1.0d0)

Definition at line 54 of file ftnunit.f90.

7.11.2.3 integer, save ftnunit::failed\_asserts = 0

Definition at line 70 of file ftnunit.f90.

7.11.2.4 logical, save ftnunit::has\_run = .false.

Definition at line 71 of file ftnunit.f90.

7.11.2.5 character(len=20), save ftnunit::html\_file = 'ftnunit.html'

Definition at line 74 of file ftnunit.f90.

7.11.2.6 logical, save ftnunit::ignore\_previous\_test = .false.

Definition at line 73 of file ftnunit.f90.

7.11.2.7 character(len=32), save ftnunit::ignore\_reason

Definition at line 77 of file ftnunit.f90.

7.11.2.8 character(len=32), save ftnunit::ignore\_reason\_previous\_test = ''

Definition at line 78 of file ftnunit.f90.

7.11.2.9 logical, save ftnunit::ignore\_test = .false.

Definition at line 72 of file ftnunit.f90.

7.11.2.10 integer, save ftnunit::last\_test

Definition at line 59 of file ftnunit.f90.

7.11.2.11 integer, parameter ftnunit::mode\_all = 0

Definition at line 50 of file ftnunit.f90.

7.11.2.12 integer, parameter ftnunit::mode\_list = 2

Definition at line 52 of file ftnunit.f90.

7.11.2.13 **integer, parameter** ftnunit::mode\_single = 1

Definition at line 51 of file ftnunit.f90.

7.11.2.14 **integer, save** ftnunit::nofails

Definition at line 61 of file ftnunit.f90.

7.11.2.15 **integer, save** ftnunit::nofails\_prev

Definition at line 62 of file ftnunit.f90.

7.11.2.16 **integer, save** ftnunit::noruns

Definition at line 67 of file ftnunit.f90.

7.11.2.17 **integer, save** ftnunit::notests\_failed

Definition at line 64 of file ftnunit.f90.

7.11.2.18 **integer, save** ftnunit::notests\_ignored

Definition at line 65 of file ftnunit.f90.

7.11.2.19 **integer, save** ftnunit::notests\_run

Definition at line 63 of file ftnunit.f90.

7.11.2.20 **integer, save** ftnunit::notests\_work\_in\_progress

Definition at line 66 of file ftnunit.f90.

7.11.2.21 **logical, save** ftnunit::previous = .false.

Definition at line 69 of file ftnunit.f90.

7.11.2.22 **integer, save** ftnunit::run\_test\_level = 99

Definition at line 56 of file ftnunit.f90.

7.11.2.23 **integer, save** ftnunit::single\_test = -1

Definition at line 57 of file ftnunit.f90.

7.11.2.24 **procedure (proc), pointer, public** ftnunit::subptrtestinit => null()

Definition at line 44 of file ftnunit.f90.

7.11.2.25 procedure (proc), pointer, public ftnunit::subptrtestprograminit => null()

Definition at line 43 of file ftnunit.f90.

7.11.2.26 integer, save ftnunit::test\_mode = mode\_all

Definition at line 58 of file ftnunit.f90.

7.11.2.27 character(len=80), save ftnunit::testname

Definition at line 75 of file ftnunit.f90.

7.11.2.28 integer, save ftnunit::testno

Definition at line 60 of file ftnunit.f90.

7.11.2.29 character(len=80), save ftnunit::testtitle = 'Result of unit tests'

Definition at line 76 of file ftnunit.f90.

7.11.2.30 character(len=\*), parameter ftnunit::work\_in\_progress = 'work-in-progress'

Definition at line 79 of file ftnunit.f90.

## 7.12 ftnunit\_hooks Module Reference

### Functions/Subroutines

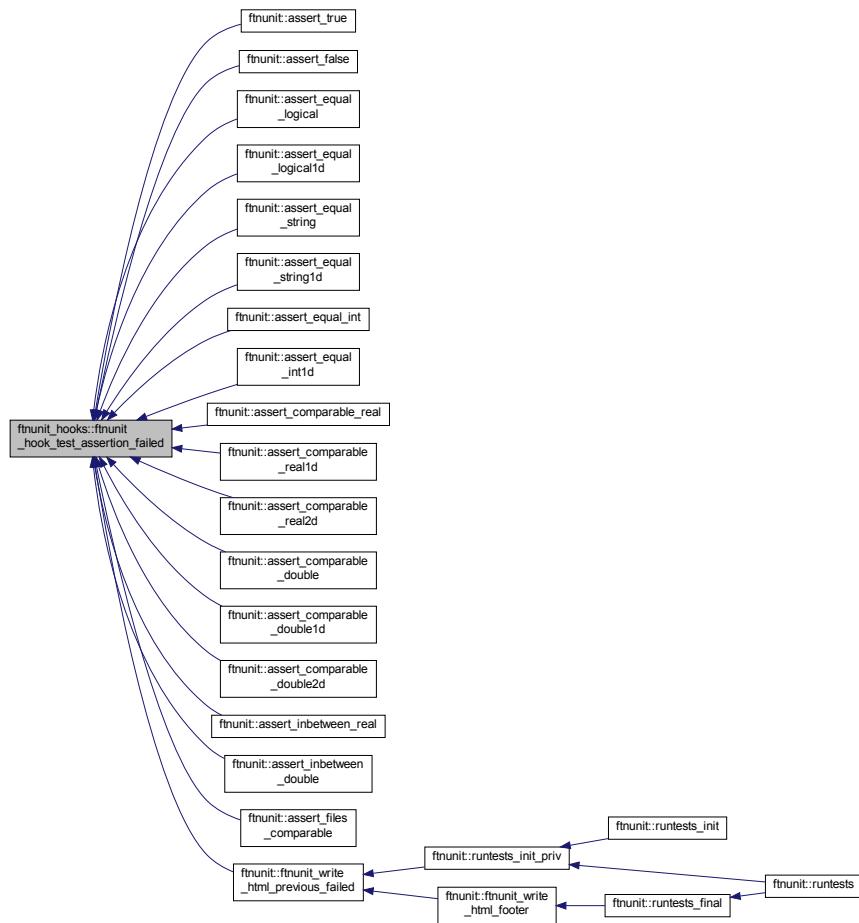
- subroutine [ftnunit\\_hook\\_test\\_start](#) (text)
- subroutine [ftnunit\\_hook\\_test\\_stop](#) (text)
- subroutine [ftnunit\\_hook\\_test\\_assertion\\_failed](#) (text, assert\_text, failure\_text)
- subroutine [ftnunit\\_hook\\_test\\_completed](#)

### 7.12.1 Function/Subroutine Documentation

7.12.1.1 subroutine ftnunit\_hooks::ftnunit\_hook\_test\_assertion\_failed ( character(len=\*) text, character(len=\*) assert\_text, character(len=\*) failure\_text )

Definition at line 57 of file ftnunit\_hooks.f90.

Here is the caller graph for this function:



### 7.12.1.2 subroutine ftnunit\_hooks::ftnunit\_hook\_test\_completed( )

Definition at line 71 of file ftnunit\_hooks.f90.

Here is the caller graph for this function:



### 7.12.1.3 subroutine ftnunit\_hooks::ftnunit\_hook\_test\_start( character(len=\*) text )

Definition at line 31 of file ftnunit\_hooks.f90.

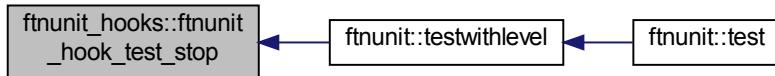
Here is the caller graph for this function:



#### 7.12.1.4 subroutine ftnunit\_hooks::ftnunit\_hook\_test\_stop ( character(len=\*) text )

Definition at line 43 of file ftnunit\_hooks.f90.

Here is the caller graph for this function:



## 7.13 general Module Reference

### 7.14 geometrymodulertoovertopping Module Reference

#### Functions/Subroutines

- subroutine, public [checkcrosssection](#) (psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, succes, errorMessage)
   
*checkCrossSection: check cross section*
- subroutine, public [initializegeometry](#) (psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, geometry, succes, errorMessage)
   
*initializeGeometry: initialize the geometry*
- subroutine, public [allocatevectorsgeometry](#) (nCoordinates, geometry)
   
*allocateVectorsGeometry: allocate the geometry vectors*
- subroutine, public [deallocategeometry](#) (geometry)
   
*deallocateGeometry: deallocate the geometry vectors*
- subroutine, public [calculatesegmentsslopes](#) (geometry, succes, errorMessage)
   
*calculateSegmentSlopes: calculate the segment slopes*
- subroutine, public [determinesegmenttypes](#) (geometry)
   
*determineSegmentTypes: determine the segment types*
- subroutine, public [copygeometry](#) (geometry, geometryCopy)
   
*copyGeometry: copy a geometry structure*
- subroutine, public [isequalgeometry](#) (geometry1, geometry2, succes, errorMessage)
   
*isEqualGeometry: are two geometries equal*
- subroutine, public [mergesequentialberms](#) (geometry, geometryMergedBerms, succes, errorMessage)

- mergeSequentialBerms: merge sequential berms*
- subroutine, public **adjustnonhorizontalberms** (geometry, geometryFlatBerms, succes, errorMessage)
  - adjustNonHorizontalBerms: adjust non-horizontal berms*
- subroutine, public **removeberms** (geometry, geometryNoBerms, succes, errorMessage)
  - removeBerms: remove berms*
- subroutine, public **removedikesegments** (geometry, index, geometryAdjusted, succes, errorMessage)
  - removeDikeSegments: remove dike segments*
- subroutine, public **splitcrosssection** (geometry, L0, NwideBerms, geometrysectionB, geometrysectionF, succes, errorMessage)
  - splitCrossSection: split a cross section*
- subroutine, public **calculatehorzlenghts** (geometry, yLower, yUpper, horzLengths, succes, errorMessage)
  - calculateHorzLengths: calculate horizontal lengths*
- subroutine, public **calculatehorzdistance** (geometry, yLower, yUpper, dx, succes, errorMessage)
  - calculateHorzDistance: calculate horizontal distance*
- subroutine, public **writecrosssection** (geometry, geometryName)
  - writeCrossSection: write a cross section*

### 7.14.1 Function/Subroutine Documentation

7.14.1.1 subroutine, public **geometrymodulertoovertopping::adjustnonhorizontalberms** ( type (tpgeometry), intent(in) *geometry*, type (tpgeometry), intent(out) *geometryFlatBerms*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

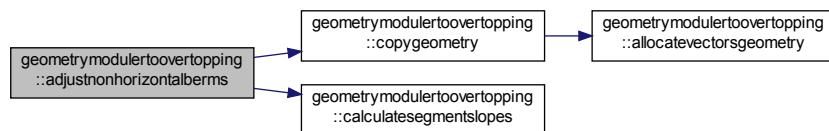
*adjustNonHorizontalBerms: adjust non-horizontal berms*

#### Parameters

in	<i>geometry</i>	structure with geometry data
out	<i>geometryflat-berms</i>	geometry data with horizontal berms
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 578 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.2 subroutine, public geometrymodulertoovertopping::allocatevectorsgeometry ( integer, intent(in) *nCoordinates*, type (tpgeometry), intent(inout) *geometry* )

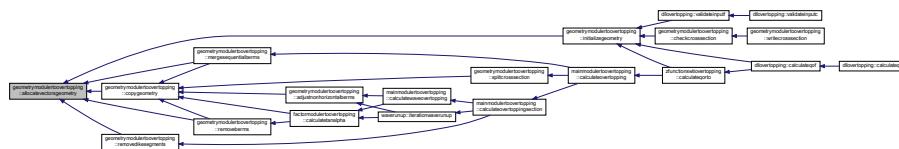
allocateVectorsGeometry: allocate the geometry vectors

### Parameters

in	<i>ncoordinates</i>	number of coordinates
in, out	<i>geometry</i>	structure with geometry data

Definition at line 199 of file geometryModuleRTOvertopping.f90.

Here is the caller graph for this function:



**7.14.1.3 subroutine, public geometrymodulertoovertopping::calculatehorzdistance ( type (tpgeometry), intent(in) *geometry*, real(wp), intent(in) *yLower*, real(wp), intent(in) *yUpper*, real(wp), intent(out) *dx*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )**

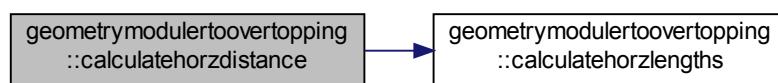
*calculateHorzDistance*: calculate horizontal distance

### Parameters

in	<i>geometry</i>	structure with geometry data
in	<i>ylower</i>	y-coordinate lower bound (m+NAP)
in	<i>yupper</i>	y-coordinate upper bound (m+NAP)
out	<i>dx</i>	horizontal distance between bounds (m)
out	<i>succes</i>	flag for succes
out	<i>errorMessage</i>	error message

Definition at line 1023 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.14.1.4 subroutine, public geometrymodulertoovertopping::calculatehorzlenghts ( type (tpgeometry), intent(in) *geometry*, real(wp), intent(in) *yLower*, real(wp), intent(in) *yUpper*, real(wp), dimension(geometry%ncoordinates-1), intent(out) *horzLengths*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )**

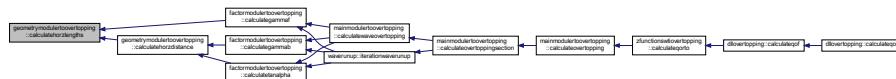
*calculateHorzLengths*: calculate horizontal lengths

## Parameters

in	<i>geometry</i>	structure with geometry data
in	<i>ylower</i>	y-coord. lower bound (m+NAP)
in	<i>yupper</i>	y-coord. upper bound (m+NAP)
out	<i>horzlenghts</i>	horizontal lengths segments (m)
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 927 of file geometryModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.14.1.5 subroutine, public **geometryModulertoOvertopping::calculatesegmentsslopes** ( type (tpgeometry), intent(inout) *geometry*, logical, intent(out) *succes*, character(len=\*) , intent(out) *errorMessage* )

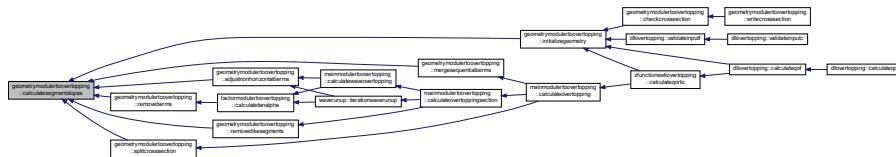
*calculateSegmentSlopes*: calculate the segment slopes

## Parameters

in, out	<i>geometry</i>	structure with geometry data
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 248 of file geometryModuleRTOvertopping.f90.

Here is the caller graph for this function:



7.14.1.6 subroutine, public **geometryModulertoOvertopping::checkcrosssection** ( real(wp), intent(in) *psi*, integer, intent(in) *nCoordinates*, real(wp), dimension (*ncoordinates*), intent(in) *xCoordinates*, real(wp), dimension (*ncoordinates*), intent(in) *yCoordinates*, real(wp), dimension(*ncoordinates*-1), intent(in) *roughnessFactors*, logical, intent(out) *succes*, character(len=\*) , intent(out) *errorMessage* )

*checkCrossSection*: check cross section

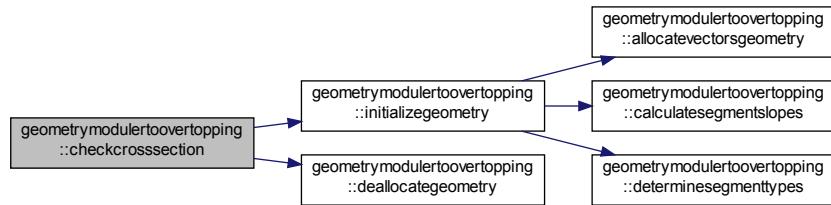
## Parameters

in	<i>psi</i>	dike normal (degree)
in	<i>ncoordinates</i>	number of coordinates
in	<i>xcoordinates</i>	x-coordinates (m)

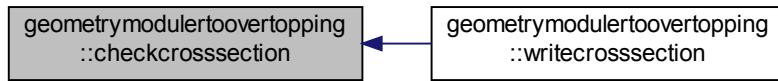
in	<i>ycoordinates</i>	y-coordinates (m+NAP)
in	<i>roughnessfactors</i>	roughness factors
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 34 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.7 subroutine, public geometrymodulertoovertopping::copygeometry ( type (tpgeometry), intent(in) *geometry*, type (tpgeometry), intent(inout) *geometryCopy* )

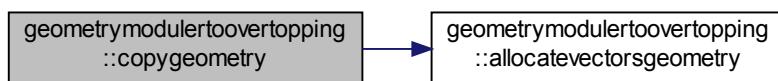
*copyGeometry*: copy a geometry structure

#### Parameters

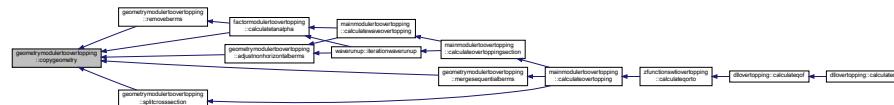
in	<i>geometry</i>	structure with geometry data
in, out	<i>geometryCopy</i>	structure with geometry data copy

Definition at line 330 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.14.1.8 subroutine, public geometryModuloToOvertopping::deallocateGeometry ( type (tpgeometry), intent(inout) geometry )

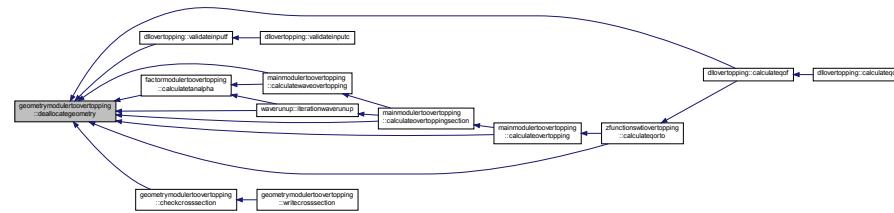
deallocateGeometry: deallocate the geometry vectors

Parameters

in, out	<i>geometry</i>	structure with geometry data
---------	-----------------	------------------------------

Definition at line 225 of file geometryModuleRTOvertopping.f90.

Here is the caller graph for this function:



#### 7.14.1.9 subroutine, public geometryModuloToOvertopping::determineSegmentTypes ( type (tpgeometry), intent(inout) geometry )

determineSegmentTypes: determine the segment types

Parameters

in, out	<i>geometry</i>	structure with geometry data
---------	-----------------	------------------------------

Definition at line 287 of file geometryModuleRTOvertopping.f90.

Here is the caller graph for this function:



#### 7.14.1.10 subroutine, public geometryModuloToOvertopping::initializeGeometry ( real(wp), intent(in) psi, integer, intent(in) nCoordinates, real(wp), dimension (ncoordinates), intent(in) xCoordinates, real(wp), dimension (ncoordinates), intent(in) yCoordinates, real(wp), dimension(ncoordinates-1), intent(in) roughnessFactors, type (tpgeometry), intent(out) geometry, logical, intent(out) succes, character(len=\*) intent(out) errorMessage )

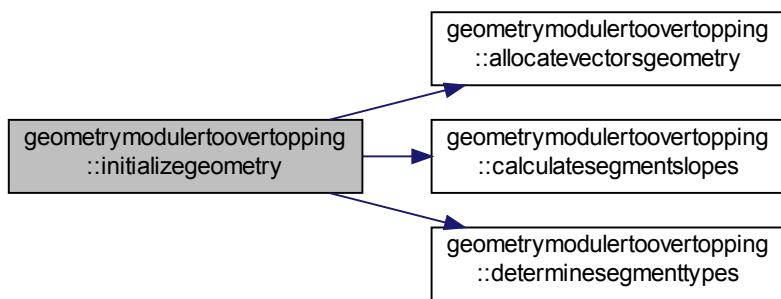
initializeGeometry: initialize the geometry

## Parameters

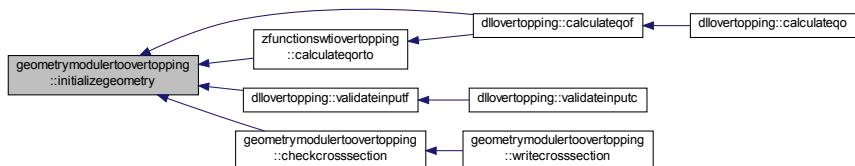
in	<i>psi</i>	dike normal (degree)
in	<i>ncoordinates</i>	number of coordinates
in	<i>xcoordinates</i>	x-coordinates (m)
in	<i>ycoordinates</i>	y-coordinates (m+NAP)
in	<i>roughnessfactors</i>	roughness factors
out	<i>geometry</i>	structure with geometry data
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 142 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.11 subroutine, public geometrymodulertoovertopping::isequalgeometry ( type(tpgeometry), intent(in) *geometry1*, type(tpgeometry), intent(in) *geometry2*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

isEqualGeometry: are two geometries equal

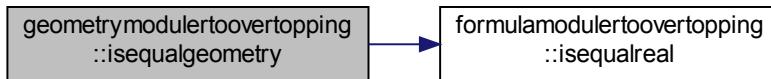
## Parameters

in	<i>geometry1</i>	structure with geometry data 1
in	<i>geometry2</i>	structure with geometry data 2
out	<i>succes</i>	flag for succes

out	<i>errormessage</i>	error message
-----	---------------------	---------------

Definition at line 377 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



7.14.1.12 subroutine, public geometrymodulertoovertopping::mergesequentialberms ( type (tpgeometry), intent(in) *geometry*, type (tpgeometry), intent(inout) *geometryMergedBerms*, logical, intent(out) *succes*, character(len=\*) , intent(out) *errorMessage* )

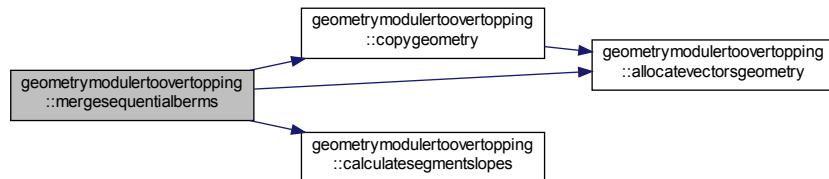
mergeSequentialBerms: merge sequential berms

#### Parameters

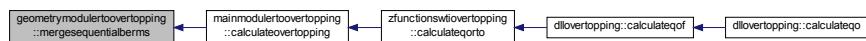
in	<i>geometry</i>	structure with geometry data
in, out	<i>geome- trymergedberms</i>	geometry data with merged sequential berms
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 471 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.13 subroutine, public geometrymodulertoovertopping::removeberms ( type (tpgeometry), intent(in) *geometry*, type (tpgeometry), intent(out) *geometryNoBerms*, logical, intent(out) *succes*, character(len=\*) , intent(out) *errorMessage* )

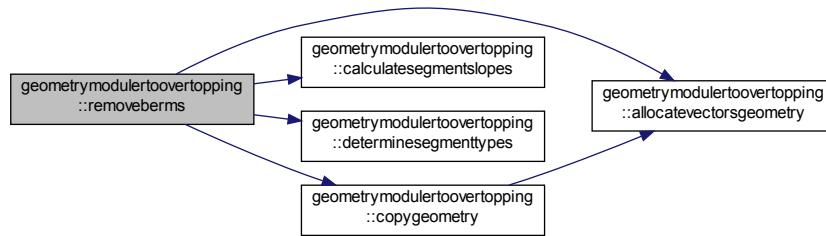
removeBerms: remove berms

## Parameters

in	<i>geometry</i>	structure with geometry data
out	<i>geome- trynoberms</i>	geometry data without berms
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 664 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.14.1.14 subroutine, public geometrymodulertoovertopping::removedikesegments ( type (tpgeometry), intent(in) *geometry*, integer, intent(in) *index*, type (tpgeometry), intent(out) *geometryAdjusted*, logical, intent(out) *succes*, character(len=\*), intent(out) *errormessage* )**

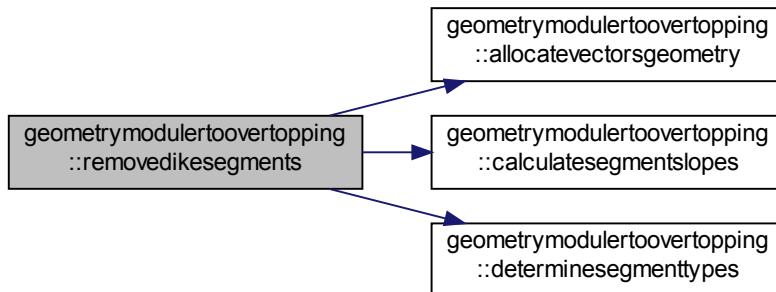
removeDikeSegments: remove dike segments

## Parameters

in	<i>geometry</i>	structure with geometry data
in	<i>index</i>	index starting point new cross section
out	<i>geometryad- justed</i>	geometry data with removed dike segments
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 761 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.14.1.15 subroutine, public geometrymodulertoovertopping::splitcrosssection ( type(tpgeometry), intent(in) *geometry*, real(wp), intent(in) *L0*, integer, intent(out) *NwideBerms*, type(tpgeometry), intent(out) *geometrysectionB*, type(tpgeometry), intent(out) *geometrysectionF*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

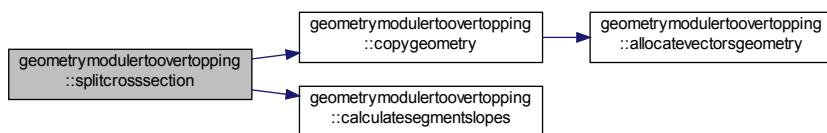
*splitCrossSection*: split a cross section

Parameters

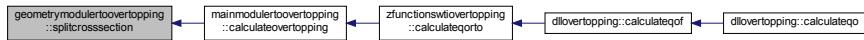
in	<i>geometry</i>	structure with geometry data
in	<i>L0</i>	wave length (m)
out	<i>nwideberms</i>	number of wide berms
out	<i>geometrysec-</i> <i>tionb</i>	geometry data with wide berms to ordinary berms
out	<i>geometrysec-</i> <i>tionf</i>	geometry data with wide berms to foreshores
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 825 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.14.1.16 subroutine, public geometrymodulertoovertopping::writecrosssection ( type (tpgeometry), intent(in) geometry, character(len=\*), intent(in) geometryName )**

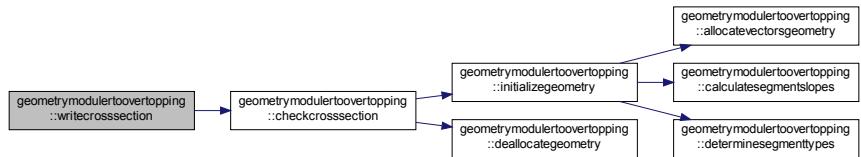
writeCrossSection: write a cross section

#### Parameters

in	geometry	structure with geometry data
in	geometryname	description of geometry data

Definition at line 1066 of file geometryModuleRTOvertopping.f90.

Here is the call graph for this function:



## 7.15 interpolationtriangular Module Reference

Module with function for triangular interpolation.

### Functions/Subroutines

- real(kind=wp) function [triangularinterpolation](#) (x1, y1, h1, x2, y2, h2, x3, y3, h3, x, y)

*Function for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.*

- real(kind=wp) function [datainterpolation](#) (var1, var2, weight, iscyclic)

### 7.15.1 Detailed Description

Module with function for triangular interpolation.

### 7.15.2 Function/Subroutine Documentation

**7.15.2.1 real(kind=wp) function interpolationtriangular::datainterpolation ( real(kind=wp), intent(in) var1, real(kind=wp), intent(in) var2, real(kind=wp), intent(in) weight, integer iscyclic )**

**Parameters**

in	<i>var1</i>	First input argument for in interpolation functions using a scalar weight value
in	<i>var2</i>	Second input argument for in interpolation functions using a scalar weight value
in	<i>weight</i>	Value for weighing two variables: 'weight' holds for var1 and '(1-weight)' holds for var2
	<i>iscyclic</i>	If the input is of cyclic character, then iscyclic = 1 (for instance: wave direction)

**Returns**

Result value after linear interpolation between var1 and var2 using weightvalue weight

Definition at line 111 of file interpolationTriangular.f90.

7.15.2.2 `real(kind=wp) function interpolationtriangular::triangularinterpolation ( real(kind=wp), intent(in) x1, real(kind=wp), intent(in) y1, real(kind=wp), intent(in) h1, real(kind=wp), intent(in) x2, real(kind=wp), intent(in) y2, real(kind=wp), intent(in) h2, real(kind=wp), intent(in) x3, real(kind=wp), intent(in) y3, real(kind=wp), intent(in) h3, real(kind=wp), intent(in) x, real(kind=wp), intent(in) y )`

Function for for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.

**Parameters**

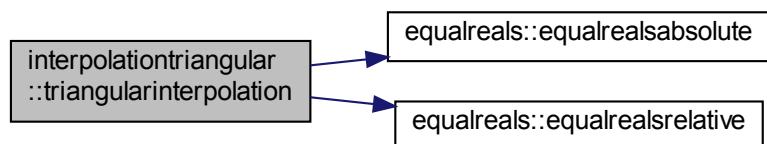
in	<i>x1</i>	x-coordinate station 1
in	<i>y1</i>	y-coordinate station 1
in	<i>h1</i>	Result value station 1 (water level)
in	<i>x2</i>	x-coordinate station 2
in	<i>y2</i>	y-coordinate station 2
in	<i>h2</i>	Result value station 2 (water level)
in	<i>x3</i>	x-coordinate station 3
in	<i>y3</i>	y-coordinate station 3
in	<i>h3</i>	Result value station 3 (water level)
in	<i>x</i>	x-coordinate point of examination
in	<i>y</i>	y-coordinate point of examination

**Returns**

Result value (water level) on point of examination after triangular interpolation

Definition at line 26 of file interpolationTriangular.f90.

Here is the call graph for this function:



## 7.16 mainmodulertoovertopping Module Reference

### Functions/Subroutines

- subroutine, public [calculateovertopping](#) (geometry, load, modelFactors, overtopping, succes, errorMessage)
 

*calculateOvertopping: calculate the overtopping*
- subroutine, public [calculateovertoppingsection](#) (geometry, h, Hm0, Tm\_10, L0, gammaBeta\_z, gammaBeta\_o, modelFactors, overtopping, succes, errorMessage)
 

*calculateOvertoppingSection: calculate the overtopping for a section*
- subroutine, public [calculatewaveovertopping](#) (geometry, h, Hm0, Tm\_10, z2, gammaBeta\_o, modelFactors, Qo, succes, errorMessage)
 

*calculateWaveOvertopping: calculate wave overtopping*
- subroutine [calculateovertoppingnegativefreeboard](#) (load, geometry, overtopping, succes, errorMessage)
 

*calculateOvertoppingNegativeFreeboard: calculate overtopping in case of negative freeboard*
- subroutine, public [interpolateresultssections](#) (geometry, L0, NwideBerms, overtoppingB, overtoppingF, overtopping, succes, errorMessage)
 

*interpolateResultsSections: interpolate results for split cross sections*
- subroutine, public [checkinputdata](#) (geometry, load, modelFactors, succes, errorMessage)
 

*checkInputdata: check the input data*
- subroutine, public [checkmodelfactors](#) (modelFactors, succes, errorMessage)
 

*checkModelFactors: check the input data*
- subroutine, public [convertovertoppinginput](#) (modelFactors, success, errorMessage)
 

*convertOvertoppingInput: convert the model factors from C-like to Fortran*

### 7.16.1 Function/Subroutine Documentation

7.16.1.1 subroutine, public mainmodulertoovertopping::calculateovertopping ( type (tpgeometry), intent(in) *geometry*, type (tpload), intent(in) *load*, type (tpovertoppinginput), intent(in) *modelFactors*, type (tpovertopping), intent(out) *overtopping*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )

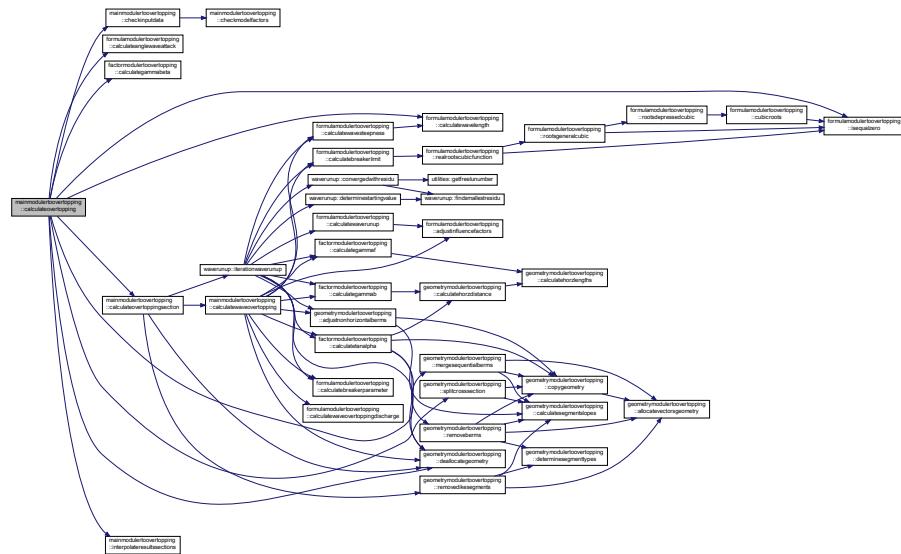
*calculateOvertopping: calculate the overtopping*

#### Parameters

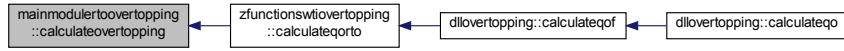
in	<i>geometry</i>	structure with geometry data
in	<i>load</i>	structure with load parameters
in	<i>modelFactors</i>	structure with model factors
out	<i>overtopping</i>	structure with overtopping results
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 36 of file mainModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.16.1.2 subroutine mainmodulertoovertopping::calculateovertoppingnegativefreeboard ( type (tpload), intent(in) *load*, type (tpgeometry), intent(in) *geometry*, type (tpovertopping), intent(inout) *overtopping*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* ) [private]**

*calculateOvertoppingNegativeFreeboard*: calculate overtopping in case of negative freeboard

#### Parameters

<i>in</i>	<i>geometry</i>	structure with geometry data
<i>in</i>	<i>load</i>	structure with load parameters
<i>in, out</i>	<i>overtopping</i>	structure with overtopping results
<i>out</i>	<i>succes</i>	flag for succes
<i>out</i>	<i>errormessage</i>	error message

Definition at line 478 of file mainModuleRTOvertopping.f90.

**7.16.1.3 subroutine, public mainmodulertoovertopping::calculateovertoppingsection ( type (tpgeometry), intent(in) *geometry*, real(wp), intent(in) *h*, real(wp), intent(in) *Hm0*, real(wp), intent(in) *Tm\_10*, real(wp), intent(in) *L0*, real(wp), intent(inout) *gammaBeta\_z*, real(wp), intent(inout) *gammaBeta\_o*, type (tpovertoppinginput), intent(in) *modelFactors*, type (tpovertopping), intent(out) *overtopping*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )**

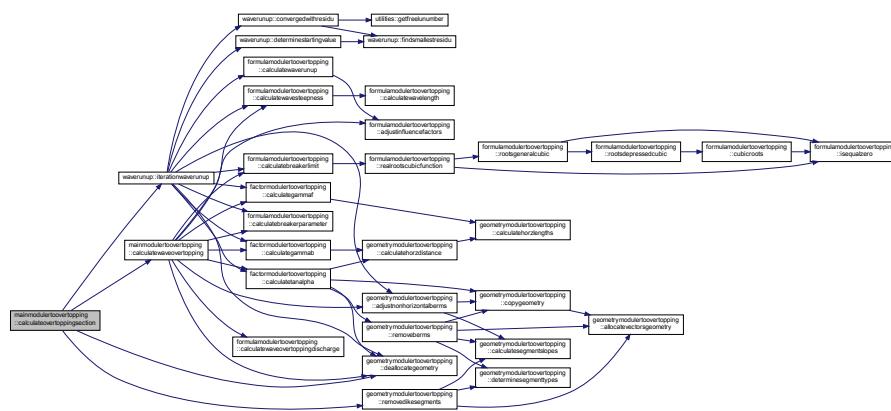
*calculateOvertoppingSection*: calculate the overtopping for a section

## Parameters

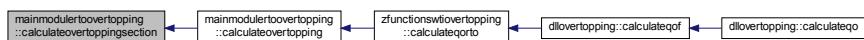
in	<i>geometry</i>	structure with geometry data
in	<i>h</i>	local water level (m+NAP)
in	<i>hm0</i>	significant wave height (m)
in	<i>tm_10</i>	spectral wave period (s)
in	<i>l0</i>	wave length (m)
in,out	<i>gammabeta_z</i>	influence angle wave attack wave run-up
in,out	<i>gammabeta_o</i>	influence angle wave attack overtopping
in	<i>modelfactors</i>	structure with model factors
out	<i>overtopping</i>	structure with overtopping results
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 164 of file mainModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.16.1.4 subroutine, public mainmodulertoovertopping::calculatewaveovertopping ( type (tpgeometry), intent(in) *geometry*, real(*wp*), intent(in) *h*, real(*wp*), intent(in) *Hm0*, real(*wp*), intent(in) *Tm\_10*, real(*wp*), intent(in) *z2*, real(*wp*), intent(inout) *gammaBeta\_o*, type (tpovertoppinginput), intent(in) *modelFactors*, real(*wp*), intent(out) *Qo*, logical, intent(out) *succes*, character(len=\*), intent(out) *errorMessage* )**

**calculateWaveOvertopping:** calculate wave overtopping

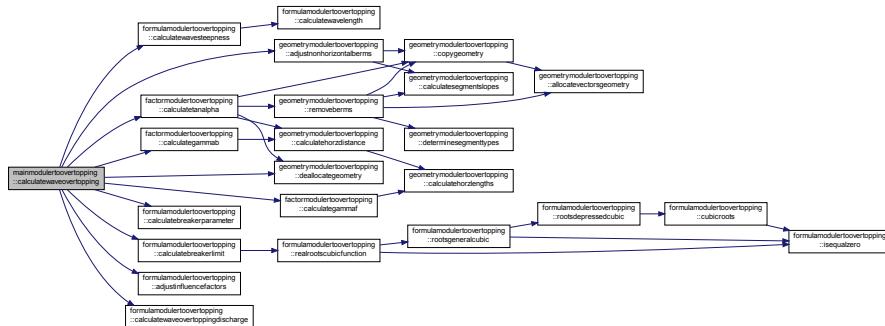
## Parameters

in	<i>geometry</i>	structure with geometry data
in	<i>h</i>	local water level (m+NAP)
in	<i>hm0</i>	significant wave height (m)

in	<i>tm_10</i>	spectral wave period (s)
in	<i>z2</i>	2% wave run-up (m)
in,out	<i>gammabeta_o</i>	influence angle wave attack overtopping
in	<i>modelfactors</i>	structure with model factors
out	<i>qo</i>	wave overtopping discharge (m <sup>3</sup> /m per s)
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 390 of file mainModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.16.1.5 subroutine, public mainmodulerotovertopping::checkinputdata ( type (tpgeometry), intent(in) *geometry*, type (tupload), intent(in) *load*, type (tpovertoppinginput), intent(in) *modelFactors*, logical, intent(out) *succes*, character(len=\*) *errorMessage* )**

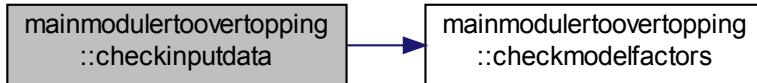
*checkInputdata*: check the input data

#### Parameters

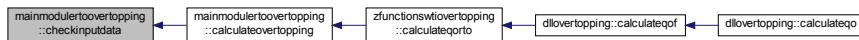
in	<i>geometry</i>	structure with geometry data
in	<i>load</i>	structure with load parameters
in	<i>modelfactors</i>	structure with model factors
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 595 of file mainModuleRTOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.16.1.6 subroutine, public mainmodulertoovertopping::checkmodelfactors ( type (t<sub>povertoppinginput</sub>), intent(in) *modelFactors*, logical, intent(out) *succes*, character(len=\*) , intent(out) *errorMessage* )**

checkModelFactors: check the input data

#### Parameters

in	<i>modelfactors</i>	structure with model factors
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 649 of file mainModuleRTOvertopping.f90.

Here is the caller graph for this function:



**7.16.1.7 subroutine, public mainmodulertoovertopping::convertovertoppinginput ( type (t<sub>povertoppinginput</sub>), intent(inout) *modelFactors*, logical, intent(out) *success*, character(len=\*) , intent(inout) *errorMessage* )**

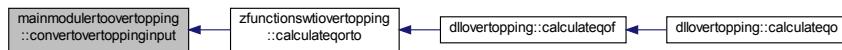
convertOvertoppingInput: convert the model factors from C-like to Fortran

#### Parameters

in, out	<i>modelfactors</i>	model factors and other input for overtopping
out	<i>success</i>	flag for success
in, out	<i>errormessage</i>	error message; only set when not successful

Definition at line 744 of file mainModuleRTOvertopping.f90.

Here is the caller graph for this function:



**7.16.1.8 subroutine, public mainmodulertoovertopping::interpolateResultsSections ( type (tpgeometry), intent(in) geometry, real(wp), intent(in) L0, integer, intent(in) NwideBerms, type (tpovertopping), intent(in) overtoppingB, type (tpovertopping), intent(in) overtoppingF, type (tpovertopping), intent(out) overtopping, logical, intent(out) succes, character(len=\*), intent(out) errorMessage )**

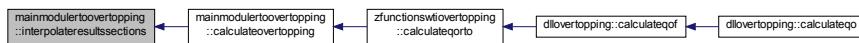
interpolateResultsSections: interpolate results for split cross sections

#### Parameters

in	geometry	structure with geometry data
in	L0	wave length (m)
in	nwideberms	number of wide berms
in	overtoppingb	structure with overtopping results ordinary berms
in	overtoppingf	structure with overtopping results foreshores
out	overtopping	structure with combined overtopping results
out	succes	flag for succes
out	errormessage	error message

Definition at line 514 of file mainModuleRTOvertopping.f90.

Here is the caller graph for this function:



## 7.17 modulelogging Module Reference

### Data Types

- type [tlogging](#)

*TLogging: structure for steering the logging.*

### Variables

- integer, parameter [maxfilenameLength](#) = 256  
*maximum length of filename*
- type([tlogging](#)) [currentlogging](#)  
*copy of argument logging*

### 7.17.1 Variable Documentation

#### 7.17.1.1 type([tlogging](#)) [modulelogging::currentlogging](#)

*copy of argument logging*

Definition at line 21 of file ModuleLogging.f90.

7.17.1.2 integer, parameter modulelogging::maxfilenamelen gth = 256

maximum length of filename

Definition at line 13 of file ModuleLogging.f90.

## 7.18 overtoppinginterface Module Reference

### Data Types

- type [overtoppinggeometrytype](#)
- type [overtoppinggeometrytypef](#)
- type [tpprofilecoordinate](#)

### Variables

- integer, parameter, public [varmodelfactorcriticalovertopping](#) = 8  
*Model factor critical overtopping.*

#### 7.18.1 Variable Documentation

7.18.1.1 integer, parameter, public overtoppinginterface::varmodel factorcriticalovertopping = 8

Model factor critical overtopping.

Definition at line 17 of file overtoppingInterface.f90.

## 7.19 physics Module Reference

## 7.20 physics\_breakwater Module Reference

Module with routines for breakwater.

### Functions/Subroutines

- subroutine, public [calculatebreakwater](#) (breakwaterType, heightBreakwater, waterLevel, Hs)  
*Function to transform the wave height for the influence of an occupant breakwater.*

#### 7.20.1 Detailed Description

Module with routines for breakwater.

#### 7.20.2 Function/Subroutine Documentation

7.20.2.1 subroutine, public physics\_breakwater::calculatebreakwater ( integer, intent(in) *breakwaterType*, real (kind=wp), intent(in) *heightBreakwater*, real (kind=wp), intent(in) *waterLevel*, real (kind=wp), intent(inout) *Hs* )

Function to transform the wave height for the influence of an occupant breakwater.

This subroutine performs a transformation on the wave height, due to breakwater construction which breaks waves. The breakwater formulation is based on the formula of Goda (1969) and Seelig (1979). Reference is made to the report "Rekenregels voor waterbouwkundig ontwerpen" of RWS Directie Sluizen en Stuwen, Utrecht, 1990.

Determine the shape factors (alpha and beta) depending on the breakwater type (caisson, vertical wall, rubble mound) Compute first the transmittance coefficient and afterwards the reduced wave height.

#### Parameters

in	<i>breakwatertype</i>	breakwater type
in	<i>heightbreakwater</i>	height breakwater [m + NAP]
in	<i>waterlevel</i>	water level [m + NAP]
in,out	<i>hs</i>	wave height [m]

Definition at line 35 of file breakwater.f90.

## 7.21 physics\_breakwatertypesenumerations Module Reference

Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing↔Enumerations.CS.

#### Variables

- integer, parameter `breakwaternotpresent` = 0  
*no breakwater*
- integer, parameter `breakwatercaisson` = 1  
*breakwater type is caisson*
- integer, parameter `breakwaterverticalwall` = 2  
*breakwater type is vertical wall*
- integer, parameter `breakwaterrubblemound` = 3  
*breakwater type is rubble mound 1:1.5*

### 7.21.1 Detailed Description

Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing↔Enumerations.CS.

### 7.21.2 Variable Documentation

#### 7.21.2.1 integer, parameter `physics_breakwatertypesenumerations::breakwatercaisson` = 1

*breakwater type is caisson*

Definition at line 20 of file breakwaterEnumerations.f90.

#### 7.21.2.2 integer, parameter `physics_breakwatertypesenumerations::breakwaternotpresent` = 0

*no breakwater*

Definition at line 17 of file breakwaterEnumerations.f90.

7.21.2.3 integer, parameter physics\_breakwatertypesenumerations::breakwaterrubblemound = 3

breakwater type is rubble mound 1:1.5

Definition at line 26 of file breakwaterEnumerations.f90.

7.21.2.4 integer, parameter physics\_breakwatertypesenumerations::breakwaterverticalwall = 2

breakwater type is vertical wall

Definition at line 23 of file breakwaterEnumerations.f90.

## 7.22 physics\_bretschneider Module Reference

Module with routines for bretschneider.

### Data Types

- type [tpbretschneider](#)

### Functions/Subroutines

- subroutine, public [calculatwavebretschneider](#) (bretschneider, Hs, Ts)

*Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)*

#### 7.22.1 Detailed Description

Module with routines for bretschneider.

#### 7.22.2 Function/Subroutine Documentation

7.22.2.1 subroutine, public physics\_bretschneider::calculatwavebretschneider ( type(tpbretschneider) bretschneider, real(kind = wp), intent(out) Hs, real(kind = wp), intent(out) Ts )

Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)

##### Parameters

	<i>bretschneider</i>	Bretschneider structure with necessary variables stored
out	<i>hs</i>	Significant wave height
out	<i>ts</i>	Significant wave period

Definition at line 37 of file bretschneider.f90.

## 7.23 physics\_foreland Module Reference

wrapper for foreland dll !!

## Functions/Subroutines

- subroutine, public **physics\_calculateforeland** (retval, hDamType, hDamHeight, hAlpha, hFc, hInvalid, hDimHm0, hHm0, hTp, hWlev, hIncomingWaveAngle, hDikeNormal, hDimX, hX, hMinStepSize, hBottomLevel, hRatioDepth, hLogging, hLoggingFileName, hHm0Dike, hTpDike, hRefractedWaveAngleDike, hMessage)

*wrapper for foreland calculation in DynamicLib-DaF.dll*

## Variables

- integer, parameter **message\_length** =1000

### 7.23.1 Detailed Description

wrapper for foreland dll !!

### 7.23.2 Function/Subroutine Documentation

- 7.23.2.1 subroutine, public **physics\_foreland::physics\_calculateforeland** ( integer, intent(out) *retval*, integer, intent(in) *hDamType*, real(kind=wp), intent(in) *hDamHeight*, real(kind=wp), intent(in) *hAlpha*, real(kind=wp), intent(in) *hFc*, real(kind=wp), intent(in) *hInvalid*, integer, intent(in) *hDimHm0*, real(kind=wp), dimension(1:hdimhm0), intent(in) *hHm0*, real(kind=wp), dimension(1:hdimhm0), intent(in) *hTp*, real(kind=wp), dimension(1:hdimhm0), intent(in) *hWlev*, real(kind=wp), dimension(1:hdimhm0), intent(in) *hIncomingWaveAngle*, real(kind=wp), intent(in) *hDikeNormal*, integer, intent(in) *hDimX*, real(kind=wp), dimension(1:hdimx), intent(in) *hX*, real(kind=wp), intent(in) *hMinStepSize*, real(kind=wp), dimension(1:hdimx), intent(in) *hBottomLevel*, real(kind=wp), intent(in) *hRatioDepth*, integer, intent(in) *hLogging*, character(len=\*), intent(in) *hLoggingFileName*, real(kind=wp), dimension(1:hdimhm0), intent(out) *hHm0Dike*, real(kind=wp), dimension(1:hdimhm0), intent(out) *hTpDike*, real(kind=wp), dimension(1:hdimhm0), intent(out) *hRefractedWaveAngleDike*, character(len=message\_length) *hMessage* )

wrapper for foreland calculation in DynamicLib-DaF.dll

#### Parameters

in	<i>hdamtype</i>	Dam types 1,2,3 [-]
in	<i>hdamheight</i>	Height of the dam [m]
in	<i>hdimhm0</i>	Total number of hydrodynamic parameters [-]
in	<i>hdimx</i>	Total number of locations [-]
in	<i>halpha</i>	Baldock coefficient [-]
in	<i>hfc</i>	Collins coefficient [-]
in	<i>hinvalid</i>	Value to be used for marking output as invalid [-]
in	<i>hhm0</i>	Array(1D) containing Hm0 values at different time steps [m]
in	<i>htp</i>	Array(1D) containing Tp values at different time steps [s]
in	<i>hwlev</i>	Array(1D) containing Water level values at different time steps [m]
in	<i>hincoming-waveangle</i>	Array(1D) containing incoming wave angles values at different time steps [degrees N]
in	<i>hdikenormal</i>	DikeNormal [degrees N]
in	<i>hx</i>	Array(1D) containing the x values of the profile (x(1) equals the location of the swan output point) [m]
in	<i>hminstepsize</i>	Minimum grid step size [m]
in	<i>hbottomlevel</i>	Array(1D) containing the depth values at the prescribed profile coordinates x(i) [-]

in	<i>hratiodepth</i>	Ratio between the wave height Hm0 (check with Alfons) and waterdepth above which breaking occurs.
in	<i>hlogging</i>	Switch which activates logging mechanism [-]
in	<i>hloggingfilename</i>	Name of the log file.
out	<i>hhm0dike</i>	Array(1D) containing the calculate Hm0 values at the toe of the dike for the provided time steps [m]
out	<i>htpdike</i>	Array(1D) containing the Tp values at the toe of the dike for the provided time steps [s]
out	<i>hrefracted-waveangledike</i>	Array(1D) containing the refracted angles [degrees N]
out	<i>retval</i>	Return 0 or -1. The first if no errors occur the later otherwise.
	<i>hmessage</i>	String containing an error message in case een error occurs [-]

Definition at line 27 of file foreland.f90.

### 7.23.3 Variable Documentation

#### 7.23.3.1 integer, parameter physics\_foreland::message\_length =1000

Definition at line 19 of file foreland.f90.

## 7.24 physics\_wavereduction Module Reference

Module with routines for wave reduction.

### Data Types

- type [tpwavereduction](#)

### Functions/Subroutines

- subroutine, public [calculatewavereduction](#) (wavereduction, errorCode, Hs)

*Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)*

### 7.24.1 Detailed Description

Module with routines for wave reduction.

### 7.24.2 Function/Subroutine Documentation

#### 7.24.2.1 subroutine, public physics\_wavereduction::calculatewavereduction ( type(tpwavereduction) *wavereduction*, integer, intent(out) *errorCode*, real(kind = wp), intent(out) *Hs* )

*Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)*

### Parameters

	<i>wavereduction</i>	Definition of wavereduction input parameters
out	<i>errorcode</i>	errorCode; 0 = succes; -1 = failure
out	<i>hs</i>	wave height

Definition at line 37 of file waveReduction.f90.

## 7.25 precision Module Reference

Module with parameters for KINDs and mathematical constants.

### Variables

- integer, parameter **sp** = kind(1.0)
 

*Single precision.*
- integer, parameter **dp** = kind(1.0d0)
 

*Double precision.*
- integer, parameter **wp** = **dp**

*Working precision.*
- real(**wp**), parameter **almostzero** = 1.0d-30
 

*Definition of zero for the machine.*
- real(**wp**), parameter **almostinf** = huge(1.0d0)
 

*Very large value as limit.*
- real(**wp**), parameter **rholimit** = 0.99999d0
 

*Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.*
- real(**wp**), parameter **betalimit** = 0.99999999d0
 

*Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCAST.*
- real(**wp**), parameter **pi** = 3.141592653589793238462643383279502884197\_wp
 

*Circle constant.*
- real(**wp**), parameter **emconstant** = 0.57721566490153286060651209008240243104215933593992\_wp
 

*Euler Mascheroni constant.*
- real(**wp**), parameter **gravityconstant** = 9.80665\_wp
 

*Gravity constant for the Netherlands.*
- integer, parameter **shortmax** = 32
- integer, parameter **longmax** = 255

### 7.25.1 Detailed Description

Module with parameters for KINDs and mathematical constants.

### 7.25.2 Variable Documentation

#### 7.25.2.1 real(wp), parameter precision::almostinf = huge(1.0d0)

Very large value as limit.

Definition at line 30 of file precision.f90.

#### 7.25.2.2 real(wp), parameter precision::almostzero = 1.0d-30

Definition of zero for the machine.

Definition at line 29 of file precision.f90.

**7.25.2.3 real(wp), parameter precision::betalimit = 0.99999999d0**

Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCAST.

Definition at line 32 of file precision.f90.

**7.25.2.4 integer, parameter precision::dp = kind(1.0d0)**

Double precision.

Definition at line 20 of file precision.f90.

**7.25.2.5 real(wp), parameter precision::emconstant = 0.57721566490153286060651209008240243104215933593992\_wp**

Euler Mascheroni constant.

Definition at line 38 of file precision.f90.

**7.25.2.6 real(wp), parameter precision::gravityconstant = 9.80665\_wp**

Gravity constant for the Netherlands.

Definition at line 39 of file precision.f90.

**7.25.2.7 integer, parameter precision::longmax = 255**

Definition at line 45 of file precision.f90.

**7.25.2.8 real(wp), parameter precision::pi = 3.141592653589793238462643383279502884197\_wp**

Circle constant.

Definition at line 37 of file precision.f90.

**7.25.2.9 real(wp), parameter precision::rholimit = 0.99999d0**

Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.

Definition at line 31 of file precision.f90.

**7.25.2.10 integer, parameter precision::shortmax = 32**

Definition at line 44 of file precision.f90.

**7.25.2.11 integer, parameter precision::sp = kind(1.0)**

Single precision.

Definition at line 19 of file precision.f90.

**7.25.2.12 integer, parameter precision::wp = dp**

Working precision.

Definition at line 25 of file precision.f90.

## 7.26 `typedefinitionsrovertopping` Module Reference

### Data Types

- type `tpgeometry`  
`tpGeometry: structure with geometry data`
- type `tpload`  
`tpLoad: structure with load parameters`
- type `tpovertopping`  
`tpOvertopping: structure with overtopping results`
- type `tpovertoppinginput`  
`OvertoppingModelFactors: C-structure with model factors.`

### Variables

- real(wp), parameter `xdiff_min` = 2.0d-2  
`minimal value distance between x-coordinates (m)`
- real(wp), parameter `margindiff` = 1.0d-14  
`margin for minimal distance (m)`
- real(wp), parameter `berm_min` = 0.0d0  
`minimal value gradient berm segment`
- real(wp), parameter `berm_max` = 1.0d0/15  
`maximal value gradient berm segment`
- real(wp), parameter `slope_min` = 1.0d0/8  
`minimal value gradient slope segment`
- real(wp), parameter `slope_max` = 1.0d0  
`maximal value gradient slope segment`
- real(wp), parameter `margingrad` = 0.0025d0  
`margin for minimal and maximal gradients`
- real(wp), parameter `rfactor_min` = 0.5d0  
`minimal value roughness factor dike segments`
- real(wp), parameter `rfactor_max` = 1.0d0  
`maximal value roughness factor dike segments`
- real(wp), parameter `mz2_min` = 0.0d0  
`minimal value model factor of 2% runup height`
- real(wp), parameter `mz2_max` = huge(mz2\_max)  
`maximal value model factor of 2% runup height`
- real(wp), parameter `frunup1_min` = 0.0d0  
`minimal value model factor 1 for wave run-up`
- real(wp), parameter `frunup1_max` = huge(fRunup1\_max)  
`maximal value model factor 1 for wave run-up`
- real(wp), parameter `frunup2_min` = 0.0d0  
`minimal value model factor 2 for wave run-up`
- real(wp), parameter `frunup2_max` = huge(fRunup2\_max)  
`maximal value model factor 2 for wave run-up`
- real(wp), parameter `frunup3_min` = 0.0d0  
`minimal value model factor 3 for wave run-up`
- real(wp), parameter `frunup3_max` = huge(fRunup3\_max)  
`maximal value model factor 3 for wave run-up`
- real(wp), parameter `fb_min` = 0.0d0

- real(wp), parameter **fb\_max** = huge(fB\_max)
  - minimal value model factor for breaking waves*
- real(wp), parameter **fn\_min** = 0.0d0
  - minimal value model factor for non-breaking waves*
- real(wp), parameter **fn\_max** = huge(fN\_max)
  - maximal value model factor for non-breaking waves*
- real(wp), parameter **fs\_min** = 0.0d0
  - minimal value model factor for shallow waves*
- real(wp), parameter **fs\_max** = huge(fS\_max)
  - maximal value model factor for shallow waves*
- real(wp), parameter **foreshore\_min** = 0.3d0
  - minimal value reduction factor foreshore*
- real(wp), parameter **foreshore\_max** = 1.0d0
  - maximal value reduction factor foreshore*
- integer, parameter **z2\_iter\_max** = 100
  - maximal number of iterations for calculation z2*
- real(wp), parameter **z2\_margin** = 0.001d0
  - margin for convergence criterium calculation z2*

## 7.26.1 Variable Documentation

### 7.26.1.1 real(wp), parameter typedefinitionsrtoovertopping::berm\_max = 1.0d0/15

maximal value gradient berm segment

Definition at line 67 of file typeDefinitionsRTOvertopping.f90.

### 7.26.1.2 real(wp), parameter typedefinitionsrtoovertopping::berm\_min = 0.0d0

minimal value gradient berm segment

Definition at line 66 of file typeDefinitionsRTOvertopping.f90.

### 7.26.1.3 real(wp), parameter typedefinitionsrtoovertopping::fb\_max = huge(fB\_max)

maximal value model factor for breaking waves

Definition at line 82 of file typeDefinitionsRTOvertopping.f90.

### 7.26.1.4 real(wp), parameter typedefinitionsrtoovertopping::fb\_min = 0.0d0

minimal value model factor for breaking waves

Definition at line 81 of file typeDefinitionsRTOvertopping.f90.

### 7.26.1.5 real(wp), parameter typedefinitionsrtoovertopping::fn\_max = huge(fN\_max)

maximal value model factor for non-breaking waves

Definition at line 84 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.6 real(wp), parameter typedefinitionsrtovertopping::fn\_min = 0.0d0**

minimal value model factor for non-breaking waves

Definition at line 83 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.7 real(wp), parameter typedefinitionsrtovertopping::foreshore\_max = 1.0d0**

maximal value reduction factor foreshore

Definition at line 88 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.8 real(wp), parameter typedefinitionsrtovertopping::foreshore\_min = 0.3d0**

minimal value reduction factor foreshore

Definition at line 87 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.9 real(wp), parameter typedefinitionsrtovertopping::frunup1\_max = huge(fRunup1\_max)**

maximal value model factor 1 for wave run-up

Definition at line 76 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.10 real(wp), parameter typedefinitionsrtovertopping::frunup1\_min = 0.0d0**

minimal value model factor 1 for wave run-up

Definition at line 75 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.11 real(wp), parameter typedefinitionsrtovertopping::frunup2\_max = huge(fRunup2\_max)**

maximal value model factor 2 for wave run-up

Definition at line 78 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.12 real(wp), parameter typedefinitionsrtovertopping::frunup2\_min = 0.0d0**

minimal value model factor 2 for wave run-up

Definition at line 77 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.13 real(wp), parameter typedefinitionsrtovertopping::frunup3\_max = huge(fRunup3\_max)**

maximal value model factor 3 for wave run-up

Definition at line 80 of file typeDefinitionsRTOvertopping.f90.

**7.26.1.14 real(wp), parameter typedefinitionsrtovertopping::frunup3\_min = 0.0d0**

minimal value model factor 3 for wave run-up

Definition at line 79 of file typeDefinitionsRTOvertopping.f90.

7.26.1.15 real(wp), parameter typedefinitionsrtoovertopping::fs\_max = huge(fs\_max)

maximal value model factor for shallow waves

Definition at line 86 of file typeDefinitionsRTOovertopping.f90.

7.26.1.16 real(wp), parameter typedefinitionsrtoovertopping::fs\_min = 0.0d0

minimal value model factor for shallow waves

Definition at line 85 of file typeDefinitionsRTOovertopping.f90.

7.26.1.17 real(wp), parameter typedefinitionsrtoovertopping::margindiff = 1.0d-14

margin for minimal distance (m)

Definition at line 65 of file typeDefinitionsRTOovertopping.f90.

7.26.1.18 real(wp), parameter typedefinitionsrtoovertopping::margingrad = 0.0025d0

margin for minimal and maximal gradients

Definition at line 70 of file typeDefinitionsRTOovertopping.f90.

7.26.1.19 real(wp), parameter typedefinitionsrtoovertopping::mz2\_max = huge(mz2\_max)

maximal value model factor of 2% runup height

Definition at line 74 of file typeDefinitionsRTOovertopping.f90.

7.26.1.20 real(wp), parameter typedefinitionsrtoovertopping::mz2\_min = 0.0d0

minimal value model factor of 2% runup height

Definition at line 73 of file typeDefinitionsRTOovertopping.f90.

7.26.1.21 real(wp), parameter typedefinitionsrtoovertopping::rfactor\_max = 1.0d0

maximal value roughness factor dike segments

Definition at line 72 of file typeDefinitionsRTOovertopping.f90.

7.26.1.22 real(wp), parameter typedefinitionsrtoovertopping::rfactor\_min = 0.5d0

minimal value roughness factor dike segments

Definition at line 71 of file typeDefinitionsRTOovertopping.f90.

7.26.1.23 real(wp), parameter typedefinitionsrtoovertopping::slope\_max = 1.0d0

maximal value gradient slope segment

Definition at line 69 of file typeDefinitionsRTOovertopping.f90.

7.26.1.24 real(wp), parameter typedefinitionsrtovertopping::slope\_min = 1.0d0/8

minimal value gradient slope segment

Definition at line 68 of file typeDefinitionsRTOvertopping.f90.

7.26.1.25 real(wp), parameter typedefinitionsrtovertopping::xdiff\_min = 2.0d-2

minimal value distance between x-coordinates (m)

Definition at line 64 of file typeDefinitionsRTOvertopping.f90.

7.26.1.26 integer, parameter typedefinitionsrtovertopping::z2\_iter\_max = 100

maximal number of iterations for calculation z2

Definition at line 89 of file typeDefinitionsRTOvertopping.f90.

7.26.1.27 real(wp), parameter typedefinitionsrtovertopping::z2\_margin = 0.001d0

margin for convergence criterium calculation z2

Definition at line 90 of file typeDefinitionsRTOvertopping.f90.

## 7.27 utilities Module Reference

Module with general utility routines.

### Functions/Subroutines

- subroutine [getfreelunumber](#) (lun)
 

*getFreeLuNumber Routine to find a free LU-number*
- pure character(len(str)) function [to\\_lower](#) (str)
 

*to\_lower Changes a string to lower case*
- pure character(len(str)) function [to\\_upper](#) (str)
 

*to\_upper Changes a string to upper case*

### Variables

- character(26), parameter, private [cap](#) = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
- character(26), parameter, private [low](#) = 'abcdefghijklmnopqrstuvwxyz'

#### 7.27.1 Detailed Description

Module with general utility routines.

#### 7.27.2 Function/Subroutine Documentation

7.27.2.1 subroutine [utilities::getfreelunumber](#) ( integer, intent(out) lun )

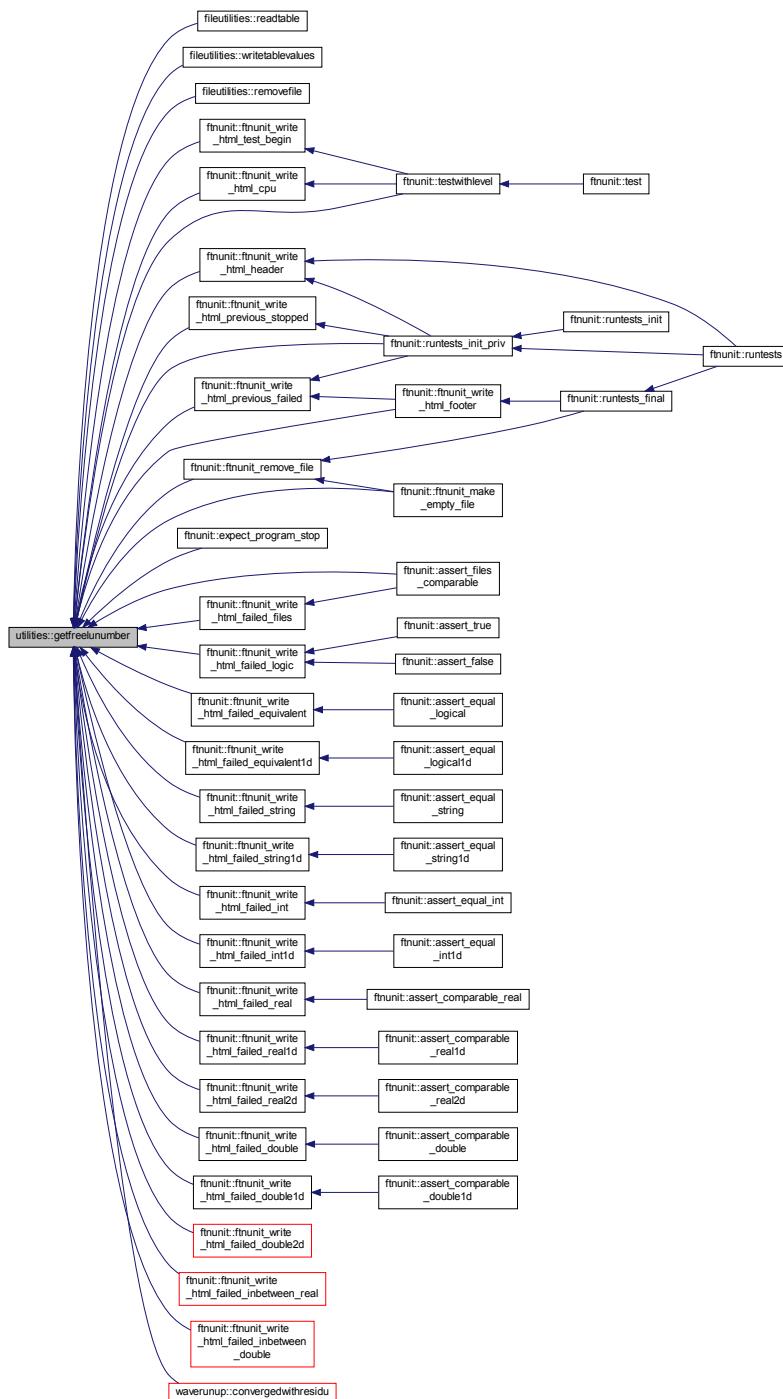
getFreeLuNumber Routine to find a free LU-number

## Parameters

out		lun	LU-number to use
-----	--	-----	------------------

Definition at line 28 of file utilities.f90.

Here is the caller graph for this function:



### 7.27.2.2 pure character(len(str)) function utilities::to\_lower ( character(\*), intent(in) str )

to\_lower Changes a string to lower case

Definition at line 58 of file utilities.f90.

### 7.27.2.3 pure character(len(str)) function utilities::to\_upper ( character(\*), intent(in) str )

to\_upper Changes a string to upper case

Definition at line 78 of file utilities.f90.

## 7.27.3 Variable Documentation

### 7.27.3.1 character(26), parameter, private utilities::cap = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

Definition at line 15 of file utilities.f90.

### 7.27.3.2 character(26), parameter, private utilities::low = 'abcdefghijklmnopqrstuvwxyz'

Definition at line 16 of file utilities.f90.

## 7.28 vectorutilities Module Reference

Module with vector utility functions.

### Functions/Subroutines

- real(kind=wp) function, dimension(size(x)) [normalize](#) (x)
 

*Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.*
- real(kind=wp) function [interpolate](#) (x1, x2, f1, f2, x)
 

*Function to interpolate and extrapolate linear on a line.*
- real(kind=wp) function [linearinterpolate](#) (X, Y, xx)
 

*Function for linearly interpolating and extrapolating.*
- real(kind=wp) function [loglinearinterpolate](#) (X, Y, xx)
 

*Function for log-linearly interpolating and extrapolating.*
- real(kind=wp) function [linearinterpolatecyclic](#) (X, Y, xx)
 

*Function for linearly interpolating cyclic data.*
- real(kind=wp) function [loglinearinterpolatecyclic](#) (X, Y, xx)
 

*Function for log-linearly interpolating and extrapolating cyclic data.*
- real(kind=wp) function [linearinterpolateangles](#) (X, Y, xx)
 

*Function for linearly interpolating and extrapolating angles.*
- subroutine [sortandlocate](#) (X, IDXsorted, xx, locID, locIDm1, Cyclic)
 

*Subroutine to bracket a given value xx within two elements of an array after sort in descending order.*
- real(kind=wp) function [stepsize](#) (i, x)
 

*Function for the Trapezium rule.*
- real(kind=wp) function [trapeziumrule](#) (x, f)
 

*Function for the Trapezium rule.*
- real(kind=wp) function [integratewithexponentcomponent](#) (x, f)
 

*Function for the Trapezium rule.*

- subroutine `ssort` (`x, iy, n, kflag`)

*Subroutine to sort a vector in either ascending or descending order.*

### 7.28.1 Detailed Description

Module with vector utility functions.

### 7.28.2 Function/Subroutine Documentation

**7.28.2.1 real (kind=wp) function vectorutilities::interpolateLine ( real (kind=wp), intent(in) `x1`, real (kind=wp), intent(in) `x2`, real (kind=wp), intent(in) `f1`, real (kind=wp), intent(in) `f2`, real (kind=wp), intent(in) `x` )**

Function to interpolate and extrapolate linear on a line.

#### Parameters

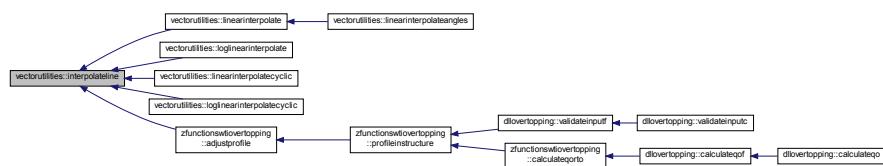
in	<code>x1</code>	first x-value
in	<code>x2</code>	second x-value
in	<code>f1</code>	first function-value
in	<code>f2</code>	second function-value
in	<code>x</code>	x-value for which the function value is desired

#### Returns

value of the dependent variable associated with `xx`

Definition at line 44 of file `vectorUtilities.f90`.

Here is the caller graph for this function:



**7.28.2.2 real (kind=wp) function vectorutilities::integrateWithExponentComponent ( real (kind=wp), dimension(:), intent(in) `x`, real (kind=wp), dimension(:), intent(in) `f` )**

Function for the Trapezium rule.

#### Parameters

in	<code>x</code>	vector of x-values
in	<code>f</code>	vector of function-values

#### Returns

value after the Trapezium Rule

Definition at line 452 of file `vectorUtilities.f90`.

7.28.2.3 real (kind=wp) function vectorutilities::linearinterpolate ( real (kind=wp), dimension(:), intent(in) X, real (kind=wp), dimension(:), intent(in) Y, real (kind=wp), intent(in) xx )

Function for linearly interpolating and extrapolating.

**Parameters**

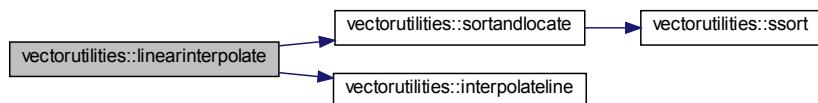
in	x	vector with independent elements X - in increasing order
in	y	vector with dependent elements Y (depending on X)
in	xx	x-value for which the dependent variable is desired

**Returns**

value of the dependent variable associated with xx

Definition at line 59 of file vectorUtilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.28.2.4 real (kind=wp) function vectorutilities::linearinterpolateangles ( real (kind=wp), dimension(:), intent(in) X, real (kind=wp), dimension(:), intent(in) Y, real (kind=wp), intent(in) xx )

Function for linearly interpolating and extrapolating angles.

**Parameters**

in	x	vector with independent elements X
in	y	vector with dependent elements Y (depending on X)
in	xx	x-value for which the dependent variable is desired

**Returns**

value of the dependent variable associated with xx

Definition at line 292 of file vectorUtilities.f90.

Here is the call graph for this function:



7.28.2.5 real (kind=wp) function vectorutilities::linearinterpolatecyclic ( real (kind=wp), dimension(:), intent(in) X, real (kind=wp), dimension(:), intent(in) Y, real (kind=wp), intent(in) xx )

Function for linearly interpolating cyclic data.

**Parameters**

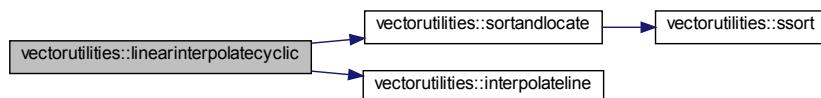
in	<i>x</i>	vector with independent elements X - in increasing order
in	<i>y</i>	vector with dependent elements Y (depending on X)
in	<i>xx</i>	x-value for which the dependent variable is desired

**Returns**

value of the dependent variable associated with *xx*

Definition at line 178 of file vectorUtilities.f90.

Here is the call graph for this function:



**7.28.2.6 real (kind=wp) function vectorutilities::loglinearinterpolate ( real (kind=wp), dimension(:), intent(in) *X*, real (kind=wp), dimension(:), intent(in) *Y*, real (kind=wp), intent(in) *xx* )**

Function for log-linearly interpolating and extrapolating.

**Parameters**

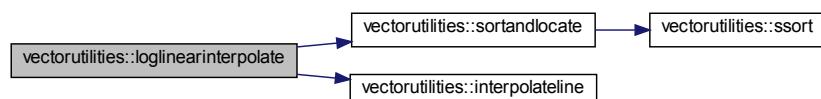
in	<i>x</i>	vector with independent elements X - in decreasing order
in	<i>y</i>	vector with dependent elements Y (depending on X)
in	<i>xx</i>	x-value for which the dependent variable is desired

**Returns**

value of the dependent variable associated with *xx*

Definition at line 109 of file vectorUtilities.f90.

Here is the call graph for this function:



**7.28.2.7 real (kind=wp) function vectorutilities::loglinearinterpolatecyclic ( real (kind=wp), dimension(:), intent(in) *X*, real (kind=wp), dimension(:), intent(in) *Y*, real (kind=wp), intent(in) *xx* )**

Function for log-linearly interpolating and extrapolating cyclic data.

**Parameters**

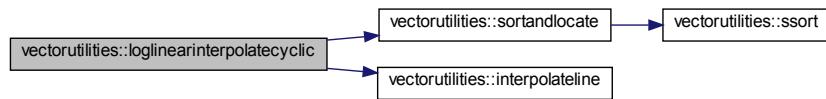
in	<i>x</i>	vector with independent elements X - in decreasing order
in	<i>y</i>	vector with dependent elements Y (depending on X)
in	<i>xx</i>	x-value for which the dependent variable is desired

**Returns**

value of the dependent variable associated with *xx*

Definition at line 227 of file vectorUtilities.f90.

Here is the call graph for this function:



### 7.28.2.8 real (kind=wp) function, dimension(size(*x*)) vectorutilities::normalize ( real (kind=wp), dimension(:), intent(in) *x* )

Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.

**Parameters**

in	<i>x</i>	vector to be normalized
----	----------	-------------------------

**Returns**

output vector

Definition at line 23 of file vectorUtilities.f90.

### 7.28.2.9 subroutine vectorutilities::sortandlocate ( real (kind=wp), dimension(:), intent(inout) *X*, integer, dimension(:) *IDXsorted*, real (kind=wp), intent(in) *xx*, integer *locID*, integer *locIDm1*, logical *Cyclic* )

Subroutine to bracket a given value *xx* within two elements of an array after sort in descending order.

**Parameters**

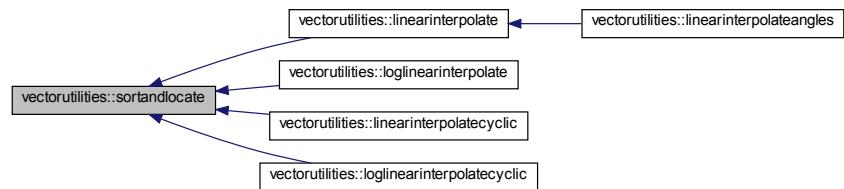
in,out	<i>x</i>	vector with independent elements X - in decreasing order
	<i>idxsorted</i>	sorted indices of <i>x</i> vector
in	<i>xx</i>	x-value for which the dependent variable is desired
	<i>locid</i>	sought index
	<i>locidm1</i>	sought index - 1
	<i>cyclic</i>	a flag to be true if data are cyclic; false if not

Definition at line 338 of file vectorUtilities.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



#### 7.28.2.10 subroutine `vectorutilities::ssort` ( `real (kind=wp), dimension(n) x, integer, dimension(n), intent(inout) iy, integer, intent(in) n, integer, intent(in) kflag` )

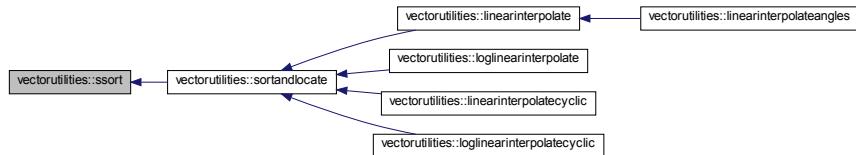
Subroutine to sort a vector in either ascending or descending order.

##### Parameters

	<i>x</i>	vector of x-values
<code>in, out</code>	<i>iy</i>	vector of indices to contain sorted indices of <i>x</i>
<code>in</code>	<i>n</i>	size of <i>x</i> and <i>iy</i>
<code>in</code>	<i>kflag</i>	a flag to indicate ascending(positive) or descending (negative) sorting

Definition at line 483 of file `vectorUtilities.f90`.

Here is the caller graph for this function:



#### 7.28.2.11 `real (kind=wp) function vectorutilities::stepsize` ( `integer, intent(in) i, real (kind=wp), dimension(:), intent(in) x` )

Function for the Trapezium rule.

**Parameters**

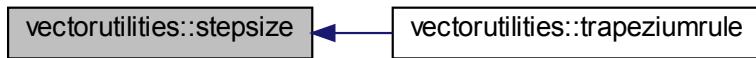
in	<i>i</i>	index of the vector x
in	<i>x</i>	vector of x-values

**Returns**

Step size

Definition at line 395 of file vectorUtilities.f90.

Here is the caller graph for this function:



**7.28.2.12 real (kind=wp) function vectorutilities::trapeziumrule ( real (kind=wp), dimension(:), intent(in) x, real (kind=wp), dimension(:), intent(in) f )**

Function for the Trapezium rule.

**Parameters**

in	<i>x</i>	vector of x-values
in	<i>f</i>	vector of function-values

**Returns**

value after the Trapizium Rule

Definition at line 423 of file vectorUtilities.f90.

Here is the call graph for this function:



## 7.29 waveparametersutilities Module Reference

Module to calculate the wave steepness or the wave period.

### Functions/Subroutines

- real(wp) function [computewavesteeepness](#) (waveHeight, wavePeriod)

*Module to calculate the the wave steepness from the wave height and the wave period.*

- real(wp) function [computewaveperiod](#) (waveHeight, waveSteepness)

*Module to calculate the the wave period from the wave height and the wave steepness.*

### 7.29.1 Detailed Description

Module to calculate the wave steepness or the wave period.

### 7.29.2 Function/Subroutine Documentation

#### 7.29.2.1 real(wp) function waveparametersutilities::computewaveperiod ( real(wp), intent(in) *waveHeight*, real(wp), intent(in) *waveSteepness* )

Module to calculate the the wave period from the wave height and the wave steepness.

##### Parameters

in	<i>waveheight</i>	wave height Hs
in	<i>wavestEEPNESS</i>	wave steepness

##### Returns

wave period Tm-1,0

Definition at line 59 of file waveParametersUtilities.f90.

#### 7.29.2.2 real(wp) function waveparametersutilities::computewavestEEPNESS ( real(wp), intent(in) *waveHeight*, real(wp), intent(in) *wavePeriod* )

Module to calculate the the wave steepness from the wave height and the wave period.

##### Parameters

in	<i>waveheight</i>	wave height Hs
in	<i>waveperiod</i>	wave period Tm-1,0

##### Returns

wave steepness

Definition at line 25 of file waveParametersUtilities.f90.

## 7.30 waverunup Module Reference

### Functions/Subroutines

- subroutine, public [iterationwaverunup](#) (geometry, h, Hm0, Tm\_10, gammaBeta\_z, modelFactors, z2, succes, errorMessage)
 

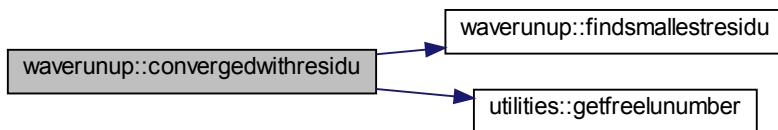
*iterationWaveRunup: iteration for the wave runup*
- real(kind=wp) function [determinestartingvalue](#) (i, relaxationFactor, z2\_start, z2\_end, Hm0)
- integer function [findsmallestresidu](#) (z2\_start, z2\_end)
- logical function [convergedwithresidu](#) (i, z2\_start, z2\_end)

### 7.30.1 Function/Subroutine Documentation

7.30.1.1 logical function `waverunup::convergedwithresidu` ( `integer, intent(in) i, real(kind=wp), dimension(:), intent(in) z2_start, real(kind=wp), dimension(:, intent(inout) z2_end )` ) [private]

Definition at line 284 of file `waveRunup.f90`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.30.1.2 `real(kind=wp)` function `waverunup::determinestartingvalue` ( `integer, intent(in) i, real(kind=wp), intent(in) relaxationFactor, real(kind=wp), dimension(:, intent(in) z2_start, real(kind=wp), dimension(:, intent(in) z2_end, real(kind=wp), intent(in) Hm0 )` ) [private]

Definition at line 231 of file `waveRunup.f90`.

Here is the call graph for this function:



Here is the caller graph for this function:



7.30.1.3 `integer` function `waverunup::findsmallestresidu` ( `real(kind=wp), dimension(:, intent(in) z2_start, real(kind=wp), dimension(:, intent(in) z2_end )` ) [private]

Definition at line 260 of file `waveRunup.f90`.

Here is the caller graph for this function:



**7.30.1.4 subroutine, public waverunup::iterationwaverunup ( type (tpgeometry), intent(in) geometry, real(wp), intent(in) h, real(wp), intent(in) hm0, real(wp), intent(in) Tm\_10, real(wp), intent(inout) gammabeta\_z, type (tpovertoppinginput), intent(in) modelFactors, real(wp), intent(out) z2, logical, intent(out) succes, character(len=\*), intent(out) errorMessage )**

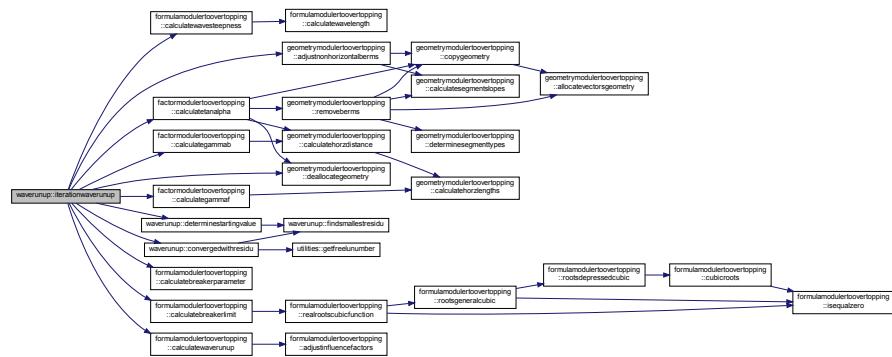
iterationWaveRunup: iteration for the wave runup

#### Parameters

in	geometry	structure with geometry data
in	h	local water level (m+NAP)
in	hm0	significant wave height (m)
in	tm_10	spectral wave period (s)
in,out	gammabeta_z	influence factor angle wave attack 2% run-up
in	modelfactors	structure with model factors
out	z2	2% wave run-up (m)
out	succes	flag for succes
out	errormessage	error message

Definition at line 33 of file waveRunup.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



## 7.31 writeutilities Module Reference

### Data Types

- interface [writetodevice](#)

## Functions/Subroutines

- subroutine `writetodevice_line` (`fileId`, `line`)  
*Write to device.*
- subroutine `writetodevice_text_real` (`fileId`, `formatText`, `textValue`, `realValue`)  
*Write to device.*
- subroutine, public `writetodevicechars` (`fileId`, `char`, `count`)  
*Write count times the same character to output device.*
- character(`len=5`) function, public `inttostr` (`intValue`)  
*Convert integer to string.*

### 7.31.1 Function/Subroutine Documentation

#### 7.31.1.1 character(len=5) function, public `writeutilities::inttostr` ( `integer, intent(in) intValue` )

Convert integer to string.

##### Parameters

<code>in</code>	<code>intValue</code>	integer number to convert
-----------------	-----------------------	---------------------------

##### Returns

return text

Definition at line 71 of file `writeUtilities.f90`.

#### 7.31.1.2 subroutine `writeutilities::writetodevice_line` ( `integer, intent(in) fileId`, `character(len=*)`, `intent(in) line` ) [private]

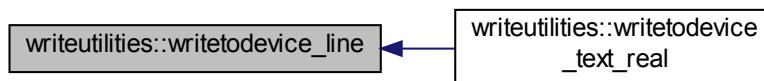
Write to device.

##### Parameters

<code>in</code>	<code>fileId</code>	device number to write to, 0 = screen
<code>in</code>	<code>line</code>	text to write

Definition at line 28 of file `writeUtilities.f90`.

Here is the caller graph for this function:



#### 7.31.1.3 subroutine `writeutilities::writetodevice_text_real` ( `integer, intent(in) fileId`, `character(len=*)`, `intent(in) formatText`, `character(len=*)`, `intent(in) textValue`, `real (kind=wp)`, `intent(in) realValue` ) [private]

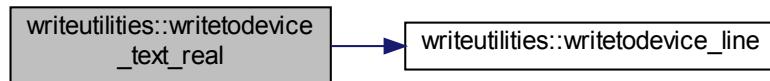
Write to device.

**Parameters**

in	<i>fileid</i>	device number to write to, 0 = screen
in	<i>formattext</i>	format specifier
in	<i>textvalue</i>	text to write
in	<i>realvalue</i>	real value to print

Definition at line 42 of file writeUtilities.f90.

Here is the call graph for this function:



**7.31.1.4 subroutine, public writeutilities::writetodevicechars ( integer, intent(in) *fileid*, character(len=1), intent(in) *char*, integer, intent(in) *count* )**

Write count times the same character to output device.

**Parameters**

in	<i>fileid</i>	device number to write to, 0 = screen
in	<i>char</i>	character to be printed
in	<i>count</i>	number of times to output the character

Definition at line 57 of file writeUtilities.f90.

## 7.32 zfunctionsztiovertopping Module Reference

Module for the Limit State Functions (Z-functions) for wave overtopping.

### Functions/Subroutines

- subroutine, public [calculateqorto](#) (dikeHeight, modelFactors, overtopping, load, geometry, succes, errorMessage)
 

*Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.*
- subroutine, public [profileinstructure](#) (nrCoordinates, xcoordinates, ycoordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)
 

*Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.*
- subroutine [adjustprofile](#) (nrCoordinates, coordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)
 

*Subroutine adjust the profile due to a desired dike height.*
- real(kind=wp) function, public [zfunclogratios](#) (qo, qc, mqo, mqc, success, errorMessage)
 

*Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)*

### 7.32.1 Detailed Description

Module for the Limit State Functions (Z-functions) for wave overtopping.

### 7.32.2 Function/Subroutine Documentation

7.32.2.1 subroutine zfunctionswtiovertopping::adjustprofile ( integer, intent(in) *nrCoordinates*, type(tpprofilecoordinate), dimension(nrcoordinates), intent(in) *coordinates*, real(kind=wp), intent(in) *dikeHeight*, integer, intent(out) *nrCoordsAdjusted*, real(kind=wp), dimension(:,), pointer *xCoordsAdjusted*, real(kind=wp), dimension(:,), pointer *zCoordsAdjusted*, logical, intent(out) *succes*, character(len=\*) intent(out) *errorMessage* ) [private]

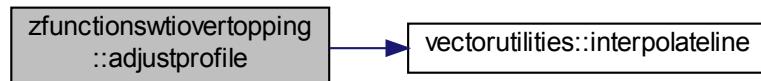
Subroutine adjust the profile due to a desired dike height.

#### Parameters

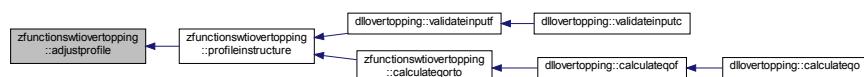
in	<i>nrcoordinates</i>	number of coordinates of the profile
in	<i>coordinates</i>	structure for the profile
in	<i>dikeheight</i>	dike height
out	<i>nrcoradsadjusted</i>	number of coordinates in the adjusted profile
	<i>xcoordsadjusted</i>	vector with x-coordinates of the adjusted profile
	<i>zcoordsadjusted</i>	vector with y-coordinates of the adjusted profile
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 109 of file zFunctionsWTIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.2.2 subroutine, public zfunctionswtiovertopping::calculateqorto ( real(kind=wp), intent(in) *dikeHeight*, type(tpovertoppinginput), intent(inout) *modelFactors*, type (tpovertopping), intent(out) *overtopping*, type (tpload), intent(in) *load*, type (tpgeometry), intent(in) *geometry*, logical, intent(out) *succes*, character(len=\*) intent(out) *errorMessage* )

Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.

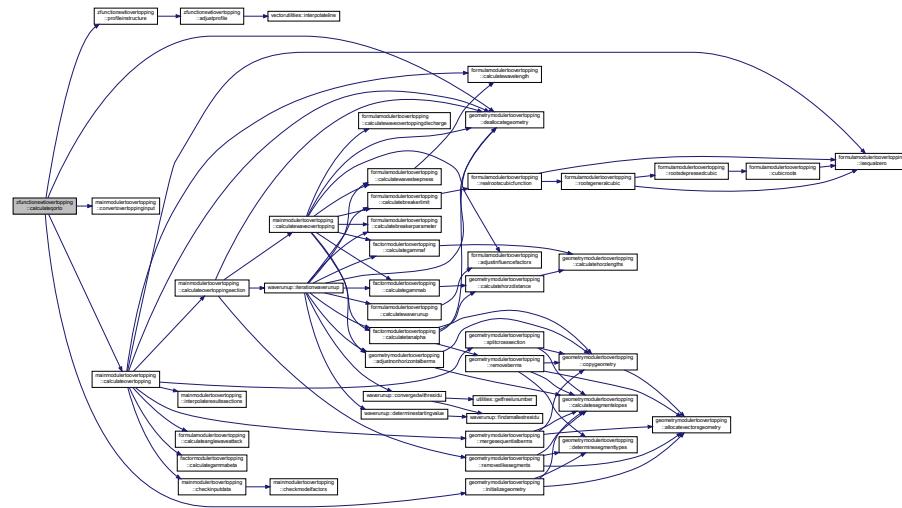
#### Parameters

in	<i>dikeheight</i>	dike height
----	-------------------	-------------

<i>in,out</i>	<i>modelfactors</i>	struct with model factors
<i>out</i>	<i>overtopping</i>	structure with overtopping results
<i>in</i>	<i>geometry</i>	structure with geometry data
<i>in</i>	<i>load</i>	structure with load parameters
<i>out</i>	<i>succes</i>	flag for succes
<i>out</i>	<i>errormessage</i>	error message

Definition at line 32 of file zFunctionsWtIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.32.2.3 subroutine, public zfunctionsWtiovertopping::profileinstructure ( integer, intent(in) *nrCoordinates*, real(kind=wp), dimension(nrcoordinates), intent(in) *xcoordinates*, real(kind=wp), dimension(nrcoordinates), intent(in) *ycoordinates*, real(kind=wp), intent(in) *dikeHeight*, integer, intent(out) *nrCoordsAdjusted*, real(kind=wp), dimension(:, pointer) *xCoordsAdjusted*, real(kind=wp), dimension(:, pointer) *zCoordsAdjusted*, logical, intent(out) *succes*, character(len=\*) *errorMessage* )**

Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.

#### Parameters

<i>in</i>	<i>nrcoordinates</i>	number of coordinates of the profile
<i>in</i>	<i>xcoordinates</i>	vector with x-coordinates of the profile
<i>in</i>	<i>ycoordinates</i>	vector with y-coordinates of the profile

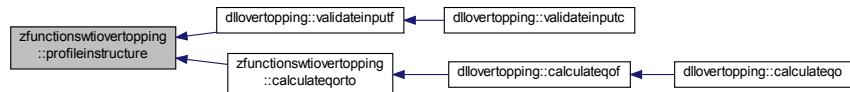
in	<i>dikeheight</i>	dike height
out	<i>nrcoordsadjusted</i>	number of coordinates in the adjusted profile
	<i>xcoordsadjusted</i>	vector with x-coordinates of the adjusted profile
	<i>zcoordsadjusted</i>	vector with y-coordinates of the adjusted profile
out	<i>succes</i>	flag for succes
out	<i>errormessage</i>	error message

Definition at line 84 of file zFunctionsWTIOvertopping.f90.

Here is the call graph for this function:



Here is the caller graph for this function:



**7.32.2.4 real (kind=wp) function, public zfunctionsztiovertopping::zfunclogratios ( real (kind=wp), intent(in) *qo*, real (kind=wp), intent(in) *qc*, real (kind=wp), intent(in) *mqa*, real (kind=wp), intent(in) *mqc*, logical, intent(out) *success*, character(len=\*), intent(out) *errorMessage* )**

Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)

#### Parameters

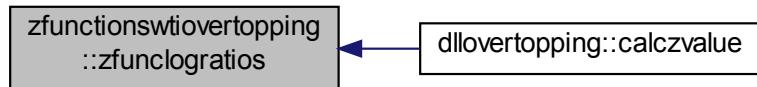
in	<i>qo</i>	computed overtopping discharge
in	<i>qc</i>	Critical overtopping discharge
in	<i>mqa</i>	Model factor computed overtopping discharge
in	<i>mqc</i>	Model factor Critical overtopping discharge
out	<i>success</i>	Flag for succes
out	<i>errormessage</i>	error message, only set if not successful

**Returns**

Value z-function

Definition at line 198 of file zFunctionsWtIOvertopping.f90.

Here is the caller graph for this function:



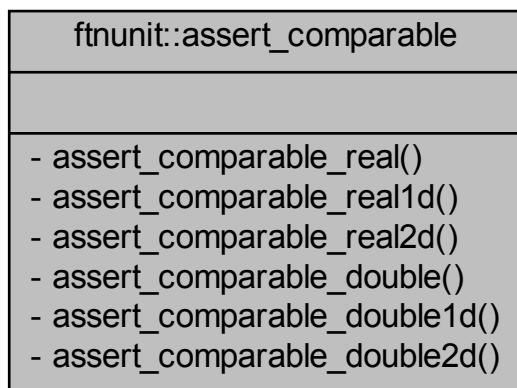


## Chapter 8

# Data Type Documentation

### 8.1 ftnunit::assert\_comparable Interface Reference

Collaboration diagram for ftnunit::assert\_comparable:



#### Private Member Functions

- subroutine `assert_comparable_real` (value1, value2, margin, text)
- subroutine `assert_comparable_real1d` (array1, array2, margin, text)
- subroutine `assert_comparable_real2d` (array1, array2, margin, text)
- subroutine `assert_comparable_double` (value1, value2, margin, text)
- subroutine `assert_comparable_double1d` (array1, array2, margin, text)
- subroutine `assert_comparable_double2d` (array1, array2, margin, text)

#### 8.1.1 Detailed Description

Definition at line 90 of file ftnunit.f90.

## 8.1.2 Member Function/Subroutine Documentation

8.1.2.1 subroutine ftnunit::assert\_comparable::assert\_comparable\_double ( real(kind=dp), intent(in) value1, real(kind=dp), intent(in) value2, real(kind=dp), intent(in) margin, character(len=\*), intent(in) text ) [private]

Definition at line 966 of file ftnunit.f90.

8.1.2.2 subroutine ftnunit::assert\_comparable::assert\_comparable\_double1d ( real(kind=dp), dimension(:), intent(in) array1, real(kind=dp), dimension(:), intent(in) array2, real(kind=dp), intent(in) margin, character(len=\*), intent(in) text ) [private]

Definition at line 997 of file ftnunit.f90.

8.1.2.3 subroutine ftnunit::assert\_comparable::assert\_comparable\_double2d ( real(kind=dp), dimension(:, :), intent(in) array1, real(kind=dp), dimension(:, :), intent(in) array2, real(kind=dp), intent(in) margin, character(len=\*), intent(in) text ) [private]

Definition at line 1061 of file ftnunit.f90.

8.1.2.4 subroutine ftnunit::assert\_comparable::assert\_comparable\_real ( real, intent(in) value1, real, intent(in) value2, real, intent(in) margin, character(len=\*), intent(in) text ) [private]

Definition at line 802 of file ftnunit.f90.

8.1.2.5 subroutine ftnunit::assert\_comparable::assert\_comparable\_real1d ( real, dimension(:), intent(in) array1, real, dimension(:), intent(in) array2, real, intent(in) margin, character(len=\*), intent(in) text ) [private]

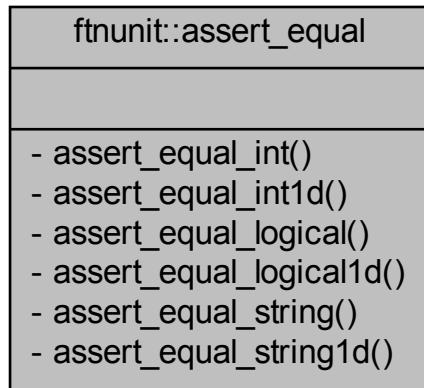
Definition at line 834 of file ftnunit.f90.

8.1.2.6 subroutine ftnunit::assert\_comparable::assert\_comparable\_real2d ( real, dimension(:, :), intent(in) array1, real, dimension(:, :), intent(in) array2, real, intent(in) margin, character(len=\*), intent(in) text ) [private]

Definition at line 898 of file ftnunit.f90.

## 8.2 ftnunit::assert\_equal Interface Reference

Collaboration diagram for ftnunit::assert\_equal:



### Private Member Functions

- subroutine `assert_equal_int` (`value1, value2, text`)
- subroutine `assert_equal_int1d` (`array1, array2, text`)
- subroutine `assert_equal_logical` (`value1, value2, text`)
- subroutine `assert_equal_logical1d` (`array1, array2, text`)
- subroutine `assert_equal_string` (`value1, value2, text`)
- subroutine `assert_equal_string1d` (`array1, array2, text`)

#### 8.2.1 Detailed Description

Definition at line 81 of file ftnunit.f90.

#### 8.2.2 Member Function/Subroutine Documentation

**8.2.2.1 subroutine ftnunit::assert\_equal::assert\_equal\_int ( integer, intent(in) `value1`, integer, intent(in) `value2`, character(len=\*)`, intent(in) text`  ) [private]**

Definition at line 724 of file ftnunit.f90.

**8.2.2.2 subroutine ftnunit::assert\_equal::assert\_equal\_int1d ( integer, dimension(:), intent(in) `array1`, integer, dimension(:), intent(in) `array2`, character(len=\*)`, intent(in) text`  ) [private]**

Definition at line 748 of file ftnunit.f90.

**8.2.2.3 subroutine ftnunit::assert\_equal::assert\_equal\_logical ( logical, intent(in) `value1`, logical, intent(in) `value2`, character(len=\*)`, intent(in) text`  ) [private]**

Definition at line 575 of file ftnunit.f90.

---

8.2.2.4 subroutine ftnunit::assert\_equal::assert\_equal\_logical1d ( logical, dimension(:), intent(in) array1, logical, dimension(:), intent(in) array2, character(len=\*), intent(in) text ) [private]

Definition at line 599 of file ftnunit.f90.

8.2.2.5 subroutine ftnunit::assert\_equal::assert\_equal\_string ( character(len=\*), intent(in) value1, character(len=\*), intent(in) value2, character(len=\*), intent(in) text ) [private]

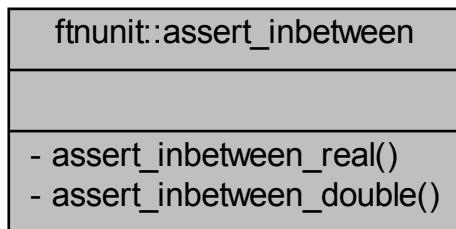
Definition at line 649 of file ftnunit.f90.

8.2.2.6 subroutine ftnunit::assert\_equal::assert\_equal\_string1d ( character(len=\*), dimension(:), intent(in) array1, character(len=\*), dimension(:), intent(in) array2, character(len=\*), intent(in) text ) [private]

Definition at line 675 of file ftnunit.f90.

## 8.3 ftnunit::assert\_inbetween Interface Reference

Collaboration diagram for ftnunit::assert\_inbetween:



### Private Member Functions

- subroutine [assert\\_inbetween\\_real](#) (value, vmin, vmax, text)
- subroutine [assert\\_inbetween\\_double](#) (value, vmin, vmax, text)

#### 8.3.1 Detailed Description

Definition at line 98 of file ftnunit.f90.

#### 8.3.2 Member Function/Subroutine Documentation

8.3.2.1 subroutine ftnunit::assert\_inbetween::assert\_inbetween\_double ( real(kind=dp), intent(in) value, real(kind=dp), intent(in) vmin, real(kind=dp), intent(in) vmax, character(len=\*), intent(in) text ) [private]

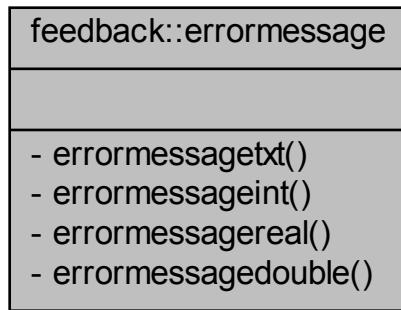
Definition at line 1168 of file ftnunit.f90.

8.3.2.2 subroutine ftnunit::assert\_inbetween::assert\_inbetween\_real ( real, intent(in) value, real, intent(in) vmin, real, intent(in) vmax, character(len=\*), intent(in) text ) [private]

Definition at line 1129 of file ftnunit.f90.

## 8.4 feedback::errormessage Interface Reference

Collaboration diagram for feedback::errormessage:



### Private Member Functions

- subroutine [errormessagetxt](#) (msgtext)  
*Write an error message.*
- subroutine [errormessageint](#) (msgtext, value)  
*Write an error message with an integer value.*
- subroutine [errormessagereal](#) (msgtext, value)  
*Write an error message with a real value.*
- subroutine [errormessagedouble](#) (msgtext, value)  
*Write an error message with a double precision real value.*

### 8.4.1 Detailed Description

Definition at line 149 of file feedback.f90.

### 8.4.2 Member Function/Subroutine Documentation

8.4.2.1 subroutine [feedback::errormessage::errormessagedouble](#) ( character(len=\*), intent(in) msgtext, real(kind=dp), intent(in) value ) [private]

Write an error message with a double precision real value.

**Parameters**

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 661 of file feedback.f90.

**8.4.2.2 subroutine feedback::errormessage::errormessageint ( character(len=\*) intent(in) *msgtext*, integer, intent(in) *value* ) [private]**

Write an error message with an integer value.

**Parameters**

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 631 of file feedback.f90.

**8.4.2.3 subroutine feedback::errormessage::errormessagereal ( character(len=\*) intent(in) *msgtext*, real, intent(in) *value* ) [private]**

Write an error message with a real value.

**Parameters**

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 646 of file feedback.f90.

**8.4.2.4 subroutine feedback::errormessage::errormessagetxt ( character(len=\*) intent(in) *msgtext* ) [private]**

Write an error message.

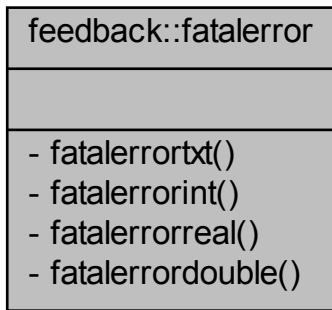
**Parameters**

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 608 of file feedback.f90.

## 8.5 feedback::fatalerror Interface Reference

Collaboration diagram for feedback::fatalerror:



### Private Member Functions

- subroutine **fatalerrortxt** (msgtext)  
*Write a fatal error message.*
- subroutine **fatalerrorint** (msgtext, value)  
*Write a fatal error message with an integer value.*
- subroutine **fatalerrorreal** (msgtext, value)  
*Write a fatal error message with a real value.*
- subroutine **fatalerrordouble** (msgtext, value)  
*Write a fatal error message with a double precision real value.*

#### 8.5.1 Detailed Description

Definition at line 163 of file feedback.f90.

#### 8.5.2 Member Function/Subroutine Documentation

**8.5.2.1 subroutine feedback::fatalerror::fatalerrordouble ( character(len=\*) intent(in) msgtext, real(kind=dp), intent(in) value ) [private]**

Write a fatal error message with a double precision real value.

##### Parameters

in	msgtext	Error message to write
in	value	Integer value to add

Definition at line 815 of file feedback.f90.

**8.5.2.2 subroutine feedback::fatalerror::fatalerrorint ( character(len=\*) intent(in) msgtext, integer, intent(in) value ) [private]**

Write a fatal error message with an integer value.

**Parameters**

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 785 of file feedback.f90.

**8.5.2.3 subroutine feedback::fatalerror::fatalerrorreal ( character(len=\*) , intent(in) *msgtext*, real, intent(in) *value* ) [private]**

Write a fatal error message with a real value.

**Parameters**

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 800 of file feedback.f90.

**8.5.2.4 subroutine feedback::fatalerror::fatalerrortxt ( character(len=\*) , intent(in) *msgtext* ) [private]**

Write a fatal error message.

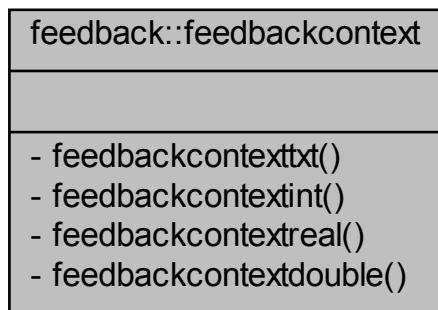
**Parameters**

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 744 of file feedback.f90.

## 8.6 feedback::feedbackcontext Interface Reference

Collaboration diagram for feedback::feedbackcontext:



### Private Member Functions

- subroutine **feedbackcontexttxt** (*text*)  
*Register a context for error messages.*
- subroutine **feedbackcontextint** (*text, value*)

*Register a context for error messages - with integer value.*

- subroutine **feedbackcontextreal** (*text, value*)

*Register a context for error messages - with real value.*

- subroutine **feedbackcontextdouble** (*text, value*)

*Register a context for error messages - with double precision real value.*

### 8.6.1 Detailed Description

Definition at line 142 of file feedback.f90.

### 8.6.2 Member Function/Subroutine Documentation

**8.6.2.1 subroutine feedback::feedbackcontext::feedbackcontextdouble ( character(len=\*) intent(in) *text*, real(kind=dp), intent(in) *value* ) [private]**

Register a context for error messages - with double precision real value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 900 of file feedback.f90.

**8.6.2.2 subroutine feedback::feedbackcontext::feedbackcontextint ( character(len=\*) intent(in) *text*, integer, intent(in) *value* ) [private]**

Register a context for error messages - with integer value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 870 of file feedback.f90.

**8.6.2.3 subroutine feedback::feedbackcontext::feedbackcontextreal ( character(len=\*) intent(in) *text*, real(kind=sp), intent(in) *value* ) [private]**

Register a context for error messages - with real value.

#### Parameters

in	<i>text</i>	text to be added to the context stack
in	<i>value</i>	number to be added to the text

Definition at line 885 of file feedback.f90.

**8.6.2.4 subroutine feedback::feedbackcontext::feedbackcontexttxt ( character(len=\*) intent(in) *text* ) [private]**

Register a context for error messages.

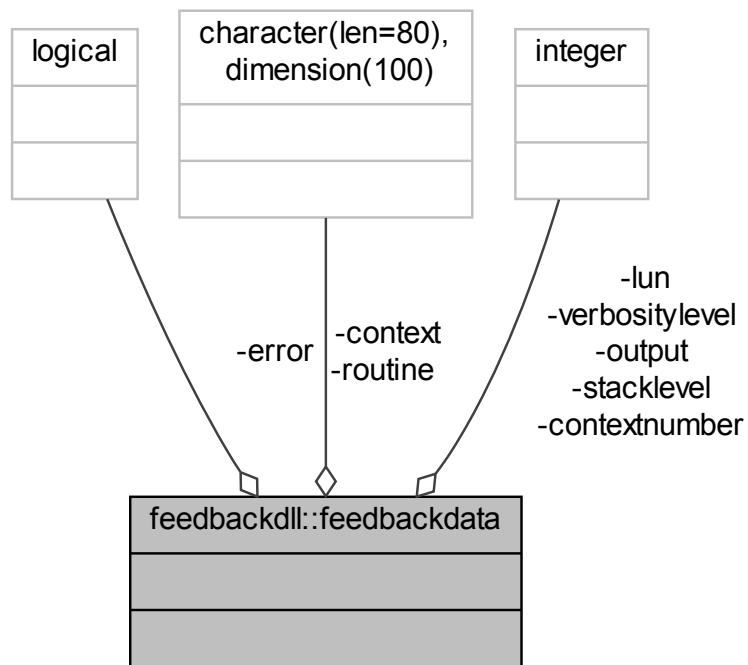
#### Parameters

in	text	text to be added to the context stack
----	------	---------------------------------------

Definition at line 852 of file feedback.f90.

## 8.7 feedbackdll::feedbackdata Type Reference

Collaboration diagram for feedbackdll::feedbackdata:



### Private Attributes

- character(len=80), dimension(100) **routine**
- character(len=80), dimension(100) **context**
- integer **output**
- integer **contextnumber**
- integer **stacklevel**
- integer **verbositylevel**
- integer **lun** = -1
- logical **error** = .true.

### 8.7.1 Detailed Description

Definition at line 24 of file feedbackDLL.f90.

## 8.7.2 Member Data Documentation

8.7.2.1 character(len=80), dimension(100) feedbackdll::feedbackdata::context [private]

Definition at line 26 of file feedbackDLL.f90.

8.7.2.2 integer feedbackdll::feedbackdata::contextnumber [private]

Definition at line 28 of file feedbackDLL.f90.

8.7.2.3 logical feedbackdll::feedbackdata::error = .true. [private]

Definition at line 32 of file feedbackDLL.f90.

8.7.2.4 integer feedbackdll::feedbackdata::lun = -1 [private]

Definition at line 31 of file feedbackDLL.f90.

8.7.2.5 integer feedbackdll::feedbackdata::output [private]

Definition at line 27 of file feedbackDLL.f90.

8.7.2.6 character(len=80), dimension(100) feedbackdll::feedbackdata::routine [private]

Definition at line 25 of file feedbackDLL.f90.

8.7.2.7 integer feedbackdll::feedbackdata::stacklevel [private]

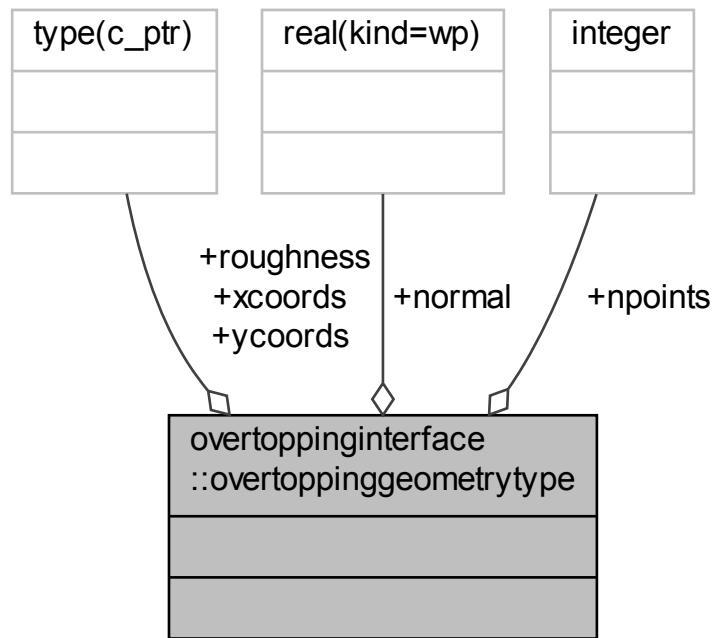
Definition at line 29 of file feedbackDLL.f90.

8.7.2.8 integer feedbackdll::feedbackdata::verbositylevel [private]

Definition at line 30 of file feedbackDLL.f90.

## 8.8 overtoppinginterface::overtoppinggeometrytype Type Reference

Collaboration diagram for overtoppinginterface::overtoppinggeometrytype:



### Public Attributes

- `real(kind=wp)` `normal`
- `integer` `npoints`
- `type(c_ptr)` `xcoords`
- `type(c_ptr)` `ycoords`
- `type(c_ptr)` `roughness`

### 8.8.1 Detailed Description

Definition at line 25 of file `overtoppingInterface.f90`.

### 8.8.2 Member Data Documentation

#### 8.8.2.1 `real(kind=wp)` `overtoppinginterface::overtoppinggeometrytype::normal`

Definition at line 26 of file `overtoppingInterface.f90`.

#### 8.8.2.2 `integer` `overtoppinginterface::overtoppinggeometrytype::npoints`

Definition at line 27 of file `overtoppingInterface.f90`.

## 8.8.2.3 type(c\_ptr) overtoppinginterface::overtoppinggeometrytype::roughness

Definition at line 30 of file overtoppingInterface.f90.

## 8.8.2.4 type(c\_ptr) overtoppinginterface::overtoppinggeometrytype::xcoords

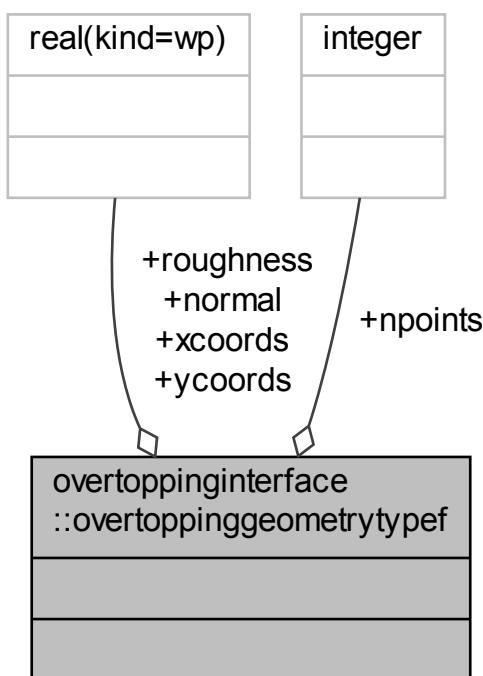
Definition at line 28 of file overtoppingInterface.f90.

## 8.8.2.5 type(c\_ptr) overtoppinginterface::overtoppinggeometrytype::ycoords

Definition at line 29 of file overtoppingInterface.f90.

## 8.9 overtoppinginterface::overtoppinggeometrytypef Type Reference

Collaboration diagram for overtoppinginterface::overtoppinggeometrytypef:



### Public Attributes

- real(kind=wp) **normal**
- integer **npoints**
- real(kind=wp), dimension(:), pointer **xcoords**
- real(kind=wp), dimension(:), pointer **ycoords**
- real(kind=wp), dimension(:), pointer **roughness**

### 8.9.1 Detailed Description

Definition at line 33 of file `overtoppingInterface.f90`.

### 8.9.2 Member Data Documentation

#### 8.9.2.1 `real(kind=wp) overtoppinginterface::overtoppinggeometrytypef::normal`

Definition at line 34 of file `overtoppingInterface.f90`.

#### 8.9.2.2 `integer overtoppinginterface::overtoppinggeometrytypef::npoints`

Definition at line 35 of file `overtoppingInterface.f90`.

#### 8.9.2.3 `real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::roughness`

Definition at line 38 of file `overtoppingInterface.f90`.

#### 8.9.2.4 `real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::xcoords`

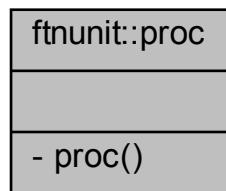
Definition at line 36 of file `overtoppingInterface.f90`.

#### 8.9.2.5 `real(kind=wp), dimension(:), pointer overtoppinginterface::overtoppinggeometrytypef::ycoords`

Definition at line 37 of file `overtoppingInterface.f90`.

## 8.10 ftnunit::proc Interface Reference

Collaboration diagram for `ftnunit::proc`:



### Private Member Functions

- subroutine `proc`

#### 8.10.1 Detailed Description

Definition at line 40 of file `ftnunit.f90`.

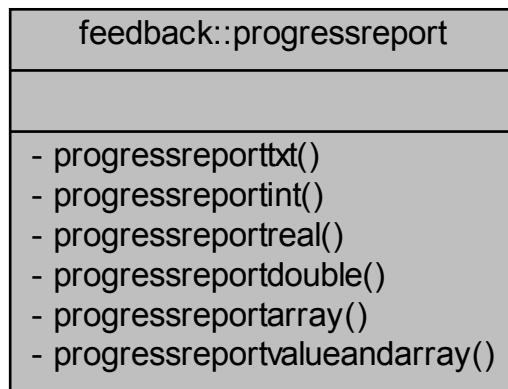
### 8.10.2 Constructor & Destructor Documentation

8.10.2.1 subroutine ftnunit::proc::proc( ) [private]

Definition at line 41 of file ftnunit.f90.

## 8.11 feedback::progressreport Interface Reference

Collaboration diagram for feedback::progressreport:



### Private Member Functions

- subroutine [progressreporttxt](#) (level, msgtext)  
*Write a single message to the output for the benefit of the user.*
- subroutine [progressreportint](#) (level, msgtext, value)  
*Write a single message with an integer value.*
- subroutine [progressreportreal](#) (level, msgtext, value)  
*Write a single message with a real value.*
- subroutine [progressreportdouble](#) (level, msgtext, value)  
*Write a single message with a double precision real value.*
- subroutine [progressreportarray](#) (level, msgText, x, size)  
*Print array values to report file.*
- subroutine [progressreportvalueandarray](#) (level, msgText, z, x, size)  
*Print z-value and array values to report file.*

### 8.11.1 Detailed Description

Definition at line 170 of file feedback.f90.

## 8.11.2 Member Function/Subroutine Documentation

8.11.2.1 subroutine `feedback::progressreport::progressreportarray` ( `integer, intent(in) level`, `character(len=*)`, `intent(in) msgText`, `real(kind=wp)`, `dimension(:)`, `intent(in) x`, `integer, intent(in) size` ) [private]

Print array values to report file.

**Parameters**

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>x</i>	x vector to write
in	<i>size</i>	the length of x

Definition at line 489 of file feedback.f90.

**8.11.2.2 subroutine feedback::progressreport::progressreportdouble ( integer, intent(in) *level*, character(len=\*), intent(in) *msgtext*, real(kind=dp), intent(in) *value* ) [private]**

Write a single message with a double precision real value.

**Parameters**

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Double value to add

Definition at line 472 of file feedback.f90.

**8.11.2.3 subroutine feedback::progressreport::progressreportint ( integer, intent(in) *level*, character(len=\*), intent(in) *msgtext*, integer, intent(in) *value* ) [private]**

Write a single message with an integer value.

**Parameters**

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 440 of file feedback.f90.

**8.11.2.4 subroutine feedback::progressreport::progressreportreal ( integer, intent(in) *level*, character(len=\*), intent(in) *msgtext*, real, intent(in) *value* ) [private]**

Write a single message with a real value.

**Parameters**

in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Real value to add

Definition at line 456 of file feedback.f90.

**8.11.2.5 subroutine feedback::progressreport::progressreporttxt ( integer, intent(in) *level*, character(len=\*), intent(in) *msgtext* ) [private]**

Write a single message to the output for the benefit of the user.

**Parameters**

in	<i>level</i>	Level of verbosity
----	--------------	--------------------

in	<i>msgtext</i>	Text to write
----	----------------	---------------

Definition at line 400 of file feedback.f90.

```
8.11.2.6 subroutine feedback::progressreport::progressreportvalueandarray ( integer, intent(in) level, character(len=*)  
intent(in) msgText, real(kind=wp), intent(in) z, real(kind=wp), dimension(:), intent(in) x, integer, intent(in) size )  
[private]
```

Print *z*-value and array values to report file.

#### Parameters

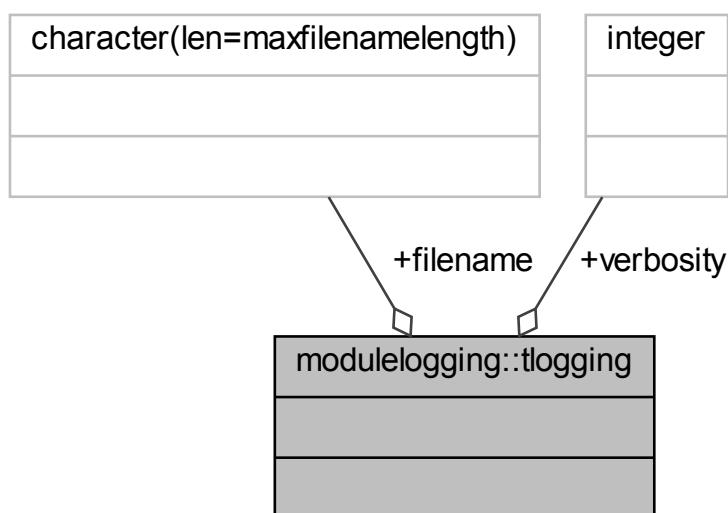
in	<i>level</i>	Level of verbosity
in	<i>msgtext</i>	Error message to write
in	<i>z</i>	<i>z</i> value to write
in	<i>x</i>	<i>x</i> vector to write
in	<i>size</i>	the length of <i>x</i>

Definition at line 525 of file feedback.f90.

## 8.12 modulelogging::tlogging Type Reference

TLogging: structure for steering the logging.

Collaboration diagram for modulelogging::tlogging:



#### Public Attributes

- integer **verbosity** = verboseNone  
*level of verbosity: one of verboseNone, verboseBasic, verboseDetailed, verboseDebugging*
- character(len=**maxfilenamelen**) **filename** = ''  
*filename of logging*

### 8.12.1 Detailed Description

TLogging: structure for steering the logging.

Definition at line 16 of file ModuleLogging.f90.

### 8.12.2 Member Data Documentation

8.12.2.1 character(len=maxfilenamelen) modulelogging::tlogging::filename = ''

filename of logging

Definition at line 18 of file ModuleLogging.f90.

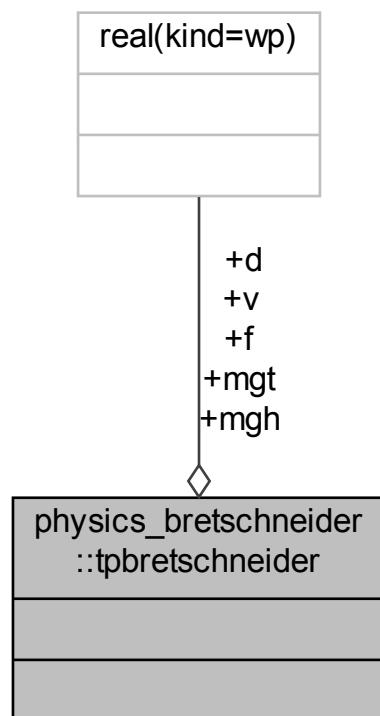
8.12.2.2 integer modulelogging::tlogging::verbosity = verboseNone

level of verbosity: one of verboseNone, verboseBasic, verboseDetailed, verboseDebugging

Definition at line 17 of file ModuleLogging.f90.

## 8.13 physics\_bretschneider::tpbretschneider Type Reference

Collaboration diagram for physics\_bretschneider::tpbretschneider:



## Public Attributes

- real(kind=wp) [v](#)  
*Wind velocity.*
- real(kind=wp) [d](#)  
*Depth of water for the section.*
- real(kind=wp) [f](#)  
*Fetch length for the section.*
- real(kind=wp) [mgh](#)  
*Model factor wave height.*
- real(kind=wp) [mgt](#)  
*Model factor wave period.*

### 8.13.1 Detailed Description

Definition at line 20 of file bretschneider.f90.

### 8.13.2 Member Data Documentation

#### 8.13.2.1 real (kind=wp) physics\_bretschneider::tpbretschneider::d

Depth of water for the section.

Definition at line 22 of file bretschneider.f90.

#### 8.13.2.2 real (kind=wp) physics\_bretschneider::tpbretschneider::f

Fetch length for the section.

Definition at line 23 of file bretschneider.f90.

#### 8.13.2.3 real (kind=wp) physics\_bretschneider::tpbretschneider::mgh

Model factor wave height.

Definition at line 24 of file bretschneider.f90.

#### 8.13.2.4 real (kind=wp) physics\_bretschneider::tpbretschneider::mgt

Model factor wave period.

Definition at line 25 of file bretschneider.f90.

#### 8.13.2.5 real (kind=wp) physics\_bretschneider::tpbretschneider::v

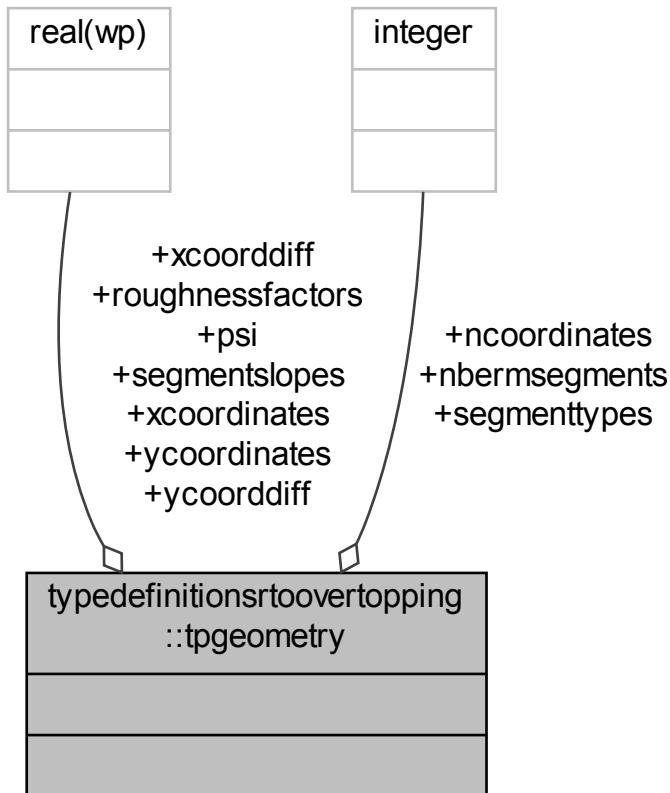
Wind velocity.

Definition at line 21 of file bretschneider.f90.

## 8.14 `typedefinitionsrovertopping::tpgeometry` Type Reference

`tpGeometry`: structure with geometry data

Collaboration diagram for `typedefinitionsrovertopping::tpgeometry`:



### Public Attributes

- `real(wp) psi`  
*dike normal (degrees)*
- `integer ncoordinates`  
*number of coordinates cross section*
- `real(wp), dimension(:,), pointer xcoordinates`  
*vector with x-coordinates cross section (m)*
- `real(wp), dimension(:,), pointer ycoordinates`  
*vector with y-coordinates cross section (m+NAP)*
- `real(wp), dimension(:,), pointer roughnessfactors`  
*vector with roughness factors cross section*
- `real(wp), dimension(:,), pointer xcoordiff`  
*vector with differences in x-coordinates (m)*
- `real(wp), dimension(:,), pointer ycoordiff`  
*vector with differences in y-coordinates (m)*

- real(wp), dimension(:), pointer **segmentslopes**  
*vector with slopes dike segments*
- integer, dimension(:), pointer **segmenttypes**  
*vector with segment types (1=slope,2=berm,3=other)*
- integer **nbermsegments**  
*number of berm segments*

### 8.14.1 Detailed Description

tpGeometry: structure with geometry data

Definition at line 18 of file typeDefinitionsRTOvertopping.f90.

### 8.14.2 Member Data Documentation

#### 8.14.2.1 integer typedefinitionsrtovertopping::tpgeometry::nbermsegments

number of berm segments

Definition at line 28 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.2 integer typedefinitionsrtovertopping::tpgeometry::ncoordinates

number of coordinates cross section

Definition at line 20 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.3 real(wp) typedefinitionsrtovertopping::tpgeometry::psi

dike normal (degrees)

Definition at line 19 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.4 real(wp), dimension(:), pointer typedefinitionsrtovertopping::tpgeometry::roughnessfactors

vector with roughness factors cross section

Definition at line 23 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.5 real(wp), dimension(:), pointer typedefinitionsrtovertopping::tpgeometry::segmentslopes

vector with slopes dike segments

Definition at line 26 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.6 integer, dimension(:), pointer typedefinitionsrtovertopping::tpgeometry::segmenttypes

vector with segment types (1=slope,2=berm,3=other)

Definition at line 27 of file typeDefinitionsRTOvertopping.f90.

#### 8.14.2.7 real(wp), dimension(:), pointer typedefinitionsrtovertopping::tpgeometry::xcoorddiff

vector with differences in x-coordinates (m)

Definition at line 24 of file typeDefinitionsRTOvertopping.f90.

8.14.2.8 `real(wp), dimension(:), pointer` `typedefinitionsrovertopping::tpgeometry::xcoordinates`

vector with x-coordinates cross section (m)

Definition at line 21 of file `typeDefinitionsRTOvertopping.f90`.

8.14.2.9 `real(wp), dimension(:), pointer` `typedefinitionsrovertopping::tpgeometry::ycoorddiff`

vector with differences in y-coordinates (m)

Definition at line 25 of file `typeDefinitionsRTOvertopping.f90`.

8.14.2.10 `real(wp), dimension(:), pointer` `typedefinitionsrovertopping::tpgeometry::ycoordinates`

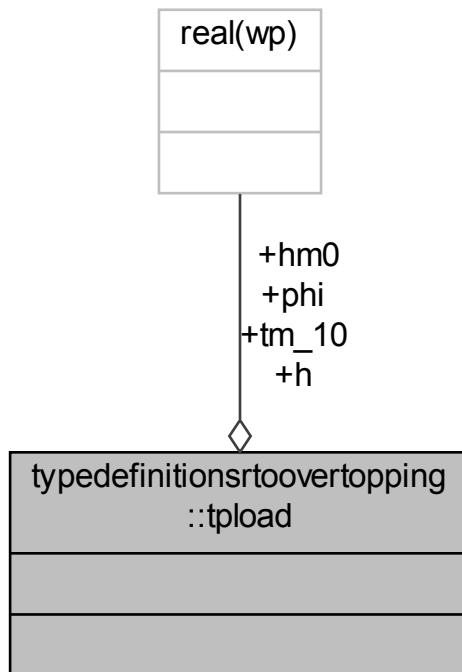
vector with y-coordinates cross section (m+NAP)

Definition at line 22 of file `typeDefinitionsRTOvertopping.f90`.

## 8.15 `typedefinitionsrovertopping::tpload` Type Reference

`tpLoad`: structure with load parameters

Collaboration diagram for `typedefinitionsrovertopping::tpload`:



### Public Attributes

- `real(wp) h`

*local water level (m+NAP)*

- real(wp) [hm0](#)

*significant wave height (m)*

- real(wp) [tm\\_10](#)

*spectral wave period (s)*

- real(wp) [phi](#)

*wave direction (degrees)*

### 8.15.1 Detailed Description

tpLoad: structure with load parameters

Definition at line 32 of file typeDefinitionsRTOvertopping.f90.

### 8.15.2 Member Data Documentation

#### 8.15.2.1 real(wp) typedefinitionsrtovertopping::tpload::h

local water level (m+NAP)

Definition at line 33 of file typeDefinitionsRTOvertopping.f90.

#### 8.15.2.2 real(wp) typedefinitionsrtovertopping::tpload::hm0

significant wave height (m)

Definition at line 34 of file typeDefinitionsRTOvertopping.f90.

#### 8.15.2.3 real(wp) typedefinitionsrtovertopping::tpload::phi

wave direction (degrees)

Definition at line 36 of file typeDefinitionsRTOvertopping.f90.

#### 8.15.2.4 real(wp) typedefinitionsrtovertopping::tpload::tm\_10

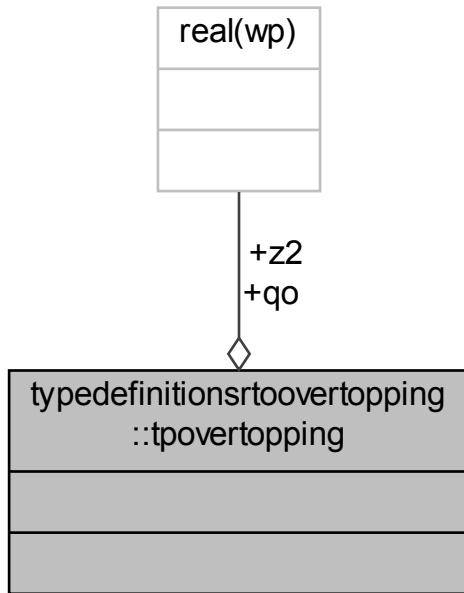
spectral wave period (s)

Definition at line 35 of file typeDefinitionsRTOvertopping.f90.

## 8.16 typedefinitionsrtovertopping::tpovertopping Type Reference

tpOvertopping: structure with overtopping results

Collaboration diagram for `typedefinitionsrtodovertopping::tpovertopping`:



## Public Attributes

- `real(wp) z2`  
2% wave run-up (m)
- `real(wp) qo`  
wave overtopping discharge (m<sup>3</sup>/m per s)

### 8.16.1 Detailed Description

`tpOvertopping`: structure with overtopping results

Definition at line 56 of file `typeDefinitionsRTOvertopping.f90`.

### 8.16.2 Member Data Documentation

#### 8.16.2.1 `real(wp) typedefinitionsrtodovertopping::tpovertopping::qo`

wave overtopping discharge (m<sup>3</sup>/m per s)

Definition at line 58 of file `typeDefinitionsRTOvertopping.f90`.

#### 8.16.2.2 `real(wp) typedefinitionsrtodovertopping::tpovertopping::z2`

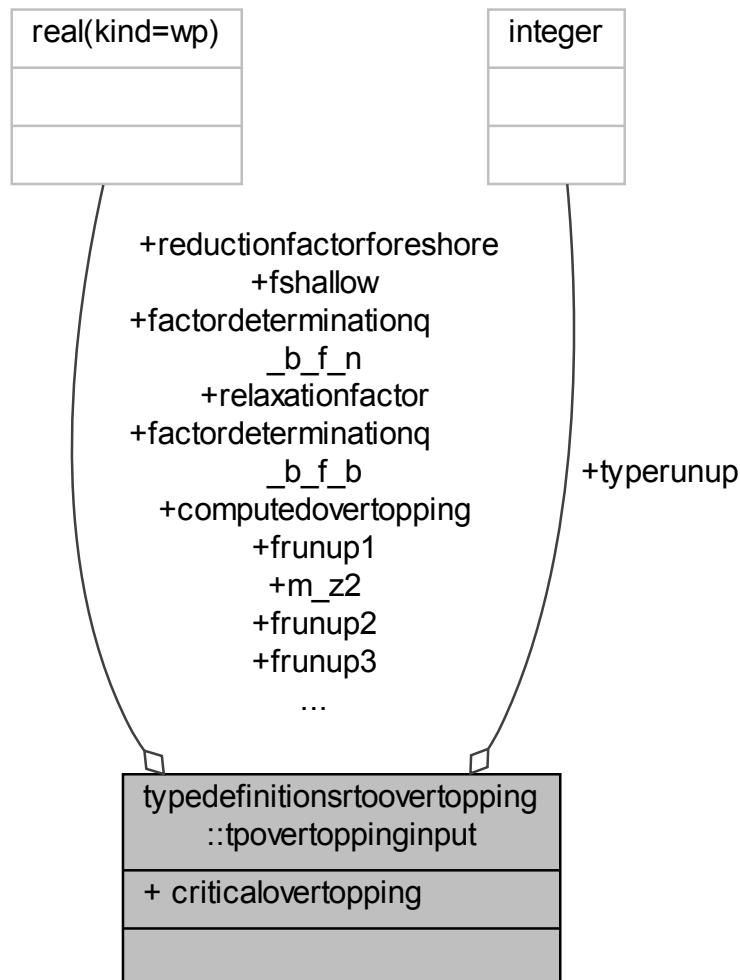
2% wave run-up (m)

Definition at line 57 of file `typeDefinitionsRTOvertopping.f90`.

## 8.17 `typedefinitionsrovertopping::tpovertoppinginput` Type Reference

OvertoppingModelFactors: C-structure with model factors.

Collaboration diagram for `typedefinitionsrovertopping::tpovertoppinginput`:



### Public Attributes

- `real(kind=wp) factordeterminationq_b_f_n`  
*model factor for non-breaking waves*
- `real(kind=wp) factordeterminationq_b_f_b`  
*model factor for breaking waves*
- `real(kind=wp) m_z2`  
*model factor describing the uncertainty of 2% runup height*
- `real(kind=wp) frunup1`  
*model factor 1 for wave run-up (for backwards compatibility)*
- `real(kind=wp) frunup2`

- `real(kind=wp) frunup3`  
*model factor 2 for wave run-up (idem)*
- `real(kind=wp) fshallow`  
*model factor for shallow waves*
- `real(kind=wp) computedovertopping`  
*model factor computed overtopping*
- `real(kind=wp) criticalovertopping`  
*model factor critical overtopping*
- `integer typerunup`  
*0: fRunup1, 2 and 3 are given; 1: m\_z2 is given*
- `real(kind=wp) relaxationfactor`  
*relaxation factor iteration procedure wave runup*
- `real(kind=wp) reductionfactorforeshore = 0.5_wp`  
*reduction factor foreshore*

### 8.17.1 Detailed Description

OvertoppingModelFactors: C-structure with model factors.

Definition at line 40 of file typeDefinitionsRTOvertopping.f90.

### 8.17.2 Member Data Documentation

#### 8.17.2.1 `real(kind=wp) typedefinitionsrovertopping::tpovertoppinginput::computedovertopping`

model factor computed overtopping

Definition at line 48 of file typeDefinitionsRTOvertopping.f90.

#### 8.17.2.2 `real(kind=wp) typedefinitionsrovertopping::tpovertoppinginput::criticalovertopping`

model factor critical overtopping

Definition at line 49 of file typeDefinitionsRTOvertopping.f90.

#### 8.17.2.3 `real(kind=wp) typedefinitionsrovertopping::tpovertoppinginput::factordeterminationq_b_f_b`

model factor for breaking waves

Definition at line 42 of file typeDefinitionsRTOvertopping.f90.

#### 8.17.2.4 `real(kind=wp) typedefinitionsrovertopping::tpovertoppinginput::factordeterminationq_b_f_n`

model factor for non-breaking waves

Definition at line 41 of file typeDefinitionsRTOvertopping.f90.

#### 8.17.2.5 `real(kind=wp) typedefinitionsrovertopping::tpovertoppinginput::frunup1`

model factor 1 for wave run-up (for backwards compatibility)

Definition at line 44 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.6 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::frunup2**

model factor 2 for wave run-up (idem)

Definition at line 45 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.7 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::frunup3**

model factor 3 for wave run-up (idem)

Definition at line 46 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.8 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::fshallow**

model factor for shallow waves

Definition at line 47 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.9 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::m\_z2**

model factor describing the uncertainty of 2% runup height

Definition at line 43 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.10 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::reductionfactorforeshore = 0.5\_wp**

reduction factor foreshore

Definition at line 52 of file typeDefinitionsRTOvertopping.f90.

**8.17.2.11 real(kind=wp) typedefinitionsrtovertopping::tpovertoppinginput::relaxationfactor**

relaxation factor iteration procedure wave runup

Definition at line 51 of file typeDefinitionsRTOvertopping.f90.

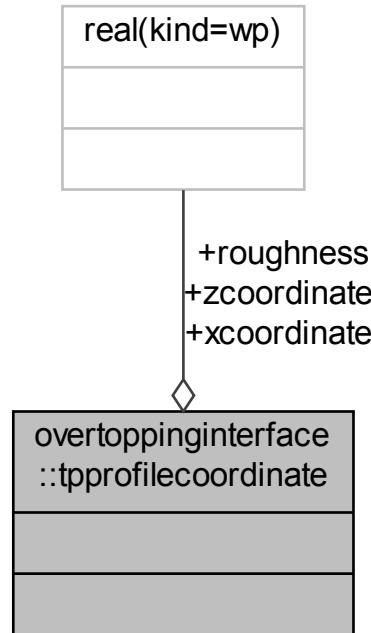
**8.17.2.12 integer typedefinitionsrtovertopping::tpovertoppinginput::typerunup**

0: fRunup1, 2 and 3 are given; 1: m\_z2 is given

Definition at line 50 of file typeDefinitionsRTOvertopping.f90.

## 8.18 overtoppinginterface::tpprofilecoordinate Type Reference

Collaboration diagram for overtoppinginterface::tpprofilecoordinate:



### Public Attributes

- `real(kind=wp) xcoordinate`  
*X-coordinate foreland profile.*
- `real(kind=wp) zcoordinate`  
*Z-coordinate foreland profile.*
- `real(kind=wp) roughness`  
*Roughness of the area between two points.*

### 8.18.1 Detailed Description

Definition at line 19 of file `overtoppingInterface.f90`.

### 8.18.2 Member Data Documentation

#### 8.18.2.1 `real(kind=wp) overtoppinginterface::tpprofilecoordinate::roughness`

Roughness of the area between two points.

Definition at line 22 of file `overtoppingInterface.f90`.

### 8.18.2.2 real(kind=wp) overtoppinginterface::tpprofilecoordinate::xcoordinate

X-coordinate foreland profile.

Definition at line 20 of file overtoppingInterface.f90.

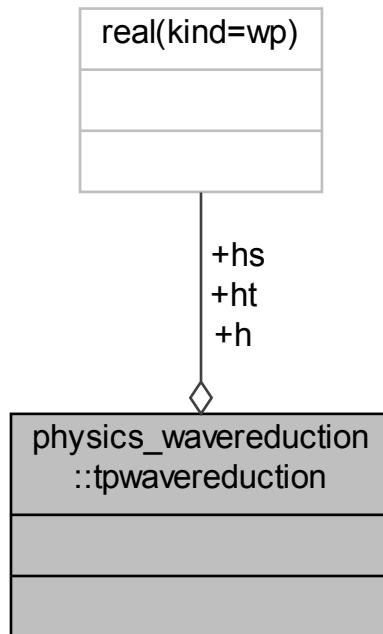
### 8.18.2.3 real(kind=wp) overtoppinginterface::tpprofilecoordinate::zcoordinate

Z-coordinate foreland profile.

Definition at line 21 of file overtoppingInterface.f90.

## 8.19 physics\_wavereduction::tpwavereduction Type Reference

Collaboration diagram for physics\_wavereduction::tpwavereduction:



### Public Attributes

- real(kind=wp) **hs**  
*wave height*
- real(kind=wp) **h**  
*local waterlevel*
- real(kind=wp) **ht**  
*level (height) of the toe of the dike*

### 8.19.1 Detailed Description

Definition at line 22 of file waveReduction.f90.

### 8.19.2 Member Data Documentation

#### 8.19.2.1 real (kind=wp) physics\_wavereduction::tpwavereduction::h

local waterlevel

Definition at line 24 of file waveReduction.f90.

#### 8.19.2.2 real (kind=wp) physics\_wavereduction::tpwavereduction::hs

wave height

Definition at line 23 of file waveReduction.f90.

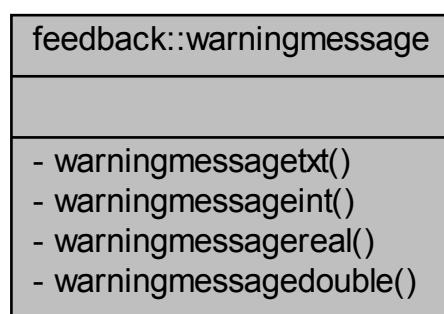
#### 8.19.2.3 real (kind=wp) physics\_wavereduction::tpwavereduction::ht

level (height) of the toe of the dike

Definition at line 25 of file waveReduction.f90.

## 8.20 feedback::warningmessage Interface Reference

Collaboration diagram for feedback::warningmessage:



### Private Member Functions

- subroutine [warningmessagetext](#) (msgtext)  
*Write a warning message.*
- subroutine [warningmessageint](#) (msgtext, value)  
*Write a warning message with an integer value.*
- subroutine [warningmessagereal](#) (msgtext, value)

*Write a warning message with a real value.*

- subroutine [warningmessagedouble](#) (*msgtext*, *value*)

*Write a warning message with a double precision real value.*

### 8.20.1 Detailed Description

Definition at line 156 of file feedback.f90.

### 8.20.2 Member Function/Subroutine Documentation

8.20.2.1 subroutine **feedback::warningmessage::warningmessagedouble** ( character(len=\*)*msgtext*, real(kind=dp), intent(in) *value* ) [private]

Write a warning message with a double precision real value.

#### Parameters

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Real (double) value to add

Definition at line 729 of file feedback.f90.

8.20.2.2 subroutine **feedback::warningmessage::warningmessageint** ( character(len=\*)*msgtext*, integer, intent(in) *value* ) [private]

Write a warning message with an integer value.

#### Parameters

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Integer value to add

Definition at line 699 of file feedback.f90.

8.20.2.3 subroutine **feedback::warningmessage::warningmessagereal** ( character(len=\*)*msgtext*, real, intent(in) *value* ) [private]

Write a warning message with a real value.

#### Parameters

in	<i>msgtext</i>	Error message to write
in	<i>value</i>	Real value to add

Definition at line 714 of file feedback.f90.

8.20.2.4 subroutine **feedback::warningmessage::warningmessagetxt** ( character(len=\*)*msgtext* ) [private]

Write a warning message.

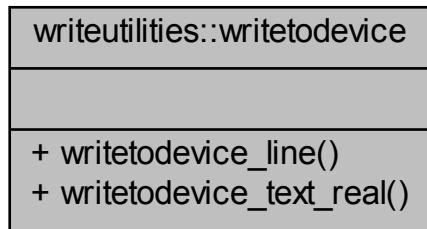
#### Parameters

in	<i>msgtext</i>	Error message to write
----	----------------	------------------------

Definition at line 676 of file feedback.f90.

## 8.21 writeutilities::writetodevice Interface Reference

Collaboration diagram for writeutilities::writetodevice:



### Public Member Functions

- subroutine [writetodevice\\_line](#) (fileId, line)  
*Write to device.*
- subroutine [writetodevice\\_text\\_real](#) (fileId, formatText, textView, realValue)  
*Write to device.*

#### 8.21.1 Detailed Description

Definition at line 14 of file writeUtilities.f90.

#### 8.21.2 Member Function/Subroutine Documentation

##### 8.21.2.1 subroutine writeutilities::writetodevice::writetodevice\_line ( integer, intent(in) fileId, character(len=\*) , intent(in) line )

Write to device.

###### Parameters

in	<i>fileId</i>	device number to write to, 0 = screen
in	<i>line</i>	text to write

Definition at line 28 of file writeUtilities.f90.

##### 8.21.2.2 subroutine writeutilities::writetodevice::writetodevice\_text\_real ( integer, intent(in) fileId, character(len=\*) , intent(in) formatText, character(len=\*) , intent(in) textView, real (kind=wp), intent(in) realValue )

Write to device.

###### Parameters

in	<i>fileId</i>	device number to write to, 0 = screen
----	---------------	---------------------------------------

in	<i>formattext</i>	format specifier
in	<i>textvalue</i>	text to write
in	<i>realvalue</i>	real value to print

Definition at line 42 of file writeUtilities.f90.

# Chapter 9

## File Documentation

### 9.1 angleUtilities.f90 File Reference

Implement various utility functions dealing with directions.

#### Modules

- module `angleutilities`  
*Module with direction utility functions.*

#### Functions/Subroutines

- real(kind=wp) function `angleutilities::anglebetween2directions` (`direction1, direction2`)  
*Function angleBetween2Directions() for the computation of the angle between two directions.  
The two directions must be given in degrees and within [0, 360]  
The computed angle between these two directions is also in degrees, and within [0, 180]*

#### 9.1.1 Detailed Description

Implement various utility functions dealing with directions.

### 9.2 breakwater.f90 File Reference

Breakwater formulas.

#### Modules

- module `physics_breakwater`  
*Module with routines for breakwater.*

#### Functions/Subroutines

- subroutine, public `physics_breakwater::calculatebreakwater` (`breakwaterType, heightBreakwater, waterLevel, Hs`)  
*Function to transform the wave height for the influence of an occupant breakwater.*

### 9.2.1 Detailed Description

Breakwater formulas.

## 9.3 breakwaterEnumerations.f90 File Reference

Module for enumerating breakwater types.

### Modules

- module [physics\\_breakwatertypesenumerations](#)

*Enumeration for breakwater types WHEN CHANGING THESE VALUES. ALSO CHANGE THEM IN HydraRing←Enumerations.CS.*

### Variables

- integer, parameter [physics\\_breakwatertypesenumerations::breakwaternotpresent](#) = 0  
*no breakwater*
- integer, parameter [physics\\_breakwatertypesenumerations::breakwatercaisson](#) = 1  
*breakwater type is caisson*
- integer, parameter [physics\\_breakwatertypesenumerations::breakwaterverticalwall](#) = 2  
*breakwater type is vertical wall*
- integer, parameter [physics\\_breakwatertypesenumerations::breakwaterrubblemound](#) = 3  
*breakwater type is rubble mound 1:1.5*

### 9.3.1 Detailed Description

Module for enumerating breakwater types.

## 9.4 bretschneider.f90 File Reference

Compute wave parameters using Bretschneider formulas.

### Data Types

- type [physics\\_bretschneider::tpbretschneider](#)

### Modules

- module [physics\\_bretschneider](#)

*Module with routines for bretschneider.*

### Functions/Subroutines

- subroutine, public [physics\\_bretschneider::calculatewavebretschneider](#) ([bretschneider](#), [Hs](#), [Ts](#))

*Function to compute wave parameters according to Bretschneider formula: wave height (Hs) wave period (Ts)*

### 9.4.1 Detailed Description

Compute wave parameters using Bretschneider formulas.

## 9.5 conversionFunctions.f90 File Reference

### Modules

- module [conversionfunctions](#)

### Functions/Subroutines

- subroutine, public [conversionfunctions::pqfrombeta](#) (beta, p, q)

*Subroutine for computing the exceedance and non-exceedance probabilities for a given reliability index (beta) assuming a standard normal distribution.*

- real(kind=wp) function, public [conversionfunctions::pfrombeta](#) (beta)

*Function for computing the non-exceedance probability (P) for a given reliability index (beta), assuming a standard normal distribution.*

- real(kind=wp) function, public [conversionfunctions::qfrombeta](#) (beta)

*Function for computing the exceedance probability (Q) for a given reliability index (beta), assuming a standard normal distribution.*

- subroutine, public [conversionfunctions::returntimefrombeta](#) (beta, returnTime)

*Subroutine for computing the return time for a given reliability index (beta) assuming a standard normal distribution.*

- subroutine, public [conversionfunctions::freqfrombeta](#) (beta, freq)

*Subroutine for computing the frequency for a given reliability index (beta) assuming a standard normal distribution.*

- subroutine, public [conversionfunctions::logqfrombeta](#) (beta, logQ)

*Subroutine for computing the (negative) logarithm of the non-exceedance probability for a given reliability index (beta) assuming a standard normal distribution.*

- subroutine, public [conversionfunctions::betafromq](#) (q, beta)

*Subroutine for computing the reliability index (beta) for a given exceedance probability. The reliability index is corrected with the subroutine PQfromBeta.*

### Variables

- real(kind=wp), parameter, private [conversionfunctions::qmin](#) = 1.0d-300
- real(kind=wp), parameter, private [conversionfunctions::upperlog](#) = 700.0d0
- real(kind=wp), parameter, private [conversionfunctions::ulimit](#) = 5.6d0

## 9.6 dllOvertopping.f90 File Reference

Main entry for the dll DikesOvertopping FUNCTIONS/SUBROUTINES exported from dllOvertopping.dll:

### Modules

- module [dllovertopping](#)

*Calculate one type of overtopping.*

## Functions/Subroutines

- subroutine, public [dlovertopping::calculateqo](#) (load, geometryInput, dikeHeight, modelFactors, overtopping, success, errorText, verbosity, logFile)
 

*Subroutine that calculates the discharge needed for the Z-function DikesOvertopping Wrapper for calculateQoF: convert C-like input structures to Fortran input structures.*
- subroutine, public [dlovertopping::calculateqqf](#) (load, geometryF, dikeHeight, modelFactors, overtopping, success, errorText, logging)
 

*Subroutine that calculates the discharge needed for the Z-function DikesOvertopping.*
- subroutine, public [dlovertopping::calcvalue](#) (criticalOvertoppingRate, modelFactors, Qo, z, success, errorMessage)
 

*Subroutine that calculates the Z-function DikesOvertopping based on the discharge calculated with calculateQoF.*
- subroutine, public [dlovertopping::validateinputc](#) (geometryInput, dikeHeight, modelFactors, success, errorText)
 

*Subroutine that validates the geometry Wrapper for ValidateInputC: convert C-like input structures to Fortran input structures.*
- subroutine, public [dlovertopping::validateinputf](#) (geometryF, dikeHeight, modelFactors, success, errorText)
 

*Subroutine that validates the geometry.*
- subroutine, public [dlovertopping::versionnumber](#) (version)
 

*Subroutine that delivers the version number.*
- type(overtoppinggeometrytypef) function [dlovertopping::geometry\\_c\\_f](#) (geometryInput)
 

*Private subroutine that converts geometry from c-pointer to fortran struct.*

### 9.6.1 Detailed Description

Main entry for the dll DikesOvertopping FUNCTIONS/SUBROUTINES exported from dllOvertopping.dll:

- zFuncOvertopping
- calculateQo
- calculateQoF
- ValidateInputC
- ValidateInputF
- versionNumber

## 9.7 equalReals.f90 File Reference

This file contains a module with functions to test quality of reals.

### Modules

- module [equalreals](#)

*Module with functions to test quality of reals.*

## Functions/Subroutines

- logical function `equalreals::equalrealsrelative` (x, y, margin)
 

*equalRealsRelative Function to test for equality of two reals with a relative value for the margin equalRealsRelative =  $|x-y| \leqslant \text{margin} * (|x|+|y|)/2$*
- logical function `equalreals::equalrealsabsolute` (x, y, margin)
 

*equalRealsAbsolute Function to test for equality of two reals with an absolute value for the margin equalRealsAbsolute =  $|x-y| \leqslant \text{margin}$*
- logical function `equalreals::equalrealsinvector` (n, vector, margin)
 

*Function to test for equality of reals in a vector with a relative value of the margin.*
- logical function `equalreals::equalvectorswithreals` (n, vector1, vector2, margin, equality)
 

*Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.*
- logical function `equalreals::equaltableswithreals` (matrix1, matrix2, margin)
 

*Function to test for equality of two vectors with reals Two vectors are equal if all elements in the vector are equal.*
- logical function `equalreals::equalvectorswithrealsweightedmargin` (n, vector1, vector2, margin, equality)
 

*Function to test for equality of two vectors with reals. Two vectors are equal if all elements in the vector are almost equal. Here, margin is weighted according to tested numbers.*
- logical function `equalreals::equalvectorswithintegers` (n, vector1, vector2, equality)

### 9.7.1 Detailed Description

This file contains a module with functions to test quality of reals.

## 9.8 factorModuleRTOvertopping.f90 File Reference

This file contains a module with functions for the slope angle and influence factors.

## Modules

- module `factormodulertooverlapping`

## Functions/Subroutines

- subroutine, public `factormodulertooverlapping::calculatetanalpha` (h, Hm0, z2, geometry, tanAlpha, succes, errorMessage)
 

*calculateTanAlpha representative slope angle*
- subroutine, public `factormodulertooverlapping::calculategammabeta` (Hm0, Tm\_10, beta, gammaBeta\_← z, gammaBeta\_o)
 

*calculateGammaBeta influence factor angle of wave attack*
- subroutine, public `factormodulertooverlapping::calculategammaf` (h, ksi0, ksi0Limit, gammaB, z2, geometry, gammaF, succes, errorMessage)
 

*calculateGammaF influence factor roughness*
- subroutine, public `factormodulertooverlapping::calculategammab` (h, Hm0, z2, geometry, NbermSegments, gammaB, succes, errorMessage)
 

*calculateGammaB influence factor berms*

### 9.8.1 Detailed Description

This file contains a module with functions for the slope angle and influence factors.

## 9.9 feedback.f90 File Reference

Feedback library: error messaging and progress reporting.

### Data Types

- interface `feedback::feedbackcontext`
- interface `feedback::errormessage`
- interface `feedback::warningmessage`
- interface `feedback::fatalerror`
- interface `feedback::progressreport`

### Modules

- module `feedback`

### Functions/Subroutines

- integer function `feedback::p ()`

*Returns the pointer to the feedback dll, avoiding many loadlibrary calls.*
- logical function, public `feedback::todll ()`

*Is feedback writing to the LogMessage function in HydraRing.IO.Native.dll ?*
- logical function, public `feedback::fatalerroroccurred (value)`

*Has a fatal error occurred? Mainly used in unit tests.*
- subroutine, public `feedback::resetfatalerroroccurred ()`

*Reset fatal error occurrence Mainly used in unit tests.*
- subroutine `feedback::setfatalerroroccurred (value)`

*Set fatal error occurrence Mainly used in unit tests.*
- logical function, public `feedback::getfatalerrorexpected (value)`

*Get value of FatalErrorExpected Mainly used in unit tests.*
- subroutine, public `feedback::setfatalerrorexpected (value)`

*Set the value of FatalErrorExpected Mainly used in unit tests.*
- logical function, public `feedback::allow_log ()`

*Is logging active?*
- subroutine, public `feedback::setallowlogging (value)`

*Set logging on/off.*
- subroutine, public `feedback::getthreadid (threadId)`

*Get the tread ID Note that multi threading is not implemented, so always the value 0 is returned.*
- subroutine, public `feedback::setfatalerroraction (action)`

*Set the value of FatalErrorAction possible values: onfatalerrorStop = 1 : Action on fatal error: exit the program onfatalerrorThrowException = 2 : Action on fatal error: throw exception (for use as dll) onfatalerrorUnittest = 3 : Action on fatal error: handle as unit test.*
- subroutine, public `feedback::getfatalerrormessage (message)`

*Get the last of FatalErrorMessage.*
- subroutine, public `feedback::feedbackerror (error)`

*Return the error status of the current thread If an error has been reported (via the errorMessage routines), this routine will return true, otherwise false.*
- subroutine, public `feedback::feedbackclose`

*Close the log file.*
- subroutine, public `feedback::feedbackinitialise (output, logfile, verbosity)`

*Initialise the feedback subsystem.*

- subroutine `feedback::progressreporttxt` (level, msgtext)  
*Write a single message to the output for the benefit of the user.*
- subroutine `feedback::progressreportint` (level, msgtext, value)  
*Write a single message with an integer value.*
- subroutine `feedback::progressreportreal` (level, msgtext, value)  
*Write a single message with a real value.*
- subroutine `feedback::progressreportdouble` (level, msgtext, value)  
*Write a single message with a double precision real value.*
- subroutine `feedback::progressreportarray` (level, msgText, x, size)  
*Print array values to report file.*
- subroutine `feedback::progressreportvalueandarray` (level, msgText, z, x, size)  
*Print z-value and array values to report file.*
- subroutine, public `feedback::feedbackenter` (routine)  
*Register a new routine level.*
- subroutine, public `feedback::feedbackleave`  
*Drop a routine level from the stack trace.*
- subroutine, public `feedback::feedbackprintstack`  
*Print the stack trace as known via feedbackEnter and feedbackLeave.*
- subroutine `feedback::errormessagetxt` (msgtext)  
*Write an error message.*
- subroutine `feedback::errormessageint` (msgtext, value)  
*Write an error message with an integer value.*
- subroutine `feedback::errormessagereal` (msgtext, value)  
*Write an error message with a real value.*
- subroutine `feedback::errormessagedouble` (msgtext, value)  
*Write an error message with a double precision real value.*
- subroutine `feedback::warningmessagetxt` (msgtext)  
*Write a warning message.*
- subroutine `feedback::warningmessageint` (msgtext, value)  
*Write a warning message with an integer value.*
- subroutine `feedback::warningmessagereal` (msgtext, value)  
*Write a warning message with a real value.*
- subroutine `feedback::warningmessagedouble` (msgtext, value)  
*Write a warning message with a double precision real value.*
- subroutine `feedback::fatalerrortxt` (msgtext)  
*Write a fatal error message.*
- subroutine `feedback::fatalerrorint` (msgtext, value)  
*Write a fatal error message with an integer value.*
- subroutine `feedback::fatalerrorreal` (msgtext, value)  
*Write a fatal error message with a real value.*
- subroutine `feedback::fatalerrordouble` (msgtext, value)  
*Write a fatal error message with a double precision real value.*
- subroutine, public `feedback::feedbackdropcontext`  
*Drop the entire context.*
- subroutine, public `feedback::feedbackprintcontext`  
*Register a context for error messages.*
- subroutine `feedback::feedbackcontexttxt` (text)  
*Register a context for error messages.*
- subroutine `feedback::feedbackcontextint` (text, value)  
*Register a context for error messages - with integer value.*
- subroutine `feedback::feedbackcontextreal` (text, value)

- subroutine **feedback::feedbackcontextdouble** (text, value)
 

*Register a context for error messages - with real value.*
- subroutine, public **feedback::throw** (msgText)
 

*Stop execution by throwing an exception Useful if running in a dll and main program is a GUI in e.g. C++ or C#.*

## Variables

- integer, parameter **feedback::id** = 0
 

*Thread ID; multi threading is not supported yet!*
- integer, parameter **feedback::sp** = kind(1.0)
- integer, parameter **feedback::dp** = kind(1.0d0)
- integer, parameter **feedback::wp** = kind(1.0d0)

### 9.9.1 Detailed Description

Feedback library: error messaging and progress reporting.

## 9.10 feedback\_parameters.f90 File Reference

Feedback parameters: parameters needed bij feedback and feedback\_dll.

## Modules

- module **feedback\_parameters**

## Variables

- integer, parameter **feedback\_parameters::onfatalerrorstop** = 1
 

*Action on fatal error: exit the program.*
- integer, parameter **feedback\_parameters::onfatalerrorthrowexception** = 2
 

*Action on fatal error: throw exception (for use as dll)*
- integer, parameter **feedback\_parameters::onfatalerrorunittest** = 3
 

*Action on fatal error: handle as unit test.*
- integer, parameter **feedback\_parameters::onlyscreen** = 0
 

*Mode: write to screen only.*
- integer, parameter **feedback\_parameters::onlyfile** = 1
 

*Mode: write to file only.*
- integer, parameter **feedback\_parameters::screenandfile** = 2
 

*Mode: write to both screen and file.*
- integer, parameter **feedback\_parameters::dll** = 3
 

*Mode: write to dll.*
- integer, parameter **feedback\_parameters::verbosenone** = -1
 

*Reporting: NO messages.*
- integer, parameter **feedback\_parameters::verbosebasic** = 0
 

*Reporting: basic messages only.*
- integer, parameter **feedback\_parameters::verbosetailed** = 1
 

*Reporting: detailed messages (including traceback)*
- integer, parameter **feedback\_parameters::verbosedebugging** = 2
 

*Reporting: all messages.*

### 9.10.1 Detailed Description

Feedback parameters: parameters needed bij feedback and feedback\_dll.

## 9.11 feedbackDLL.f90 File Reference

Feedback dll: error messaging and progress reporting.

### Data Types

- type [feedbackdll::feedbackdata](#)

### Modules

- module [feedbackdll](#)

### Functions/Subroutines

- subroutine [feedbackdll::initialise](#) (output, logfile, verbosity)  
*Initialise the feedback dll.*
- subroutine [feedbackdll::printmessage](#) (msgtext)  
*Write a message to output device (as selected in Initialise)*
- subroutine [feedbackdll::outputtodll](#) (active)  
*Is output is to another dll?*
- subroutine [feedbackdll::getallowlogging](#) (value)  
*Is logging active.*
- subroutine [feedbackdll::setallowlogging](#) (value)  
*Set logging on/off.*
- subroutine [feedbackdll::getverbosity](#) (level)  
*Get the current verbosity level.*
- subroutine [feedbackdll::setonfatalerroraction](#) (action)  
*Set the OnFatalErrorAction.*
- subroutine [feedbackdll::getonfatalerroraction](#) (onfatalerrorAction)  
*Get the current OnFatalErrorAction.*
- subroutine [feedbackdll::getfeedbackerror](#) (error)  
*Get the error status (true/false)*
- subroutine [feedbackdll::setfatalerrormessage](#) (message)  
*Set the errormessage.*
- subroutine [feedbackdll::getfatalerrormsg](#) (message)  
*Get the last error message.*
- subroutine [feedbackdll::getverbosylevel](#) (verbosityLevel)  
*Get the verbosity level.*
- subroutine [feedbackdll::pushroutine](#) (routine, success)  
*Add a routine name to the call stack.*
- subroutine [feedbackdll::poproutine](#) (success)  
*Removes and get a routine name from the call stack.*
- subroutine [feedbackdll::printstack](#)  
*Print the call stack.*
- subroutine [feedbackdll::getstacklevel](#) (level)

- subroutine **feedbackdll::closefile**  
*Closes the output file.*
- subroutine **feedbackdll::getunitnumber** (lun)  
*Get the unit number of the logfile.*
- subroutine **feedbackdll::dropcontext**  
*Reset the context stack.*
- subroutine **feedbackdll::printcontext**  
*Print the context stack.*
- subroutine **feedbackdll::contexttxt** (text, success)  
*Add a line of text to the context stack.*
- subroutine **feedbackdll::getsetfatalerrorexpected** (func, value)  
*Get or Set the FatalErrorExpected if func=get: get the value if func=set: set the value.*
- subroutine **feedbackdll::getsetfatalerroroccurred** (func, value)  
*Get or Set the FatalErrorOccured if func=get: get the value if func=set: set the value.*
- subroutine **feedbackdll::writetodll** (msgText)  
*Write message to another dll.*

## Variables

- type(feedbackdata) **feedbackdll::fbdata**  
*Array holding the actual information per thread (multiple threads not yet implemented)*
- integer **feedbackdll::fatalerroraction** = 1
- character(len=255) **feedbackdll::fatalerrormessage**
- integer, parameter **feedbackdll::id** = 0
- logical **feedbackdll::allow\_logging** = .true.
- logical **feedbackdll::isfatalerrorexpected** = .false.  
*if isFatalErrorExpected = .true. then log error message in fatalErrorMsgText and set boolean isFatalErrorOccured to true if isFatalErrorExpected = .false. then same action as onfatalerrorStop*
- logical **feedbackdll::isfatalerroroccurred** = .false.

### 9.11.1 Detailed Description

Feedback dll: error messaging and progress reporting.

To be used in combination with the feedback module. feedback is the (fortran) interface to the programmer. This module make it possible to write to the same file from different dll's/exe's.

By putting all data and print statements in this dll, and let feedback initialise this dll and let feedback only print using this dll, all output ends up at the right location.

## 9.12 fileUtilities.f90 File Reference

Module for reading information from files.

## Modules

- module **fileutilities**

## Functions/Subroutines

- subroutine `fileutilities::readtable` (filename, nColumns, table)
- subroutine `fileutilities::writetablevalues` (filename, table)
- subroutine `fileutilities::removefile` (filename)

*Routine to throw away a file.*

### 9.12.1 Detailed Description

Module for reading information from files.

## 9.13 foreland.f90 File Reference

### Modules

- module `physics_foreland`  
*wrapper for foreland dll !!*

## Functions/Subroutines

- subroutine, public `physics_foreland::physics_calculateforeland` (retval, hDamType, hDamHeight, hAlpha, hFc, hInvalid, hDimHm0, hHm0, hTp, hWlev, hIncomingWaveAngle, hDikeNormal, hDimX, hX, hMinStepSize, hBottomLevel, hRatioDepth, hLogging, hLoggingFileName, hHm0Dike, hTpDike, hRefractedWaveAngleDike, hMessage)

*wrapper for foreland calculation in DynamicLib-DaF.dll*

### Variables

- integer, parameter `physics_foreland::message_length` =1000

## 9.14 formulaModuleRTOvertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

### Modules

- module `formulamodulertoovertopping`

## Functions/Subroutines

- subroutine, public `formulamodulertoovertopping::calculatewaverunup` (Hm0, ksi0, ksi0Limit, gammaB, gammaF, gammaBeta, modelFactors, z2, succes, errorMessage)  
*calculateWaveRunup: calculate wave runup*
- subroutine, public `formulamodulertoovertopping::calculatewaveovertoppingdischarge` (h, Hm0, tanAlpha, gammaB, gammaF, gammaBeta, ksi0, hCrest, modelFactors, Qo, succes, errorMessage)  
*calculateWaveOvertoppingDischarge: calculate the wave overtopping discharge*
- subroutine, public `formulamodulertoovertopping::calculatewavelength` (Tm\_10, L0)  
*calculateWaveLength: calculate the wave length*

- subroutine, public `formulamodulertoovertopping::calculatewavesteeppness` (`Hm0, Tm_10, s0, succes, errorMessage`)
 

*calculateWaveSteepness: calculate the wave steepness*
- subroutine, public `formulamodulertoovertopping::calculatebreakerparameter` (`tanAlpha, s0, ksi0, succes, errorMessage`)
 

*calculateBreakerParameter: calculate the breaker parameter*
- subroutine, public `formulamodulertoovertopping::calculateanglewaveattack` (`phi, psi, beta`)
 

*calculateAngleWaveAttack: calculate the angle of wave attack*
- subroutine, public `formulamodulertoovertopping::calculatebreakerlimit` (`modelFactors, gammaB, ksi0Limit, succes, errorMessage`)
 

*calculateBreakerLimit: calculate the breaker limit*
- subroutine, public `formulamodulertoovertopping::adjustinfluencefactors` (`gammaB, gammaF, gammaBeta, gammaBetaType, ksi0, ksi0Limit, succes, errorMessage`)
 

*adjustInfluenceFactors: adjust the influence factors*
- subroutine, public `formulamodulertoovertopping::realrootscubicfunction` (`a, b, c, d, N, x, succes, errorMessage`)
 

*realRootsCubicFunction: calculate the roots of a cubic function*
- subroutine, public `formulamodulertoovertopping::rootsgeneralcubic` (`a, b, c, d, z, succes, errorMessage`)
 

*rootsGeneralCubic: calculate the roots of a generic cubic function*
- subroutine, public `formulamodulertoovertopping::rootsdepressedcubic` (`p, q, z`)
 

*rootsDepressedCubic: calculate the roots of a depressed cubic function*
- subroutine, public `formulamodulertoovertopping::cubicroots` (`z, roots`)
 

*cubicRoots: calculate the roots of a cubic function*
- logical function, public `formulamodulertoovertopping::isequalreal` (`x1, x2`)
 

*isEqualReal: are two reals (almost) equal*
- logical function, public `formulamodulertoovertopping::isequalzero` (`x`)
 

*isEqualZero: is a real (almost) zero*

### 9.14.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

## 9.15 ftnunit.f90 File Reference

### Data Types

- interface `ftnunit::proc`
- interface `ftnunit::assert_equal`
- interface `ftnunit::assert_comparable`
- interface `ftnunit::assert_inbetween`

### Modules

- module `ftnunit`

## Functions/Subroutines

- subroutine, public `ftnunit::setruntestlevel` (`runtTestLevel`)  
*Set the run test level 0 = fast tests 1.. 2.. 3 = slow tests.*
- subroutine, public `ftnunit::settesttitle` (`message`)  
*Set the title that is displayed in the report of the test run.*
- subroutine, public `ftnunit::test` (`proc, text, ignore`)
- subroutine, public `ftnunit::testwithlevel` (`proc, text, testlevel, ignore`)
- subroutine, public `ftnunit::runtests_init`
- subroutine `ftnunit::runtests_init_priv`
- subroutine, public `ftnunit::runtests_final` (`stop`)
- logical function `ftnunit::issilent` ()
- subroutine `ftnunit::setworkingdir`
- subroutine, public `ftnunit::runtests` (`testproc`)
- subroutine `ftnunit::expect_program_stop`
- subroutine, public `ftnunit::assert_true` (`cond, text`)
- subroutine, public `ftnunit::assert_false` (`cond, text`)
- subroutine `ftnunit::assert_equal_logical` (`value1, value2, text`)
- subroutine `ftnunit::assert_equal_logical1d` (`array1, array2, text`)
- subroutine `ftnunit::assert_equal_string` (`value1, value2, text`)
- subroutine `ftnunit::assert_equal_string1d` (`array1, array2, text`)
- subroutine `ftnunit::assert_equal_int` (`value1, value2, text`)
- subroutine `ftnunit::assert_equal_int1d` (`array1, array2, text`)
- subroutine `ftnunit::assert_comparable_real` (`value1, value2, margin, text`)
- subroutine `ftnunit::assert_comparable_real1d` (`array1, array2, margin, text`)
- subroutine `ftnunit::assert_comparable_real2d` (`array1, array2, margin, text`)
- subroutine `ftnunit::assert_comparable_double` (`value1, value2, margin, text`)
- subroutine `ftnunit::assert_comparable_double1d` (`array1, array2, margin, text`)
- subroutine `ftnunit::assert_comparable_double2d` (`array1, array2, margin, text`)
- subroutine `ftnunit::assert_inbetween_real` (`value, vmin, vmax, text`)
- subroutine `ftnunit::assert_inbetween_double` (`value, vmin, vmax, text`)
- subroutine, public `ftnunit::assert_files_comparable` (`filename1, filename2, text, tolerance`)
- subroutine `write_header`
- logical function `ftnunit::ftnunit_file_exists` (`filename`)
- subroutine `ftnunit::ftnunit_remove_file` (`filename`)
- subroutine `ftnunit::ftnunit_make_empty_file` (`filename`)
- subroutine `ftnunit::ftnunit_write_html_header`
- subroutine `ftnunit::ftnunit_write_html_footer`
- subroutine `ftnunit::colour_number` (`number, coloured, colour1, colour2, allowed`)
- subroutine `ftnunit::ftnunit_write_html_test_begin` (`text`)
- subroutine `ftnunit::ftnunit_write_html_ignored` (`lun`)
- subroutine `ftnunit::ftnunit_write_html_cpu` (`cpu`)
- subroutine `ftnunit::ftnunit_write_html_previous_failed`
- subroutine `ftnunit::ftnunit_write_html_previous_stopped`
- subroutine `ftnunit::ftnunit_write_html_close_row` (`lun`)
- subroutine `ftnunit::ftnunit_write_html_failed_logic` (`text, expected`)
- subroutine `ftnunit::ftnunit_write_html_failed_equivalent` (`text`)
- subroutine `ftnunit::ftnunit_write_html_failed_equivalent1d` (`text, idx, value1, value2, addtext`)
- subroutine `ftnunit::ftnunit_write_html_failed_string` (`text, value1, value2`)
- subroutine `ftnunit::ftnunit_write_html_failed_string1d` (`text, idx, value1, value2, addtext`)
- subroutine `ftnunit::ftnunit_write_html_failed_int` (`text, value1, value2`)
- subroutine `ftnunit::ftnunit_write_html_failed_int1d` (`text, idx, value1, value2, addtext`)
- subroutine `ftnunit::ftnunit_write_html_failed_real` (`text, value1, value2`)
- subroutine `ftnunit::ftnunit_write_html_failed_real1d` (`text, idx, value1, value2, addtext`)

- subroutine `ftnunit::ftnunit_write_html_failed_real2d` (text, idx1, idx2, value1, value2, addtext)
- subroutine `ftnunit::ftnunit_write_html_failed_double` (text, value1, value2)
- subroutine `ftnunit::ftnunit_write_html_failed_double1d` (text, idx, value1, value2, addtext)
- subroutine `ftnunit::ftnunit_write_html_failed_double2d` (text, idx1, idx2, value1, value2, addtext)
- subroutine `ftnunit::ftnunit_write_html_failed_inbetween_real` (text, value, vmin, vmax)
- subroutine `ftnunit::ftnunit_write_html_failed_inbetween_double` (text, value, vmin, vmax)
- subroutine `ftnunit::ftnunit_write_html_failed_files` (text, string1, string2, string3)

## Variables

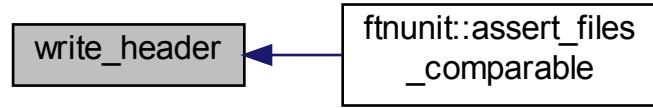
- procedure(proc), pointer, public `ftnunit::subptrtestprograminit` => null()
- procedure(proc), pointer, public `ftnunit::subptrtestinit` => null()
- integer, parameter `ftnunit::mode_all` = 0
- integer, parameter `ftnunit::mode_single` = 1
- integer, parameter `ftnunit::mode_list` = 2
- integer, parameter `ftnunit::dp` = kind(1.0d0)
- integer, save `ftnunit::run_test_level` = 99
- integer, save `ftnunit::single_test` = -1
- integer, save `ftnunit::test_mode` = mode\_all
- integer, save `ftnunit::last_test`
- integer, save `ftnunit::testno`
- integer, save `ftnunit::nofails`
- integer, save `ftnunit::nofails_prev`
- integer, save `ftnunit::notests_run`
- integer, save `ftnunit::notests_failed`
- integer, save `ftnunit::notests_ignored`
- integer, save `ftnunit::notests_work_in_progress`
- integer, save `ftnunit::noruns`
- logical, save `ftnunit::call_final` = .true.
- logical, save `ftnunit::previous` = .false.
- integer, save `ftnunit::failed_asserts` = 0
- logical, save `ftnunit::has_run` = .false.
- logical, save `ftnunit::ignore_test` = .false.
- logical, save `ftnunit::ignore_previous_test` = .false.
- character(len=20), save `ftnunit::html_file` = 'ftnunit.html'
- character(len=80), save `ftnunit::testname`
- character(len=80), save `ftnunit::testtitle` = 'Result of unit tests'
- character(len=32), save `ftnunit::ignore_reason`
- character(len=32), save `ftnunit::ignore_reason_previous_test` = ''
- character(len=\*), parameter `ftnunit::work_in_progress` = 'work-in-progress'

### 9.15.1 Function/Subroutine Documentation

#### 9.15.1.1 subroutine assert\_files\_comparable::write\_header( ) [private]

Definition at line 1407 of file ftnunit.f90.

Here is the caller graph for this function:



## 9.16 ftnunit\_hooks.f90 File Reference

### Modules

- module [ftnunit\\_hooks](#)

### Functions/Subroutines

- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_start](#) (text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_stop](#) (text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_assertion\\_failed](#) (text, assert\_text, failure\_text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_completed](#)

## 9.17 ftnunit\_hooks\_teamcity.f90 File Reference

### Modules

- module [ftnunit\\_hooks](#)

### Functions/Subroutines

- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_start](#) (text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_stop](#) (text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_assertion\\_failed](#) (text, assert\_text, failure\_text)
- subroutine [ftnunit\\_hooks::ftnunit\\_hook\\_test\\_completed](#)

## 9.18 general.f90 File Reference

Overall module for general library.

### Modules

- module [general](#)

### 9.18.1 Detailed Description

Overall module for general library.

## 9.19 geometryModuleRTOvertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping related to the geometry.

### Modules

- module `geometrymodulertoovertopping`

### Functions/Subroutines

- subroutine, public `geometrymodulertoovertopping::checkcrosssection` (`psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, succes, errorMessage`)
 

*checkCrossSection: check cross section*
- subroutine, public `geometrymodulertoovertopping::initializegeometry` (`psi, nCoordinates, xCoordinates, yCoordinates, roughnessFactors, geometry, succes, errorMessage`)
 

*initializeGeometry: initialize the geometry*
- subroutine, public `geometrymodulertoovertopping::allocatevectorsgeometry` (`nCoordinates, geometry`)
 

*allocateVectorsGeometry: allocate the geometry vectors*
- subroutine, public `geometrymodulertoovertopping::deallocategeometry` (`geometry`)
 

*deallocateGeometry: deallocate the geometry vectors*
- subroutine, public `geometrymodulertoovertopping::calculatesegmentsslopes` (`geometry, succes, errorMessage`)
 

*calculateSegmentSlopes: calculate the segment slopes*
- subroutine, public `geometrymodulertoovertopping::determinesegmenttypes` (`geometry`)
 

*determineSegmentTypes: determine the segment types*
- subroutine, public `geometrymodulertoovertopping::copygeometry` (`geometry, geometryCopy`)
 

*copyGeometry: copy a geometry structure*
- subroutine, public `geometrymodulertoovertopping::isequalgeometry` (`geometry1, geometry2, succes, errorMessage`)
 

*isEqualGeometry: are two geometries equal*
- subroutine, public `geometrymodulertoovertopping::mergesequentialberms` (`geometry, geometryMergedBerms, succes, errorMessage`)
 

*mergeSequentialBerms: merge sequential berms*
- subroutine, public `geometrymodulertoovertopping::adjustnonhorizontalberms` (`geometry, geometryFlatBerms, succes, errorMessage`)
 

*adjustNonHorizontalBerms: adjust non-horizontal berms*
- subroutine, public `geometrymodulertoovertopping::removeberms` (`geometry, geometryNoBerms, succes, errorMessage`)
 

*removeBerms: remove berms*
- subroutine, public `geometrymodulertoovertopping::removedikesegments` (`geometry, index, geometryAdjusted, succes, errorMessage`)
 

*removeDikeSegments: remove dike segments*
- subroutine, public `geometrymodulertoovertopping::splitcrosssection` (`geometry, L0, NwideBerms, geometrysectionB, geometrysectionF, succes, errorMessage`)
 

*splitCrossSection: split a cross section*
- subroutine, public `geometrymodulertoovertopping::calculatehorzlenghts` (`geometry, yLower, yUpper, horzLengths, succes, errorMessage`)

- subroutine, public `geometrymodulertoovertopping::calculatehorzlengths` (`geometry`, `yLower`, `yUpper`, `dx`, `succes`, `errorMessage`)
  - calculateHorzLengths: calculate horizontal lengths*
- subroutine, public `geometrymodulertoovertopping::calculatehorzdistance` (`geometry`, `yLower`, `yUpper`, `dx`, `succes`, `errorMessage`)
  - calculateHorzDistance: calculate horizontal distance*
- subroutine, public `geometrymodulertoovertopping::writecrosssection` (`geometry`, `geometryName`)
  - writeCrossSection: write a cross section*

### 9.19.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping related to the geometry.

## 9.20 interpolationTriangular.f90 File Reference

Module containing the function for triangular interpolation.

### Modules

- module `interpolationtriangular`
  - Module with function for triangular interpolation.*

### Functions/Subroutines

- real(kind=wp) function `interpolationtriangular::triangularinterpolation` (`x1`, `y1`, `h1`, `x2`, `y2`, `h2`, `x3`, `y3`, `h3`, `x`, `y`)
  - Function for triangular interpolation Using three result values of three different stations on the Dutch RD grid a plane is spaned. The result values are in general water levels. For a fourth point on the Dutch RD grid the result value is calculated lying on this plane. In general a water level is calculated.*
- real(kind=wp) function `interpolationtriangular::datainterpolation` (`var1`, `var2`, `weight`, `iscyclic`)

### 9.20.1 Detailed Description

Module containing the function for triangular interpolation.

## 9.21 mainModuleRTOvertopping.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

### Modules

- module `mainmodulertooverlapping`

### Functions/Subroutines

- subroutine, public `mainmodulertooverlapping::calculateovertopping` (`geometry`, `load`, `modelFactors`, `overtopping`, `succes`, `errorMessage`)
  - calculateOvertopping: calculate the overtopping*
- subroutine, public `mainmodulertooverlapping::calculateovertoppingsection` (`geometry`, `h`, `Hm0`, `Tm_10`, `L0`, `gammaBeta_z`, `gammaBeta_o`, `modelFactors`, `overtopping`, `succes`, `errorMessage`)
  - calculateOvertoppingSection: calculate the overtopping for a section*

- subroutine, public `mainmodulertoovertopping::calculatewaveovertopping` (geometry, h, Hm0, Tm\_10, z2, gammaBeta\_o, modelFactors, Qo, succes, errorMessage)
   
*calculateWaveOvertopping: calculate wave overtopping*
- subroutine `mainmodulertoovertopping::calculateovertoppingnegativefreeboard` (load, geometry, overtopping, succes, errorMessage)
   
*calculateOvertoppingNegativeFreeboard: calculate overtopping in case of negative freeboard*
- subroutine, public `mainmodulertoovertopping::interpolateresultssections` (geometry, L0, NwideBerms, overtoppingB, overtoppingF, overtopping, succes, errorMessage)
   
*interpolateResultsSections: interpolate results for split cross sections*
- subroutine, public `mainmodulertoovertopping::checkinputdata` (geometry, load, modelFactors, succes, errorMessage)
   
*checkInputdata: check the input data*
- subroutine, public `mainmodulertoovertopping::checkmodelfactors` (modelFactors, succes, errorMessage)
   
*checkModelFactors: check the input data*
- subroutine, public `mainmodulertoovertopping::convertovertoppinginput` (modelFactors, success, errorMessage)
   
*convertOvertoppingInput: convert the model factors from C-like to Fortran*

### 9.21.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

## 9.22 ModuleLogging.f90 File Reference

Module for steering the extra logging.

### Data Types

- type `modulelogging::tlogging`
  
*TLogging: structure for steering the logging.*

### Modules

- module `modulelogging`

### Variables

- integer, parameter `modulelogging::maxfilenameLength` = 256
   
*maximum length of filename*
- type(`tlogging`) `modulelogging::currentlogging`
  
*copy of argument logging*

### 9.22.1 Detailed Description

Module for steering the extra logging.

## 9.23 overtopplingInterface.f90 File Reference

This file contains the parameters and types (structs) as part of the interface to and from dllOvertopping.

## Data Types

- type `overtoppinginterface::tpprofilecoordinate`
- type `overtoppinginterface::overtoppinggeometrytype`
- type `overtoppinginterface::overtoppinggeometrytypeef`

## Modules

- module `overtoppinginterface`

## Variables

- integer, parameter, public `overtoppinginterface::varmodelfactorcriticalovertopping = 8`  
*Model factor critical overtopping.*

### 9.23.1 Detailed Description

This file contains the parameters and types (structs) as part of the interface to and from dllOvertopping.

## 9.24 physics.f90 File Reference

Overall module for physics library.

## Modules

- module `physics`

### 9.24.1 Detailed Description

Overall module for physics library.

## 9.25 precision.f90 File Reference

Define KINDs for controlling the precision of the REAL variables and define parameters for mathematical constants.

## Modules

- module `precision`  
*Module with parameters for KINDs and mathematical constants.*

## Variables

- integer, parameter `precision::sp = kind(1.0)`  
*Single precision.*
- integer, parameter `precision::dp = kind(1.0d0)`  
*Double precision.*
- integer, parameter `precision::wp = dp`

*Working precision.*

- real(wp), parameter `precision::almostzero` = 1.0d-30

*Definition of zero for the machine.*

- real(wp), parameter `precision::almostinf` = huge(1.0d0)

*Very large value as limit.*

- real(wp), parameter `precision::rholimit` = 0.99999d0

*Limit value for the correlation coefficient DO NOT CHANGE THIS VALUE WITHOUT SUFFICIENT RESEARCH.*

- real(wp), parameter `precision::betalimit` = 0.99999999d0

*Limit value for the beta ratio in combining DO NOT CHANGE ONLY USED FOR EXTRA WIND STOCAST.*

- real(wp), parameter `precision::pi` = 3.141592653589793238462643383279502884197\_wp

*Circle constant.*

- real(wp), parameter `precision::emconstant` = 0.57721566490153286060651209008240243104215933593992←\_wp

*Euler Mascheroni constant.*

- real(wp), parameter `precision::gravityconstant` = 9.80665\_wp

*Gravity constant for the Netherlands.*

- integer, parameter `precision::shortmax` = 32

- integer, parameter `precision::longmax` = 255

### 9.25.1 Detailed Description

Define KINDs for controlling the precision of the REAL variables and define parameters for mathematical constants.

**Note:** Use the parameter "wp" to set the working precision for using routines.

## 9.26 typeDefinitionsRTOvertopping.f90 File Reference

This file contains a module with the type definitions for Dikes Overtopping.

### Data Types

- type `typedefinitionsrovertopping::tpgeometry`

*tpGeometry: structure with geometry data*

- type `typedefinitionsrovertopping::tpload`

*tpLoad: structure with load parameters*

- type `typedefinitionsrovertopping::tpovertoppinginput`

*OvertoppingModelFactors: C-structure with model factors.*

- type `typedefinitionsrovertopping::tpovertopping`

*tpOvertopping: structure with overtopping results*

### Modules

- module `typedefinitionsrovertopping`

## Variables

- real(wp), parameter `typedefinitionsrovertopping::xdiff_min` = 2.0d-2  
*minimal value distance between x-coordinates (m)*
- real(wp), parameter `typedefinitionsrovertopping::margindiff` = 1.0d-14  
*margin for minimal distance (m)*
- real(wp), parameter `typedefinitionsrovertopping::berm_min` = 0.0d0  
*minimal value gradient berm segment*
- real(wp), parameter `typedefinitionsrovertopping::berm_max` = 1.0d0/15  
*maximal value gradient berm segment*
- real(wp), parameter `typedefinitionsrovertopping::slope_min` = 1.0d0/8  
*minimal value gradient slope segment*
- real(wp), parameter `typedefinitionsrovertopping::slope_max` = 1.0d0  
*maximal value gradient slope segment*
- real(wp), parameter `typedefinitionsrovertopping::margingrad` = 0.0025d0  
*margin for minimal and maximal gradients*
- real(wp), parameter `typedefinitionsrovertopping::rfactor_min` = 0.5d0  
*minimal value roughness factor dike segments*
- real(wp), parameter `typedefinitionsrovertopping::rfactor_max` = 1.0d0  
*maximal value roughness factor dike segments*
- real(wp), parameter `typedefinitionsrovertopping::mz2_min` = 0.0d0  
*minimal value model factor of 2% runup height*
- real(wp), parameter `typedefinitionsrovertopping::mz2_max` = huge(mz2\_max)  
*maximal value model factor of 2% runup height*
- real(wp), parameter `typedefinitionsrovertopping::frunup1_min` = 0.0d0  
*minimal value model factor 1 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::frunup1_max` = huge(fRunup1\_max)  
*maximal value model factor 1 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::frunup2_min` = 0.0d0  
*minimal value model factor 2 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::frunup2_max` = huge(fRunup2\_max)  
*maximal value model factor 2 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::frunup3_min` = 0.0d0  
*minimal value model factor 3 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::frunup3_max` = huge(fRunup3\_max)  
*maximal value model factor 3 for wave run-up*
- real(wp), parameter `typedefinitionsrovertopping::fb_min` = 0.0d0  
*minimal value model factor for breaking waves*
- real(wp), parameter `typedefinitionsrovertopping::fb_max` = huge(fB\_max)  
*maximal value model factor for breaking waves*
- real(wp), parameter `typedefinitionsrovertopping::fn_min` = 0.0d0  
*minimal value model factor for non-breaking waves*
- real(wp), parameter `typedefinitionsrovertopping::fn_max` = huge(fN\_max)  
*maximal value model factor for non-breaking waves*
- real(wp), parameter `typedefinitionsrovertopping::fs_min` = 0.0d0  
*minimal value model factor for shallow waves*
- real(wp), parameter `typedefinitionsrovertopping::fs_max` = huge(fS\_max)  
*maximal value model factor for shallow waves*
- real(wp), parameter `typedefinitionsrovertopping::foreshore_min` = 0.3d0  
*minimal value reduction factor foreshore*
- real(wp), parameter `typedefinitionsrovertopping::foreshore_max` = 1.0d0

- maximal value reduction factor foreshore
- integer, parameter `typedefinitionsrovertopping::z2_iter_max` = 100  
*maximal number of iterations for calculation z2*
- real(wp), parameter `typedefinitionsrovertopping::z2_margin` = 0.001d0  
*margin for convergence criterium calculation z2*

### 9.26.1 Detailed Description

This file contains a module with the type definitions for Dikes Overtopping.

## 9.27 utilities.f90 File Reference

General utility routines.

### Modules

- module `utilities`

*Module with general utility routines.*

### Functions/Subroutines

- subroutine `utilities::getfreelunumber` (lun)  
*getFreeLuNumber Routine to find a free LU-number*
- pure character(len(str)) function `utilities::to_lower` (str)  
*to\_lower Changes a string to lower case*
- pure character(len(str)) function `utilities::to_upper` (str)  
*to\_upper Changes a string to upper case*

### Variables

- character(26), parameter, private `utilities::cap` = 'ABCDEFGHIJKLMNPQRSTUVWXYZ'
- character(26), parameter, private `utilities::low` = 'abcdefghijklmnopqrstuvwxyz'

### 9.27.1 Detailed Description

General utility routines.

## 9.28 vectorUtilities.f90 File Reference

Implement various utility functions for vectors.

### Modules

- module `vectorutilities`

*Module with vector utility functions.*

## Functions/Subroutines

- real(kind=wp) function dimension(size(x)) [vectorutilities::normalize](#) (x)
 

*Function for normalizing a vector If the vector is the null vector, a unit vector with uniform components is returned instead.*
- real(kind=wp) function [vectorutilities::interpolateline](#) (x1, x2, f1, f2, x)
 

*Function to interpolate and extrapolate linear on a line.*
- real(kind=wp) function [vectorutilities::linearinterpolate](#) (X, Y, xx)
 

*Function for linearly interpolating and extrapolating.*
- real(kind=wp) function [vectorutilities::loglinearinterpolate](#) (X, Y, xx)
 

*Function for log-linearly interpolating and extrapolating.*
- real(kind=wp) function [vectorutilities::linearinterpolatecyclic](#) (X, Y, xx)
 

*Function for linearly interpolating cyclic data.*
- real(kind=wp) function [vectorutilities::loglinearinterpolatecyclic](#) (X, Y, xx)
 

*Function for log-linearly interpolating and extrapolating cyclic data.*
- real(kind=wp) function [vectorutilities::linearinterpolateangles](#) (X, Y, xx)
 

*Function for linearly interpolating and extrapolating angles.*
- subroutine [vectorutilities::sortandlocate](#) (X, IDXsorted, xx, locID, locIDm1, Cyclic)
 

*Subroutine to bracket a given value xx within two elements of an array after sort in descending order.*
- real(kind=wp) function [vectorutilities::stepsize](#) (i, x)
 

*Function for the Trapezium rule.*
- real(kind=wp) function [vectorutilities::trapeziumrule](#) (x, f)
 

*Function for the Trapezium rule.*
- real(kind=wp) function [vectorutilities::intgratewithexponentcomponent](#) (x, f)
 

*Function for the Trapezium rule.*
- subroutine [vectorutilities::ssort](#) (x, iy, n, kflag)
 

*Subroutine to sort a vector in either ascending or descending order.*

### 9.28.1 Detailed Description

Implement various utility functions for vectors.

## 9.29 waveParametersUtilities.f90 File Reference

Module waveParametersUtilities for wave parameter computations.

### Modules

- module [waveparametersutilities](#)

*Module to calculate the wave steepness or the wave period.*

## Functions/Subroutines

- real(wp) function [waveparametersutilities::computewavestEEPNESS](#) (waveHeight, wavePeriod)
 

*Module to calculate the the wave steepness from the wave height and the wave period.*
- real(wp) function [waveparametersutilities::computewavePERIOD](#) (waveHeight, waveSteepness)
 

*Module to calculate the the wave period from the wave height and the wave steepness.*

### 9.29.1 Detailed Description

Module waveParametersUtilities for wave parameter computations.

## 9.30 waveReduction.f90 File Reference

Wave reduction routines.

### Data Types

- type [physics\\_wavereduction::tpwavereduction](#)

### Modules

- module [physics\\_wavereduction](#)

*Module with routines for wave reduction.*

### Functions/Subroutines

- subroutine, public [physics\\_wavereduction::calculatewavereduction](#) (wavereduction, errorCode, Hs)  
*Computation of wave reduction due to shallow foreland (simple model) Wave reduction function to compute the wave reduction (simple model)*

### 9.30.1 Detailed Description

Wave reduction routines.

## 9.31 waveRunup.f90 File Reference

This file contains a module with the core computations for Dikes Overtopping.

### Modules

- module [waverunup](#)

### Functions/Subroutines

- subroutine, public [waverunup::iterationwaverunup](#) (geometry, h, Hm0, Tm\_10, gammaBeta\_z, modelFactors, z2, succes, errorMessage)  
*iterationWaveRunup: iteration for the wave runup*
- real(kind=wp) function [waverunup::determinestartingvalue](#) (i, relaxationFactor, z2\_start, z2\_end, Hm0)
- integer function [waverunup::findsmallestresidu](#) (z2\_start, z2\_end)
- logical function [waverunup::convergedwithresidu](#) (i, z2\_start, z2\_end)

### 9.31.1 Detailed Description

This file contains a module with the core computations for Dikes Overtopping.

## 9.32 writeUtilities.f90 File Reference

General write utilities.

### Data Types

- interface `writeutilities::writetodevice`

### Modules

- module `writeutilities`

### Functions/Subroutines

- subroutine `writeutilities::writetodevice_line` (fileId, line)
 

*Write to device.*
- subroutine `writeutilities::writetodevice_text_real` (fileId, formatText, textView, realValue)
 

*Write to device.*
- subroutine, public `writeutilities::writetodevicechars` (fileId, char, count)
 

*Write count times the same character to output device.*
- character(len=5) function, public `writeutilities::inttostr` (intValue)
 

*Convert integer to string.*

#### 9.32.1 Detailed Description

General write utilities.

## 9.33 zFunctionsWTIOvertopping.f90 File Reference

This file contains the limit state functions for wave overtopping within WTI.

### Modules

- module `zfunctionswtiovertopping`

*Module for the Limit State Functions (Z-functions) for wave overtopping.*

### Functions/Subroutines

- subroutine, public `zfunctionswtiovertopping::calculateqorto` (dikeHeight, modelFactors, overtopping, load, geometry, succes, errorMessage)
 

*Subroutine to calculate the overtopping discharge with the RTO-overtopping dll.*
- subroutine, public `zfunctionswtiovertopping::profileinstructure` (nrCoordinates, xcoordinates, ycoordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)
 

*Subroutine to fill the profile in a structure and call the adjustment function of the profile due to a desired dike height.*
- subroutine `zfunctionswtiovertopping::adjustprofile` (nrCoordinates, coordinates, dikeHeight, nrCoordsAdjusted, xCoordsAdjusted, zCoordsAdjusted, succes, errorMessage)
 

*Subroutine adjust the profile due to a desired dike height.*
- real(kind=wp) function, public `zfunctionswtiovertopping::zfunclogratios` (qo, qc, mqo, mqc, success, errorMessage)
 

*Routine to compute the limit state value by using the logs of the overtopping discharges (computed and desired)*

### 9.33.1 Detailed Description

This file contains the limit state functions for wave overtopping within WTI.

# Index

adjustinfluencefactors  
    formulamodulertoovertopping, 67

adjustnonhorizontalberms  
    geometrymodulertoovertopping, 109

adjustprofile  
    zfunctionsztiovertopping, 154

allocatevectorgsgeometry  
    geometrymodulertoovertopping, 109

allow\_log  
    Feedback library, 14

allow\_logging  
    feedbackdll, 64

almostinf  
    precision, 132

almostzero  
    precision, 132

angleUtilities.f90, 193

anglebetween2directions  
    angleutilities, 35

angleutilities, 35  
    anglebetween2directions, 35

assert\_comparable\_double  
    ftnunit, 77  
        ftnunit::assert\_comparable, 160

assert\_comparable\_double1d  
    ftnunit, 77  
        ftnunit::assert\_comparable, 160

assert\_comparable\_double2d  
    ftnunit, 77  
        ftnunit::assert\_comparable, 160

assert\_comparable\_real  
    ftnunit, 78  
        ftnunit::assert\_comparable, 160

assert\_comparable\_real1d  
    ftnunit, 78  
        ftnunit::assert\_comparable, 160

assert\_comparable\_real2d  
    ftnunit, 78  
        ftnunit::assert\_comparable, 160

assert\_equal\_int  
    ftnunit, 79  
        ftnunit::assert\_equal, 161

assert\_equal\_int1d  
    ftnunit, 79  
        ftnunit::assert\_equal, 161

assert\_equal\_logical  
    ftnunit, 79  
        ftnunit::assert\_equal, 161

assert\_equal\_logical1d

    ftnunit, 80  
        ftnunit::assert\_equal, 161

assert\_equal\_string  
    ftnunit, 80  
        ftnunit::assert\_equal, 162

assert\_equal\_string1d  
    ftnunit, 80  
        ftnunit::assert\_equal, 162

assert\_false  
    ftnunit, 81

assert\_files\_comparable  
    ftnunit, 81

assert\_inbetween\_double  
    ftnunit, 81  
        ftnunit::assert\_inbetween, 162

assert\_inbetween\_real  
    ftnunit, 82  
        ftnunit::assert\_inbetween, 162

assert\_true  
    ftnunit, 82

berm\_max  
    typedefinitionsrtodovertopping, 135

berm\_min  
    typedefinitionsrtodovertopping, 135

betafromq  
    conversionfunctions, 36

betalimit  
    precision, 132

breakwater.f90, 193

breakwaterEnumerations.f90, 194

breakwatercaisson  
    physics\_breakwatertypesenumerations, 128

breakwaternotpresent  
    physics\_breakwatertypesenumerations, 128

breakwaterrubblemound  
    physics\_breakwatertypesenumerations, 128

breakwaterverticalwall  
    physics\_breakwatertypesenumerations, 129

bretschneider.f90, 194

calculateanglewaveattack  
    formulamodulertoovertopping, 67

calculatebreakerlimit  
    formulamodulertoovertopping, 67

calculatebreakerparameter  
    formulamodulertoovertopping, 68

calculatebreakwater  
    physics\_breakwater, 127

calculategammab

factormodulertoovertopping, 50  
 calculategammabeta  
     factormodulertoovertopping, 51  
 calculategammaf  
     factormodulertoovertopping, 51  
 calculatehorzdistance  
     geometrymodulertoovertopping, 111  
 calculatehorzlengths  
     geometrymodulertoovertopping, 111  
 calculateovertopping  
     mainmodulertoovertopping, 121  
 calculateovertoppingnegativefreeboard  
     mainmodulertoovertopping, 122  
 calculateovertoppingsection  
     mainmodulertoovertopping, 122  
 calculateqo  
     dllovertopping, 42  
 calculateqof  
     dllovertopping, 43  
 calculateqorto  
     zfunctionswtiovertopping, 154  
 calculatesegmentsslopes  
     geometrymodulertoovertopping, 112  
 calculatetanalpha  
     factormodulertoovertopping, 52  
 calculatewavebretschneider  
     physics\_bretschneider, 129  
 calculatewavelength  
     formulamodulertoovertopping, 68  
 calculatewaveovertopping  
     mainmodulertoovertopping, 123  
 calculatewaveovertoppingdischarge  
     formulamodulertoovertopping, 68  
 calculatewavereduction  
     physics\_wavereduction, 131  
 calculatewaverunup  
     formulamodulertoovertopping, 70  
 calculatewavesteeplness  
     formulamodulertoovertopping, 71  
 calczvalue  
     dllovertopping, 44  
 call\_final  
     ftnunit, 104  
 cap  
     utilities, 140  
 checkcrosssection  
     geometrymodulertoovertopping, 112  
 checkinputdata  
     mainmodulertoovertopping, 124  
 checkmodelfactors  
     mainmodulertoovertopping, 125  
 closefile  
     feedbackdll, 58  
 colour\_number  
     ftnunit, 82  
 computedovertopping  
     typedefinitionsrtovertopping::tpovertoppinginput, 185  
                 computewaveperiod  
                     waveparametersutilities, 149  
                 computewavesteeplness  
                     waveparametersutilities, 149  
 context  
     feedbackdll::feedbackdata, 169  
 contextnumber  
     feedbackdll::feedbackdata, 169  
 contexttxt  
     feedbackdll, 58  
 convergedwithresidu  
     waverunup, 150  
 conversionFunctions.f90, 195  
 conversionfunctions, 36  
     betafromq, 36  
     freqfrombeta, 37  
     logqfrombeta, 37  
     pfrombeta, 39  
     pqfrombeta, 39  
     qfrombeta, 40  
     qmin, 41  
     returntimefrombeta, 41  
     ulimit, 41  
     upperlog, 41  
 convertovertoppinginput  
     mainmodulertoovertopping, 125  
 copygeometry  
     geometrymodulertoovertopping, 113  
 criticalovertopping  
     typedefinitionsrtovertopping::tpovertoppinginput, 185  
 cubicroots  
     formulamodulertoovertopping, 71  
 currentlogging  
     modulelogging, 126  
 d  
     physics\_bretschneider::tpbretschneider, 178  
 datainterpolation  
     interpolationtriangular, 119  
 deallocategeometry  
     geometrymodulertoovertopping, 114  
 determinesegmenttypes  
     geometrymodulertoovertopping, 114  
 determinestartingvalue  
     waverunup, 150  
 dll  
     Feedback library, 32  
 dllOvertopping.f90, 195  
 dllovertopping, 42  
     calculateqo, 42  
     calculateqof, 43  
     calczvalue, 44  
     geometry\_c\_f, 44  
     validateinputc, 45  
     validateinputf, 45  
     versionnumber, 46  
 dp  
     feedback, 55

ftnunit, 104  
precision, 133  
dropcontext  
  feedbackdll, 58

emconstant  
  precision, 133  
equalReals.f90, 196  
equalreals, 46  
  equalrealsabsolute, 47  
  equalrealsinvector, 47  
  equalrealsrelative, 48  
  equaltableswithreals, 48  
  equalvectorswithintegers, 49  
  equalvectorswithreals, 49  
  equalvectorswithrealsweightedmargin, 49

equalrealsabsolute  
  equalreals, 47

equalrealsinvector  
  equalreals, 47

equalrealsrelative  
  equalreals, 48

equaltableswithreals  
  equalreals, 48

equalvectorswithintegers  
  equalreals, 49

equalvectorswithreals  
  equalreals, 49

equalvectorswithrealsweightedmargin  
  equalreals, 49

error  
  feedbackdll::feedbackdata, 169

errormessagedouble  
  Feedback library, 15  
  feedback::errormessage, 163

errormessageint  
  Feedback library, 15  
  feedback::errormessage, 164

errormessagereal  
  Feedback library, 15  
  feedback::errormessage, 164

errormessageetxt  
  Feedback library, 15  
  feedback::errormessage, 164

expect\_program\_stop  
  ftnunit, 83

f  
  physics\_bretschneider::tpbretschneider, 178

factorModuleRTOvertopping.f90, 197  
factordeterminationq\_b\_f\_b  
  typedefinitionsrovertopping::tpovertoppinginput,  
    185

factordeterminationq\_b\_f\_n  
  typedefinitionsrovertopping::tpovertoppinginput,  
    185

factormodulertovertopping, 50  
  calculategammab, 50  
  calculategammabeta, 51

  calculategammaf, 51  
  calculatetanalpha, 52

failed\_asserts  
  ftnunit, 104

fatalerroraction  
  feedbackdll, 64

fatalerrordouble  
  Feedback library, 16  
  feedback::fatalerror, 165

fatalerrorint  
  Feedback library, 16  
  feedback::fatalerror, 165

fatalerrormessage  
  feedbackdll, 64

fatalerroccured  
  Feedback library, 16

fatalerrorreal  
  Feedback library, 16  
  feedback::fatalerror, 166

fatalerrortxt  
  Feedback library, 17  
  feedback::fatalerror, 166

fb\_max  
  typedefinitionsrovertopping, 135

fb\_min  
  typedefinitionsrovertopping, 135

fbdata  
  feedbackdll, 64

feedback, 53  
  dp, 55  
  id, 55  
  sp, 55  
  wp, 55

Feedback library, 11  
  allow\_log, 14  
  dll, 32

  errormessagedouble, 15

  errormessageint, 15

  errormessagereal, 15

  errormessageetxt, 15

  fatalerrordouble, 16

  fatalerrorint, 16

  fatalerroroccured, 16

  fatalerrorreal, 16

  fatalerrortxt, 17

  feedbackclose, 17

  feedbackcontextdouble, 18

  feedbackcontextint, 18

  feedbackcontextreal, 18

  feedbackcontexttxt, 18

  feedbackdropcontext, 19

  feedbackenter, 19

  feedbackerror, 19

  feedbackinitialise, 20

  feedbackleave, 20

  feedbackprintcontext, 21

  feedbackprintstack, 21

  getfatalerrorexpected, 22

getfatalerrormessage, 22  
 getthreadid, 24  
 onlyfile, 32  
 onlyscreen, 32  
 p, 24  
 progressreportarray, 25  
 progressreportdouble, 26  
 progressreportint, 26  
 progressreportreal, 26  
 progressreporttxt, 26  
 progressreportvalueandarray, 27  
 resetfatalerroroccurred, 27  
 screenandfile, 32  
 setallowlogging, 28  
 setfatalerroraction, 28  
 setfatalerrorexpected, 28  
 setfatalerroroccurred, 29  
 throw, 29  
 todll, 30  
 verbosebasic, 32  
 verbosedebugging, 32  
 verbosedetailed, 32  
 verbosenone, 33  
 warningmessagedouble, 31  
 warningmessageint, 31  
 warningmessagereal, 31  
 warningmessagetxt, 31  
 feedback.f90, 198  
 feedback::errormessage, 163  
     errormessagedouble, 163  
     errormessageint, 164  
     errormessagereal, 164  
     errormessagetxt, 164  
 feedback::fatalerror, 165  
     fatalerrordouble, 165  
     fatalerrorint, 165  
     fatalerrorreal, 166  
     fatalerrortxt, 166  
 feedback::feedbackcontext, 166  
     feedbackcontextdouble, 167  
     feedbackcontextint, 167  
     feedbackcontextreal, 167  
     feedbackcontexttxt, 167  
 feedback::progressreport, 173  
     progressreportarray, 174  
     progressreportdouble, 175  
     progressreportint, 175  
     progressreportreal, 175  
     progressreporttxt, 175  
     progressreportvalueandarray, 176  
 feedback::warningmessage, 189  
     warningmessagedouble, 190  
     warningmessageint, 190  
     warningmessagereal, 190  
     warningmessagetxt, 190  
 feedback\_parameters, 56  
     onfatalerrorstop, 56  
     onfatalerrorthrowexception, 56  
     onfatalerrorunittest, 56  
 feedback\_parameters.f90, 200  
 feedbackDLL.f90, 201  
 feedbackclose  
     Feedback library, 17  
 feedbackcontextdouble  
     Feedback library, 18  
     feedback::feedbackcontext, 167  
 feedbackcontextint  
     Feedback library, 18  
     feedback::feedbackcontext, 167  
 feedbackcontextreal  
     Feedback library, 18  
     feedback::feedbackcontext, 167  
 feedbackcontexttxt  
     Feedback library, 18  
     feedback::feedbackcontext, 167  
 feedbackdll, 56  
     allow\_logging, 64  
     closefile, 58  
     contexttxt, 58  
     dropcontext, 58  
     fatalerroraction, 64  
     fatalerrormessage, 64  
     fbdata, 64  
     getallowlogging, 58  
     getfatalerrormsg, 58  
     getfeedbackerror, 59  
     getonfatalerroraction, 59  
     getsetfatalerrorexpected, 59  
     getsetfatalerroroccurred, 59  
     getstacklevel, 59  
     getunitnumber, 59  
     getverbosity, 61  
     getverbositylevel, 61  
     id, 64  
     initialise, 61  
     isfatalerrorexpected, 64  
     isfatalerroroccurred, 64  
     outputtodll, 61  
     poproutine, 61  
     printcontext, 61  
     printmessage, 62  
     printstack, 62  
     pushroutine, 63  
     setallowlogging, 63  
     setfatalerrormessage, 63  
     setonfatalerroraction, 63  
     writetodll, 63  
 feedbackdll::feedbackdata, 168  
     context, 169  
     contextnumber, 169  
     error, 169  
     lun, 169  
     output, 169  
     routine, 169  
     stacklevel, 169  
     verbositylevel, 169

feedbackdropcontext  
    Feedback library, 19

feedbackenter  
    Feedback library, 19

feedbackerror  
    Feedback library, 19

feedbackinitialise  
    Feedback library, 20

feedbackleave  
    Feedback library, 20

feedbackprintcontext  
    Feedback library, 21

feedbackprintstack  
    Feedback library, 21

fileUtilities.f90, 202

filename  
    modulelogging::tlogging, 177

fileutilities, 65

- readable, 65
- removefile, 65
- writetablevalues, 65

findsmallestresidu  
    waverunup, 150

fn\_max  
    typedefinitionsrovertopping, 135

fn\_min  
    typedefinitionsrovertopping, 135

foreland.f90, 203

foreshore\_max  
    typedefinitionsrovertopping, 136

foreshore\_min  
    typedefinitionsrovertopping, 136

formulaModuleRTOvertopping.f90, 203

formulamodulertovertopping, 66

- adjustinfluencefactors, 67
- calculateanglewaveattack, 67
- calculatebreakerlimit, 67
- calculatebreakerparameter, 68
- calculatewavelength, 68
- calculatewaveovertoppingdischarge, 68
- calculatewaverunup, 70
- calculatwavesteeppness, 71
- cubicroots, 71
- isequalreal, 72
- isequalzero, 72
- realrootscubicfunction, 72
- rootsdepressedcubic, 74
- rootsgeneralcubic, 74

freqfrombeta  
    conversionfunctions, 37

frunup1  
    typedefinitionsrovertopping::tpovertoppinginput, 185

frunup1\_max  
    typedefinitionsrovertopping, 136

frunup1\_min  
    typedefinitionsrovertopping, 136

frunup2  
    typedefinitionsrovertopping::tpovertoppinginput, 185

frunup2\_max  
    typedefinitionsrovertopping, 136

frunup2\_min  
    typedefinitionsrovertopping, 136

frunup3  
    typedefinitionsrovertopping::tpovertoppinginput, 186

frunup3\_max  
    typedefinitionsrovertopping, 136

frunup3\_min  
    typedefinitionsrovertopping, 136

fs\_max  
    typedefinitionsrovertopping, 136

fs\_min  
    typedefinitionsrovertopping, 137

fshallow  
    typedefinitionsrovertopping::tpovertoppinginput, 186

ftnunit, 75

- assert\_comparable\_double, 77
- assert\_comparable\_double1d, 77
- assert\_comparable\_double2d, 77
- assert\_comparable\_real, 78
- assert\_comparable\_real1d, 78
- assert\_comparable\_real2d, 78
- assert\_equal\_int, 79
- assert\_equal\_int1d, 79
- assert\_equal\_logical, 79
- assert\_equal\_logical1d, 80
- assert\_equal\_string, 80
- assert\_equal\_string1d, 80
- assert\_false, 81
- assert\_files\_comparable, 81
- assert\_inbetween\_double, 81
- assert\_inbetween\_real, 82
- assert\_true, 82
- call\_final, 104
- colour\_number, 82
- dp, 104
- expect\_program\_stop, 83
- failed\_asserts, 104
- ftnunit\_file\_exists, 83
- ftnunit\_make\_empty\_file, 83
- ftnunit\_remove\_file, 84
- ftnunit\_write\_html\_close\_row, 84
- ftnunit\_write\_html\_cpu, 85
- ftnunit\_write\_html\_failed\_double, 86
- ftnunit\_write\_html\_failed\_double1d, 87
- ftnunit\_write\_html\_failed\_double2d, 87
- ftnunit\_write\_html\_failed\_equivalent, 88
- ftnunit\_write\_html\_failed\_equivalent1d, 88
- ftnunit\_write\_html\_failed\_files, 89
- ftnunit\_write\_html\_failed\_inbetween\_double, 89
- ftnunit\_write\_html\_failed\_inbetween\_real, 90
- ftnunit\_write\_html\_failed\_int, 91
- ftnunit\_write\_html\_failed\_int1d, 91

ftnunit\_write\_html\_failed\_logic, 92  
 ftnunit\_write\_html\_failed\_real, 93  
 ftnunit\_write\_html\_failed\_real1d, 93  
 ftnunit\_write\_html\_failed\_real2d, 94  
 ftnunit\_write\_html\_failed\_string, 94  
 ftnunit\_write\_html\_failed\_string1d, 95  
 ftnunit\_write\_html\_footer, 95  
 ftnunit\_write\_html\_header, 96  
 ftnunit\_write\_html\_ignored, 96  
 ftnunit\_write\_html\_previous\_failed, 97  
 ftnunit\_write\_html\_previous\_stopped, 98  
 ftnunit\_write\_html\_test\_begin, 98  
 has\_run, 104  
 html\_file, 104  
 ignore\_previous\_test, 104  
 ignore\_reason, 104  
 ignore\_reason\_previous\_test, 104  
 ignore\_test, 104  
 issilent, 99  
 last\_test, 104  
 mode\_all, 104  
 mode\_list, 104  
 mode\_single, 104  
 nofails, 105  
 nofails\_prev, 105  
 noruns, 105  
 notests\_failed, 105  
 notests\_ignored, 105  
 notests\_run, 105  
 notests\_work\_in\_progress, 105  
 previous, 105  
 run\_test\_level, 105  
 runtests, 99  
 runtests\_final, 100  
 runtests\_init, 100  
 runtests\_init\_priv, 101  
 setruntestlevel, 102  
 settesttitle, 102  
 setworkingdir, 102  
 single\_test, 105  
 subptrtestinit, 105  
 subptrtestprograminit, 105  
 test, 102  
 test\_mode, 106  
 testname, 106  
 testno, 106  
 testtitle, 106  
 testwithlevel, 103  
 work\_in\_progress, 106  
 ftnunit.f90, 204  
 write\_header, 206  
 ftnunit::assert\_comparable, 159  
 assert\_comparable\_double, 160  
 assert\_comparable\_double1d, 160  
 assert\_comparable\_double2d, 160  
 assert\_comparable\_real, 160  
 assert\_comparable\_real1d, 160  
 assert\_comparable\_real2d, 160  
 ftnunit::assert\_equal, 161  
 assert\_equal\_int, 161  
 assert\_equal\_int1d, 161  
 assert\_equal\_logical, 161  
 assert\_equal\_logical1d, 161  
 assert\_equal\_string, 162  
 assert\_equal\_string1d, 162  
 ftnunit::assert\_inbetween, 162  
 assert\_inbetween\_double, 162  
 assert\_inbetween\_real, 162  
 ftnunit::proc, 172  
 proc, 173  
 ftnunit\_file\_exists  
 ftnunit, 83  
 ftnunit\_hook\_test\_assertion\_failed  
 ftnunit\_hooks, 106  
 ftnunit\_hook\_test\_completed  
 ftnunit\_hooks, 107  
 ftnunit\_hook\_test\_start  
 ftnunit\_hooks, 107  
 ftnunit\_hook\_test\_stop  
 ftnunit\_hooks, 108  
 ftnunit\_hooks, 106  
 ftnunit\_hook\_test\_assertion\_failed, 106  
 ftnunit\_hook\_test\_completed, 107  
 ftnunit\_hook\_test\_start, 107  
 ftnunit\_hook\_test\_stop, 108  
 ftnunit\_hooks.f90, 207  
 ftnunit\_hooks\_teamcity.f90, 207  
 ftnunit\_make\_empty\_file  
 ftnunit, 83  
 ftnunit\_remove\_file  
 ftnunit, 84  
 ftnunit\_write\_html\_close\_row  
 ftnunit, 84  
 ftnunit\_write\_html\_cpu  
 ftnunit, 85  
 ftnunit\_write\_html\_failed\_double  
 ftnunit, 86  
 ftnunit\_write\_html\_failed\_double1d  
 ftnunit, 87  
 ftnunit\_write\_html\_failed\_double2d  
 ftnunit, 87  
 ftnunit\_write\_html\_failed\_equivalent  
 ftnunit, 88  
 ftnunit\_write\_html\_failed\_equivalent1d  
 ftnunit, 88  
 ftnunit\_write\_html\_failed\_files  
 ftnunit, 89  
 ftnunit\_write\_html\_failed\_inbetween\_double  
 ftnunit, 89  
 ftnunit\_write\_html\_failed\_inbetween\_real  
 ftnunit, 90  
 ftnunit\_write\_html\_failed\_int  
 ftnunit, 91  
 ftnunit\_write\_html\_failed\_int1d  
 ftnunit, 91  
 ftnunit\_write\_html\_failed\_logic

ftnunit, 92  
ftnunit\_write\_html\_failed\_real  
    ftnunit, 93  
ftnunit\_write\_html\_failed\_real1d  
    ftnunit, 93  
ftnunit\_write\_html\_failed\_real2d  
    ftnunit, 94  
ftnunit\_write\_html\_failed\_string  
    ftnunit, 94  
ftnunit\_write\_html\_failed\_string1d  
    ftnunit, 95  
ftnunit\_write\_html\_footer  
    ftnunit, 95  
ftnunit\_write\_html\_header  
    ftnunit, 96  
ftnunit\_write\_html\_ignored  
    ftnunit, 96  
ftnunit\_write\_html\_previous\_failed  
    ftnunit, 97  
ftnunit\_write\_html\_previous\_stopped  
    ftnunit, 98  
ftnunit\_write\_html\_test\_begin  
    ftnunit, 98

general, 108  
general.f90, 207  
geometry\_c\_f  
    dlovertopping, 44  
geometryModuleRTOvertopping.f90, 208  
geometrymodulertooverlapping, 108  
    adjustnonhorizontalberms, 109  
    allocatevectorsgeometry, 109  
    calculatehorzdistance, 111  
    calculatehorzlenghts, 111  
    calculatesegmentsslopes, 112  
    checkcrosssection, 112  
    copygeometry, 113  
    deallocategeometry, 114  
    determinesegmenttypes, 114  
    initializegeometry, 114  
    isequalgeometry, 115  
    mergesequentialberms, 116  
    removeberms, 116  
    removedikesegments, 117  
    splitcrosssection, 118  
    writecrosssection, 119

getallowlogging  
    feedbackdll, 58  
getfatalerrorexpected  
    Feedback library, 22  
getfatalerrormessage  
    Feedback library, 22  
getfatalerrormsg  
    feedbackdll, 58  
getfeedbackerror  
    feedbackdll, 59  
getfreelunumber  
    utilities, 138  
getonfatalerroraction

    feedbackdll, 59  
getsetfatalerrorexpected  
    feedbackdll, 59  
getsetfatalerroroccurred  
    feedbackdll, 59  
getstacklevel  
    feedbackdll, 59  
getthreadid  
    Feedback library, 24  
getunitnumber  
    feedbackdll, 59  
getverbosity  
    feedbackdll, 61  
getverbosylevel  
    feedbackdll, 61  
gravityconstant  
    precision, 133

h  
    physics\_wavereduction::tpwavereduction, 189  
    typedefinitionsrovertopping::tpload, 182

has\_run  
    ftnunit, 104

hm0  
    typedefinitionsrovertopping::tpload, 182

hs  
    physics\_wavereduction::tpwavereduction, 189

ht  
    physics\_wavereduction::tpwavereduction, 189

html\_file  
    ftnunit, 104

id  
    feedback, 55  
    feedbackdll, 64

ignore\_previous\_test  
    ftnunit, 104

ignore\_reason  
    ftnunit, 104

ignore\_reason\_previous\_test  
    ftnunit, 104

ignore\_test  
    ftnunit, 104

initialise  
    feedbackdll, 61

initializegeometry  
    geometrymodulertooverlapping, 114

interpolateline  
    vectorutilities, 141

interpolateresultssections  
    mainmodulertooverlapping, 126

interpolationTriangular.f90, 209  
interpolationtriangular, 119  
    datainterpolation, 119  
    triangularinterpolation, 120

intgratewithexponentcomponent  
    vectorutilities, 141

inttostr  
    writeutilities, 152

isequalgeometry  
     geometrymodulertoovertopping, 115  
 isequalreal  
     formulamodulertoovertopping, 72  
 isequalzero  
     formulamodulertoovertopping, 72  
 isfatalerrorexpected  
     feedbackdll, 64  
 isfatalerroroccurred  
     feedbackdll, 64  
 issilent  
     ftnunit, 99  
 iterationwaverunup  
     waverunup, 151  
  
 last\_test  
     ftnunit, 104  
 linearinterpolate  
     vectorutilities, 141  
 linearinterpolateangles  
     vectorutilities, 143  
 linearinterpolatecyclic  
     vectorutilities, 143  
 loglinearinterpolate  
     vectorutilities, 145  
 loglinearinterpolatecyclic  
     vectorutilities, 145  
 logqfrombeta  
     conversionfunctions, 37  
 longmax  
     precision, 133  
 low  
     utilities, 140  
 lun  
     feedbackdll::feedbackdata, 169  
  
 m\_z2  
     typedefinitionsrovertopping::tpovertoppinginput,  
         186  
 mainModuleRTOvertopping.f90, 209  
 mainmodulertoovertopping, 121  
     calculateovertopping, 121  
     calculateovertoppingnegativefreeboard, 122  
     calculateovertoppingsection, 122  
     calculatewaveovertopping, 123  
     checkinputdata, 124  
     checkmodelfactors, 125  
     convertovertoppinginput, 125  
     interpolateresultssections, 126  
 margindiff  
     typedefinitionsrovertopping, 137  
 margingrad  
     typedefinitionsrovertopping, 137  
 maxfilenamelength  
     modulelogging, 127  
 mergesequentialberms  
     geometrymodulertoovertopping, 116  
 message\_length  
     physics\_foreland, 131  
  
 mgh  
     physics\_bretschneider::tpbretschneider, 178  
 mgt  
     physics\_bretschneider::tpbretschneider, 178  
 mode\_all  
     ftnunit, 104  
 mode\_list  
     ftnunit, 104  
 mode\_single  
     ftnunit, 104  
 ModuleLogging.f90, 210  
 modulelogging, 126  
     currentlogging, 126  
     maxfilenamelength, 127  
 modulelogging::tlogging, 176  
     filename, 177  
     verbosity, 177  
 mz2\_max  
     typedefinitionsrovertopping, 137  
 mz2\_min  
     typedefinitionsrovertopping, 137  
  
 nbermsegments  
     typedefinitionsrovertopping::tpgeometry, 180  
 ncoordinates  
     typedefinitionsrovertopping::tpgeometry, 180  
 nofails  
     ftnunit, 105  
 nofails\_prev  
     ftnunit, 105  
 normal  
     overlappinginterface::overlappinggeometrytype,  
         170  
     overlappinginterface::overlappinggeometrytypef,  
         172  
 normalize  
     vectorutilities, 146  
 noruns  
     ftnunit, 105  
 notests\_failed  
     ftnunit, 105  
 notests\_ignored  
     ftnunit, 105  
 notests\_run  
     ftnunit, 105  
 notests\_work\_in\_progress  
     ftnunit, 105  
 npoints  
     overlappinginterface::overlappinggeometrytype,  
         170  
     overlappinginterface::overlappinggeometrytypef,  
         172  
  
 onfatalerrorstop  
     feedback\_parameters, 56  
 onfatalerrorthrowexception  
     feedback\_parameters, 56  
 onfatalerrorunittest  
     feedback\_parameters, 56

onlyfile  
Feedback library, 32

onlyscreen  
Feedback library, 32

output  
feedbackdll::feedbackdata, 169

outputtdll  
feedbackdll, 61

overtoppingInterface.f90, 210

overtoppinginterface, 127  
varmodelfactorcriticalovertopping, 127

overtoppinginterface::overtoppinggeometrytype, 170  
normal, 170  
npoints, 170  
roughness, 170  
xcoords, 171  
ycoords, 171

overtoppinginterface::overtoppinggeometrytypef, 171  
normal, 172  
npoints, 172  
roughness, 172  
xcoords, 172  
ycoords, 172

overtoppinginterface::tpprofilecoordinate, 187  
roughness, 187  
xcoordinate, 187  
zcoordinate, 188

p  
Feedback library, 24

pfrombeta  
conversionfunctions, 39

phi  
typedefinitionsrovertopping::tupload, 182

physics, 127

physics.f90, 211

physics\_breakwater, 127  
calculatebreakwater, 127

physics\_breakwatertypesenumerations, 128  
breakwatercaisson, 128  
breakwaternotpresent, 128  
breakwaterrubblemound, 128  
breakwaterverticalwall, 129

physics\_bretschneider, 129  
calculatewavebretschneider, 129

physics\_bretschneider::tpbretschneider, 177  
d, 178  
f, 178  
mgh, 178  
mgt, 178  
v, 178

physics\_calculateforeland  
physics\_foreland, 130

physics\_foreland, 129  
message\_length, 131  
physics\_calculateforeland, 130

physics\_wavereduction, 131  
calculatewavereduction, 131

physics\_wavereduction::tpwavereduction, 188

h, 189  
hs, 189  
ht, 189

pi  
precision, 133

poproutine  
feedbackdll, 61

qfrombeta  
conversionfunctions, 39

precision, 132  
almostinf, 132  
almostzero, 132  
betalimit, 132  
dp, 133  
emconstant, 133  
gravityconstant, 133  
longmax, 133  
pi, 133  
rholimit, 133  
shortmax, 133  
sp, 133  
wp, 133

precision.f90, 211

previous  
ftnunit, 105

printcontext  
feedbackdll, 61

printmessage  
feedbackdll, 62

printstack  
feedbackdll, 62

proc  
ftnunit::proc, 173

profileinstructure  
zfunctionsztovertopping, 155

progressreportarray  
Feedback library, 25  
feedback::progressreport, 174

progressreportdouble  
Feedback library, 26  
feedback::progressreport, 175

progressreportint  
Feedback library, 26  
feedback::progressreport, 175

progressreportreal  
Feedback library, 26  
feedback::progressreport, 175

progressreporttxt  
Feedback library, 26  
feedback::progressreport, 175

progressreportvalueandarray  
Feedback library, 27  
feedback::progressreport, 176

psi  
typedefinitionsrovertopping::tpgeometry, 180

pushroutine  
feedbackdll, 63

qfrombeta

conversionfunctions, 40  
**qmin**  
 conversionfunctions, 41  
**qo**  
 typedefinitionsrtoovertopping::tpovertopping, 183  
  
**readtable**  
 fileutilities, 65  
**realrootscubicfunction**  
 formulamodulertoovertopping, 72  
**reductionfactorforeshore**  
 typedefinitionsrtoovertopping::tpovertoppinginput, 186  
**relaxationfactor**  
 typedefinitionsrtoovertopping::tpovertoppinginput, 186  
**removeberms**  
 geometrymodulertoovertopping, 116  
**removedikesegments**  
 geometrymodulertoovertopping, 117  
**removefile**  
 fileutilities, 65  
**resetfatalerroroccured**  
 Feedback library, 27  
**returntimefrombeta**  
 conversionfunctions, 41  
**rfactor\_max**  
 typedefinitionsrtoovertopping, 137  
**rfactor\_min**  
 typedefinitionsrtoovertopping, 137  
**rholimit**  
 precision, 133  
**rootsdepressedcubic**  
 formulamodulertoovertopping, 74  
**rootsgeneralcubic**  
 formulamodulertoovertopping, 74  
**roughness**  
 overtoppinginterface::overtoppinggeometrytype, 170  
 overtoppinginterface::overtoppinggeometrytypef, 172  
 overtoppinginterface::tpprofilecoordinate, 187  
**roughnessfactors**  
 typedefinitionsrtoovertopping::tpgeometry, 180  
**routine**  
 feedbackdll::feedbackdata, 169  
**run\_test\_level**  
 ftnunit, 105  
**runtests**  
 ftnunit, 99  
**runtests\_final**  
 ftnunit, 100  
**runtests\_init**  
 ftnunit, 100  
**runtests\_init\_priv**  
 ftnunit, 101  
  
**screenandfile**  
 Feedback library, 32  
  
**segmentslopes**  
 typedefinitionsrtoovertopping::tpgeometry, 180  
**segmenttypes**  
 typedefinitionsrtoovertopping::tpgeometry, 180  
**setallowlogging**  
 Feedback library, 28  
 feedbackdll, 63  
**setfatalerroraction**  
 Feedback library, 28  
**setfatalerrorexpected**  
 Feedback library, 28  
**setfatalerrormessage**  
 feedbackdll, 63  
**setfatalerroroccured**  
 Feedback library, 29  
**setonfatalerroraction**  
 feedbackdll, 63  
**setruntestlevel**  
 ftnunit, 102  
**settesttitle**  
 ftnunit, 102  
**setworkingdir**  
 ftnunit, 102  
**shortmax**  
 precision, 133  
**single\_test**  
 ftnunit, 105  
**slope\_max**  
 typedefinitionsrtoovertopping, 137  
**slope\_min**  
 typedefinitionsrtoovertopping, 137  
**sortandlocate**  
 vectorutilities, 146  
**sp**  
 feedback, 55  
 precision, 133  
**splitcrosssection**  
 geometrymodulertoovertopping, 118  
**ssort**  
 vectorutilities, 147  
**stacklevel**  
 feedbackdll::feedbackdata, 169  
**stepsize**  
 vectorutilities, 147  
**subptrtestinit**  
 ftnunit, 105  
**subptrtestprograminit**  
 ftnunit, 105  
  
**test**  
 ftnunit, 102  
**test\_mode**  
 ftnunit, 106  
**testname**  
 ftnunit, 106  
**testno**  
 ftnunit, 106  
**testtitle**  
 ftnunit, 106

testwithlevel  
  ftnunit, 103

throw  
  Feedback library, 29

tm\_10  
  typedefinitionsrovertopping::tpovertopping, 182

to\_lower  
  utilities, 139

to\_upper  
  utilities, 140

todll  
  Feedback library, 30

trapeziumrule  
  vectorutilities, 148

triangularinterpolation  
  interpolationtriangular, 120

typeDefinitionsRTOvertopping.f90, 212

typedefinitionsrovertopping, 134  
  berm\_max, 135  
  berm\_min, 135  
  fb\_max, 135  
  fb\_min, 135  
  fn\_max, 135  
  fn\_min, 135  
  foreshore\_max, 136  
  foreshore\_min, 136  
  frunup1\_max, 136  
  frunup1\_min, 136  
  frunup2\_max, 136  
  frunup2\_min, 136  
  frunup3\_max, 136  
  frunup3\_min, 136  
  fs\_max, 136  
  fs\_min, 137  
  margindiff, 137  
  margingrad, 137  
  mz2\_max, 137  
  mz2\_min, 137  
  rfactor\_max, 137  
  rfactor\_min, 137  
  slope\_max, 137  
  slope\_min, 137  
  xdiff\_min, 138  
  z2\_iter\_max, 138  
  z2\_margin, 138

typedefinitionsrovertopping::tpgeometry, 179  
  nbermsegments, 180  
  ncoordinates, 180  
  psi, 180  
  roughnessfactors, 180  
  segmentslopes, 180  
  segmenttypes, 180  
  xcoorddiff, 180  
  xcoordinates, 180  
  ycoorddiff, 181  
  ycoordinates, 181

typedefinitionsrovertopping::tpload, 181  
  h, 182

hm0, 182  
phi, 182  
tm\_10, 182

typedefinitionsrovertopping::tpovertopping, 182  
  qo, 183  
  z2, 183

typedefinitionsrovertopping::tpovertoppinginput, 184  
  computedovertopping, 185  
  criticalovertopping, 185  
  factordeterminationq\_b\_f\_b, 185  
  factordeterminationq\_b\_f\_n, 185  
  frunup1, 185  
  frunup2, 185  
  frunup3, 186  
  fshallow, 186  
  m\_z2, 186  
  reductionfactorforeshore, 186  
  relaxationfactor, 186  
  typerunup, 186

typerunup  
  typedefinitionsrovertopping::tpovertoppinginput, 186

ulimit  
  conversionfunctions, 41

upperlog  
  conversionfunctions, 41

utilities, 138  
  cap, 140  
  getfreelunumber, 138  
  low, 140  
  to\_lower, 139  
  to\_upper, 140

utilities.f90, 214

v  
  physics\_bretschneider::tpbretschneider, 178

validateinput  
  dllovertopping, 45

validateinputf  
  dllovertopping, 45

varmodelfactorcriticalovertopping  
  overtoppinginterface, 127

vectorUtilities.f90, 214

vectorutilities, 140  
  interpolateline, 141  
  integratewithexponentcomponent, 141  
  linearinterpolate, 141  
  linearinterpolateangles, 143  
  linearinterpolatecyclic, 143  
  loglinearinterpolate, 145  
  loglinearinterpolatecyclic, 145  
  normalize, 146  
  sortandlocate, 146  
  ssort, 147  
  stepsize, 147  
  trapeziumrule, 148

verbosebasic  
  Feedback library, 32

**verbosedebugging**  
 Feedback library, 32  
**verbosetailed**  
 Feedback library, 32  
**verbosenone**  
 Feedback library, 33  
**verbosity**  
 modulelogging::tlogging, 177  
**verbositylevel**  
 feedbackdll::feedbackdata, 169  
**versionnumber**  
 dlovertopping, 46  
**warningmessagedouble**  
 Feedback library, 31  
 feedback::warningmessage, 190  
**warningmessageint**  
 Feedback library, 31  
 feedback::warningmessage, 190  
**warningmessagereal**  
 Feedback library, 31  
 feedback::warningmessage, 190  
**warningmessagetxt**  
 Feedback library, 31  
 feedback::warningmessage, 190  
**waveParametersUtilities.f90**, 215  
**waveReduction.f90**, 216  
**waveRunup.f90**, 216  
**waveparametersutilities**, 148  
 computewaveperiod, 149  
 computewavesteeppness, 149  
**waverunup**, 149  
 convergedwithresidu, 150  
 determinestartingvalue, 150  
 findsmallestresidu, 150  
 iterationwaverunup, 151  
**work\_in\_progress**  
 ftnunit, 106  
**wp**  
 feedback, 55  
 precision, 133  
**write\_header**  
 ftnunit.f90, 206  
**writeUtilities.f90**, 217  
**writecrosssection**  
 geometrymodulertovertopping, 119  
**writetablevalues**  
 fileutilities, 65  
**writetodevice\_line**  
 writeutilities, 152  
 writeutilities::writetodevice, 191  
**writetodevice\_text\_real**  
 writeutilities, 152  
 writeutilities::writetodevice, 191  
**writetodevicechars**  
 writeutilities, 153  
**writetodll**  
 feedbackdll, 63  
**writeutilities**, 151

inttostr, 152  
 writetodevice\_line, 152  
 writetodevice\_text\_real, 152  
 writetodevicechars, 153  
**writeutilities::writetodevice**, 191  
 writetodevice\_line, 191  
 writetodevice\_text\_real, 191

**xcoorddf**  
 typedefinitionsrtovertopping::tpgeometry, 180  
**xcoordinate**  
 overtoppinginterface::tpprofilecoordinate, 187  
**xcoordinates**  
 typedefinitionsrtovertopping::tpgeometry, 180  
**xcoords**  
 overtoppinginterface::overtoppinggeometrytype, 171  
 overtoppinginterface::overtoppinggeometrytypef, 172  
**xdiff\_min**  
 typedefinitionsrtovertopping, 138

**ycoorddf**  
 typedefinitionsrtovertopping::tpgeometry, 181  
**ycoordinates**  
 typedefinitionsrtovertopping::tpgeometry, 181  
**ycoords**  
 overtoppinginterface::overtoppinggeometrytype, 171  
 overtoppinginterface::overtoppinggeometrytypef, 172

**z2**  
 typedefinitionsrtovertopping::tpovertopping, 183  
**z2\_iter\_max**  
 typedefinitionsrtovertopping, 138  
**z2\_margin**  
 typedefinitionsrtovertopping, 138  
**zFunctionsWTIOvertopping.f90**, 217  
**zcoordinate**  
 overtoppinginterface::tpprofilecoordinate, 188  
**zfunclogratios**  
 zfunctionswtiovertopping, 156  
**zfunctionswtiovertopping**, 153  
 adjustprofile, 154  
 calculateqorto, 154  
 profileinstructure, 155  
 zfunclogratios, 156