

SAMSIM

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model V2.0</b>	<b>1</b>
<b>2</b>	<b>Modules Index</b>	<b>5</b>
2.1	Modules List . . . . .	5
<b>3</b>	<b>File Index</b>	<b>7</b>
3.1	File List . . . . .	7
<b>4</b>	<b>Module Documentation</b>	<b>9</b>
4.1	mo_data Module Reference . . . . .	9
4.1.1	Detailed Description . . . . .	14
4.1.2	Variable Documentation . . . . .	15
4.1.2.1	albedo . . . . .	15
4.1.2.2	albedo_flag . . . . .	15
4.1.2.3	alpha_flux_instable . . . . .	15
4.1.2.4	alpha_flux_stable . . . . .	15
4.1.2.5	atmoflux_flag . . . . .	15
4.1.2.6	bgc_abs . . . . .	15
4.1.2.7	bgc_bottom . . . . .	16
4.1.2.8	bgc_br . . . . .	16
4.1.2.9	bgc_bu . . . . .	16
4.1.2.10	bgc_flag . . . . .	16
4.1.2.11	bgc_total . . . . .	16
4.1.2.12	bottom_flag . . . . .	16
4.1.2.13	boundflux_flag . . . . .	16

4.1.2.14	bulk_salin	16
4.1.2.15	debug_flag	16
4.1.2.16	dt	16
4.1.2.17	energy_stored	17
4.1.2.18	fl_brine_bgc	17
4.1.2.19	fl_lat	17
4.1.2.20	fl_lw	17
4.1.2.21	fl_lw_input	17
4.1.2.22	fl_m	17
4.1.2.23	fl_q	17
4.1.2.24	fl_q_bottom	17
4.1.2.25	fl_q_snow	17
4.1.2.26	fl_rad	17
4.1.2.27	fl_rest	18
4.1.2.28	fl_s	18
4.1.2.29	fl_sen	18
4.1.2.30	fl_sw	18
4.1.2.31	fl_sw_input	18
4.1.2.32	flood_flag	18
4.1.2.33	flush_flag	18
4.1.2.34	flush_h	18
4.1.2.35	flush_h_old	18
4.1.2.36	flush_heat_flag	18
4.1.2.37	flush_question	18
4.1.2.38	flush_v	19
4.1.2.39	flush_v_old	19
4.1.2.40	format_bgc	19
4.1.2.41	format_integer	19
4.1.2.42	format_melt	19
4.1.2.43	format_perm	19

4.1.2.44	format_psi	19
4.1.2.45	format_snow	19
4.1.2.46	format_t	19
4.1.2.47	format_t2m_top	19
4.1.2.48	format_thick	19
4.1.2.49	freeboard	19
4.1.2.50	freeboard_snow_flag	19
4.1.2.51	freshwater	19
4.1.2.52	grav_drain	19
4.1.2.53	grav_flag	20
4.1.2.54	grav_heat_flag	20
4.1.2.55	grav_salt	20
4.1.2.56	grav_temp	20
4.1.2.57	h	20
4.1.2.58	h_abs	20
4.1.2.59	h_abs_snow	20
4.1.2.60	harmonic_flag	20
4.1.2.61	i	20
4.1.2.62	i_time	20
4.1.2.63	i_time_out	21
4.1.2.64	k	21
4.1.2.65	lab_snow_flag	21
4.1.2.66	length_input	21
4.1.2.67	length_input_lab	21
4.1.2.68	liquid_precip	21
4.1.2.69	m	21
4.1.2.70	m_snow	21
4.1.2.71	m_total	21
4.1.2.72	melt_err	21
4.1.2.73	melt_thick	22

4.1.2.74	melt_thick_output	22
4.1.2.75	melt_thick_snow	22
4.1.2.76	melt_thick_snow_old	22
4.1.2.77	n_active	22
4.1.2.78	n_bgc	22
4.1.2.79	n_bottom	22
4.1.2.80	n_middle	22
4.1.2.81	n_time_out	22
4.1.2.82	n_top	22
4.1.2.83	nlayer	22
4.1.2.84	ocean_flux_input	23
4.1.2.85	ocean_t_input	23
4.1.2.86	perm	23
4.1.2.87	phi	23
4.1.2.88	phi_s	23
4.1.2.89	precip_flag	23
4.1.2.90	precip_input	23
4.1.2.91	precipinput	23
4.1.2.92	prescribe_flag	23
4.1.2.93	psi_g	23
4.1.2.94	psi_g_snow	24
4.1.2.95	psi_l	24
4.1.2.96	psi_l_snow	24
4.1.2.97	psi_s	24
4.1.2.98	psi_s_snow	24
4.1.2.99	q	24
4.1.2.100	ray	24
4.1.2.101	s_abs	24
4.1.2.102	s_abs_snow	24
4.1.2.103	s_br	24

4.1.2.104 s_bu . . . . .	25
4.1.2.105 s_bu_bottom . . . . .	25
4.1.2.106 s_total . . . . .	25
4.1.2.107 salt_flag . . . . .	25
4.1.2.108 snow_flush_flag . . . . .	25
4.1.2.109 snow_precip_flag . . . . .	25
4.1.2.110 solid_precip . . . . .	25
4.1.2.111 styropor_flag . . . . .	25
4.1.2.112 styropor_input . . . . .	25
4.1.2.113 surface_water . . . . .	25
4.1.2.114 t . . . . .	25
4.1.2.115 t2m . . . . .	26
4.1.2.116 t2m_input . . . . .	26
4.1.2.117 t_bottom . . . . .	26
4.1.2.118 t_freeze . . . . .	26
4.1.2.119 t_snow . . . . .	26
4.1.2.120 t_test . . . . .	26
4.1.2.121 t_top . . . . .	26
4.1.2.122 tank_depth . . . . .	26
4.1.2.123 tank_flag . . . . .	26
4.1.2.124 test . . . . .	26
4.1.2.125 thick . . . . .	27
4.1.2.126 thick_0 . . . . .	27
4.1.2.127 thick_min . . . . .	27
4.1.2.128 thick_snow . . . . .	27
4.1.2.129 thickness . . . . .	27
4.1.2.130 time . . . . .	27
4.1.2.131 time_counter . . . . .	27
4.1.2.132 time_input . . . . .	27
4.1.2.133 time_out . . . . .	27

4.1.2.134	time_total	27
4.1.2.135	tinput	27
4.1.2.136	total_resist	28
4.1.2.137	ttop_input	28
4.1.2.138	turb_flag	28
4.1.2.139	v_ex	28
4.1.2.140	v_g	28
4.1.2.141	v_l	28
4.1.2.142	v_s	28
4.2	mo_flood Module Reference	28
4.2.1	Detailed Description	29
4.2.2	Function/Subroutine Documentation	29
4.2.2.1	flood(freeboard, psi_s, psi_l, S_abs, H_abs, m, T, thick, dt, Nlayer, N_active, T_↵ bottom, S_bu_bottom, H_abs_snow, m_snow, thick_snow, psi_g_snow, debug_↵ _flag, fl_brine_bgc)	29
4.2.2.2	flood_simple(freeboard, S_abs, H_abs, m, thick, T_bottom, S_bu_bottom, H_↵ abs_snow, m_snow, thick_snow, psi_g_snow, Nlayer, N_active, debug_flag)	30
4.3	mo_flush Module Reference	30
4.3.1	Detailed Description	30
4.3.2	Function/Subroutine Documentation	31
4.3.2.1	flush3(freeboard, psi_l, thick, thick_0, S_abs, H_abs, m, T, dt, Nlayer, N_active, T_bottom, S_bu_bottom, melt_thick, debug_flag, flush_heat_flag, melt_err, perm, flush_v, flush_h, psi_g, thick_snow, rho_l, snow_flush_flag, fl_brine_bgc)	31
4.3.2.2	flush4(psi_l, thick, T, thick_0, S_abs, H_abs, m, dt, Nlayer, N_active, N_top, N_↵ middle, N_bottom, melt_thick, debug_flag)	32
4.4	mo_functions Module Reference	32
4.4.1	Detailed Description	32
4.4.2	Function/Subroutine Documentation	33
4.4.2.1	func_albedo(thick_snow, T_snow, psi_l, thick_min, albedo_flag)	33
4.4.2.2	func_density(T, S)	33
4.4.2.3	func_freeboard(N_active, Nlayer, psi_s, psi_g, m, thick, m_snow, freeboard_↵ snow_flag)	33
4.4.2.4	func_sat_o2(T, S_bu)	34



4.4.2.5	<a href="#">func_t_freeze(S_bu, salt_flag)</a>	34
4.4.2.6	<a href="#">sub_input(length_input, fl_sw_input, fl_lw_input, T2m_input, precip_input, time↔ _input)</a>	34
4.4.2.7	<a href="#">sub_melt_snow(melt_thick, thick, thick_snow, H_abs, H_abs_snow, m, m_snow, psi_g_snow)</a>	34
4.4.2.8	<a href="#">sub_melt_thick(psi_l, psi_s, psi_g, T, T_freeze, T_top, fl_Q, thick_snow, dt, melt↔ _thick, thick, thick_min)</a>	35
4.4.2.9	<a href="#">sub_notzflux(time, fl_sw, fl_rest)</a>	35
4.4.2.10	<a href="#">sub_turb_flux(T_bottom, S_bu_bottom, T, S_abs, m, dt, N_bgc, bgc_bottom, bgc_abs)</a>	35
4.5	<a href="#">mo_grav_drain Module Reference</a>	36
4.5.1	<a href="#">Detailed Description</a>	36
4.5.2	<a href="#">Function/Subroutine Documentation</a>	36
4.5.2.1	<a href="#">fl_grav_drain(S_br, S_bu, psi_l, psi_s, psi_g, thick, S_abs, H_abs, T, m, dt, Nlayer, N_active, ray, T_bottom, S_bu_bottom, grav_drain, grav_temp, grav_salt, grav↔ _heat_flag, harmonic_flag, fl_brine_bgc)</a>	36
4.5.2.2	<a href="#">fl_grav_drain_simple(psi_s, psi_l, thick, S_abs, S_br, Nlayer, N_active, ray, grav↔ _drain, harmonic_flag)</a>	37
4.6	<a href="#">mo_grotz Module Reference</a>	38
4.6.1	<a href="#">Detailed Description</a>	38
4.6.2	<a href="#">Function/Subroutine Documentation</a>	39
4.6.2.1	<a href="#">grotz(testcase, description)</a>	39
4.7	<a href="#">mo_heat_fluxes Module Reference</a>	40
4.7.1	<a href="#">Detailed Description</a>	40
4.7.2	<a href="#">Function/Subroutine Documentation</a>	41
4.7.2.1	<a href="#">sub_heat_fluxes()</a>	41
4.8	<a href="#">mo_init Module Reference</a>	42
4.8.1	<a href="#">Detailed Description</a>	42
4.8.2	<a href="#">Function/Subroutine Documentation</a>	42
4.8.2.1	<a href="#">init(testcase)</a>	42
4.8.2.2	<a href="#">sub_allocate(Nlayer, length_input_lab)</a>	43
4.8.2.3	<a href="#">sub_allocate_bgc(Nlayer, N_bgc)</a>	44
4.8.2.4	<a href="#">sub_deallocate</a>	44

4.9	mo_layer_dynamics Module Reference	44
4.9.1	Detailed Description	45
4.9.2	Function/Subroutine Documentation	45
4.9.2.1	layer_dynamics(phi, N_active, Nlayer, N_bottom, N_middle, N_top, m, S_abs, H_abs, thick, thick_0, T_bottom, S_bu_bottom, bottom_flag, debug_flag, melt_↵ thick_output, N_bgc, bgc_abs, bgc_bottom)	45
4.9.2.2	top_grow(Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_↵ abs, thick, N_bgc, bgc_abs)	46
4.9.2.3	top_melt(Nlayer, N_active, N_bottom, N_middle, N_top, thick_0, m, S_abs, H_↵ abs, thick, N_bgc, bgc_abs)	46
4.10	mo_mass Module Reference	46
4.10.1	Detailed Description	47
4.10.2	Function/Subroutine Documentation	47
4.10.2.1	bgc_advection(Nlayer, N_active, N_bgc, fl_brine_bgc, bgc_abs, psi_l, T, S_abs, m, thick, bgc_bottom)	47
4.10.2.2	expulsion_flux(thick, V_ex, Nlayer, N_active, psi_g, fl_m, m)	48
4.10.2.3	mass_transfer(Nlayer, N_active, T, H_abs, S_abs, S_bu, T_bottom, S_bu_↵ bottom, fl_m)	48
4.11	mo_output Module Reference	48
4.11.1	Detailed Description	49
4.11.2	Function/Subroutine Documentation	50
4.11.2.1	output(Nlayer, T, psi_s, psi_l, thick, S_bu, ray, format_T, format_psi, format_↵ _thick, format_snow, freeboard, thick_snow, T_snow, psi_l_snow, psi_s_snow, energy_stored, freshwater, total_resist, thickness, bulk_salin, grav_drain, grav_↵ _salt, grav_temp, T2m, T_top, perm, format_perm, flush_v, flush_h, psi_g, melt_↵ _thick_output, format_melt)	50
4.11.2.2	output_begin(Nlayer, debug_flag, format_T, format_psi, format_thick, format_↵ snow, format_T2m_top, format_perm, format_melt)	50
4.11.2.3	output_begin_bgc(Nlayer, N_bgc, format_bgc)	50
4.11.2.4	output_bgc(Nlayer, N_active, bgc_bottom, N_bgc, bgc_abs, psi_l, thick, m, format_bgc)	51
4.11.2.5	output_raw(Nlayer, N_active, time, T, thick, S_bu, psi_s, psi_l, psi_g)	51
4.11.2.6	output_raw_lay(Nlayer, N_active, H_abs, m, S_abs, thick, string)	51
4.11.2.7	output_raw_snow(time, T_snow, thick_snow, S_abs_snow, m_snow, psi_s_snow, psi_l_snow, psi_g_snow)	51

4.11.2.8	output_settings(description, testcase, N_top, N_bottom, Nlayer, fl_q_bottom, T_bottom, S_bu_bottom, thick_0, time_out, time_total, dt, boundflux_flag, atmoflux_flag, albedo_flag, grav_flag, flush_flag, flood_flag, grav_heat_flag, flush_heat_flag, harmonic_flag, prescribe_flag, salt_flag, turb_flag, bottom_flag, tank_flag, precip_flag, bgc_flag, N_bgc, k_snow_flush)	52
4.12	mo_parameters Module Reference	52
4.12.1	Detailed Description	54
4.12.2	Variable Documentation	54
4.12.2.1	bbeta	54
4.12.2.2	c_l	54
4.12.2.3	c_s	54
4.12.2.4	c_s_beta	54
4.12.2.5	emissivity_ice	55
4.12.2.6	emissivity_snow	55
4.12.2.7	extinc	55
4.12.2.8	gas_snow_ice	55
4.12.2.9	gas_snow_ice2	55
4.12.2.10	grav	55
4.12.2.11	k_l	55
4.12.2.12	k_s	55
4.12.2.13	k_snow_flush	55
4.12.2.14	k_styropor	55
4.12.2.15	kappa_l	56
4.12.2.16	latent_heat	56
4.12.2.17	max_flux_plate	56
4.12.2.18	mu	56
4.12.2.19	neg_free	56
4.12.2.20	para_flush_gamma	56
4.12.2.21	para_flush_horiz	56
4.12.2.22	penetr	56
4.12.2.23	pi	56
4.12.2.24	psi_s_min	56

4.12.2.25 psi_s_top_min . . . . .	56
4.12.2.26 ratio_flood . . . . .	57
4.12.2.27 ray_crit . . . . .	57
4.12.2.28 ref_salinity . . . . .	57
4.12.2.29 rho_l . . . . .	57
4.12.2.30 rho_s . . . . .	57
4.12.2.31 rho_snow . . . . .	57
4.12.2.32 sigma . . . . .	57
4.12.2.33 turb_a . . . . .	57
4.12.2.34 turb_b . . . . .	57
4.12.2.35 wp . . . . .	57
4.12.2.36 x_grav . . . . .	57
4.12.2.37 zerok . . . . .	57
4.13 mo_snow Module Reference . . . . .	58
4.13.1 Detailed Description . . . . .	58
4.13.2 Function/Subroutine Documentation . . . . .	58
4.13.2.1 func_k_snow(m_snow, thick_snow) . . . . .	58
4.13.2.2 snow_coupling(H_abs_snow, phi_s, T_snow, H_abs, H, phi, T, m_snow, S_abs↔ _snow, m, S_bu) . . . . .	59
4.13.2.3 snow_precip(m_snow, H_abs_snow, thick_snow, psi_s_snow, dt, liquid_precip↔ _in, T2m, solid_precip_in) . . . . .	59
4.13.2.4 snow_precip_0(H_abs, S_abs, m, T, dt, liquid_precip_in, T2m, solid_precip_in) . . . . .	59
4.13.2.5 snow_thermo(psi_l_snow, psi_s_snow, psi_g_snow, thick_snow, S_abs_snow, H_abs_snow, m_snow, T_snow, m, thick, H_abs) . . . . .	60
4.13.2.6 snow_thermo_meltwater(psi_l_snow, psi_s_snow, psi_g_snow, thick_snow, S_↔ abs_snow, H_abs_snow, m_snow, T_snow, m, thick, H_abs, melt_thick_snow) . . . . .	60
4.13.2.7 sub_fl_q_0_snow(m_snow, thick_snow, T_snow, T_bound, fl_Q) . . . . .	61
4.13.2.8 sub_fl_q_0_snow_thin(m_snow, thick_snow, T_snow, psi_s, psi_l, psi_g, thick, T_bound, fl_Q_snow) . . . . .	61
4.13.2.9 sub_fl_q_snow(m_snow, thick_snow, T_snow, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q) . . . . .	61
4.14 mo_testcase_specifics Module Reference . . . . .	61
4.14.1 Detailed Description . . . . .	62

4.14.2	Function/Subroutine Documentation	62
4.14.2.1	sub_test1(time, T_top)	62
4.14.2.2	sub_test2(time, T2m)	63
4.14.2.3	sub_test3(time, liquid_precip, solid_precip)	63
4.14.2.4	sub_test34(time, T2m)	63
4.14.2.5	sub_test4(time, fl_q_bottom)	63
4.14.2.6	sub_test6(time, T2m)	63
4.14.2.7	sub_test9(time, T2m)	64
4.15	mo_thermo_functions Module Reference	64
4.15.1	Detailed Description	64
4.15.2	Function/Subroutine Documentation	65
4.15.2.1	expulsion(phi, thick, m, psi_s, psi_l, psi_g, V_ex)	65
4.15.2.2	func_ddt_s_br(T)	65
4.15.2.3	func_s_br(T, S_bu)	65
4.15.2.4	gett(H, S_bu, T_in, T, phi, k)	66
4.15.2.5	sub_fl_q(psi_s_1, psi_l_1, psi_g_1, thick_1, T_1, psi_s_2, psi_l_2, psi_g_2, thick_2, T_2, fl_Q)	67
4.15.2.6	sub_fl_q_0(psi_s, psi_l, psi_g, thick, T, T_bound, direct_flag, fl_Q)	67
4.15.2.7	sub_fl_q_styropor(k_styropor, fl_Q)	67
<b>5</b>	<b>File Documentation</b>	<b>69</b>
5.1	gpl_license.txt File Reference	69
5.1.1	Function Documentation	74
5.1.1.1	Copyright(C) 2007 Free Software Foundation	74
5.1.2	Variable Documentation	74
5.1.2.1	not	75
5.1.2.2	Version	76
5.2	mo_data.f90 File Reference	76
5.3	mo_flood.f90 File Reference	81
5.4	mo_flush.f90 File Reference	82
5.5	mo_functions.f90 File Reference	82

5.6	<a href="#">mo_grav_drain.f90 File Reference</a>	83
5.7	<a href="#">mo_grotz.f90 File Reference</a>	83
5.8	<a href="#">mo_heat_fluxes.f90 File Reference</a>	83
5.9	<a href="#">mo_init.f90 File Reference</a>	84
5.10	<a href="#">mo_layer_dynamics.f90 File Reference</a>	84
5.11	<a href="#">mo_mass.f90 File Reference</a>	84
5.12	<a href="#">mo_output.f90 File Reference</a>	85
5.13	<a href="#">mo_parameters.f90 File Reference</a>	86
5.14	<a href="#">mo_snow.f90 File Reference</a>	87
5.15	<a href="#">mo_testcase_specifics.f90 File Reference</a>	88
5.16	<a href="#">mo_thermo_functions.f90 File Reference</a>	88
5.17	<a href="#">SAMSIM.f90 File Reference</a>	89
5.17.1	<a href="#">Function/Subroutine Documentation</a>	89
5.17.1.1	<a href="#">samsim</a>	89
<b>Index</b>		<b>91</b>

## Chapter 1

# SAMSIM Semi-Adaptive Multi-phase Sea-Ice Model V2.0

V1.0 of this model was developed from scratch by Philipp Griewank during and after his PhD at Max Planck Institute of Meteorology from 2010-2014. Most elements of the model are described in the two papers "Insights into brine dynamics and sea ice desalination from a 1-D model study of gravity drainage" and "A 1-D modelling study of Arctic sea-ice salinity" of Griewank and Notz" which are both included in the repository. V2.0 of SAMSIM is a minor expansion of V1.0 released in 2018. Most work was done by Niels Fuchs as part of his Master's thesis "The impact of snow on sea-ice salinity" at the Max Planck Institute of Meteorology from 2016-2017 (thesis also in repository). The biggest change is an improvement of the flushing parametrization, as well as the settings and forcings for a large amount of laboratory experiments Niels conducted, making it possible to run lab testcases with snow.

[SAMSIM.f90](#) is the root program of the SAMSIM, the 1D thermodynamic Semi-Adaptive Multi-phase Sea-Ice Model. However, in [SAMSIM.f90](#) only the testcase and description thread are specified, which are then passed on to [mo\\_grotz](#), which is where most of the actual work is done, including timestepping. The code is intended to be understandable and subroutines, modules, functions, parameters, and global variables all have (more or less) doxygen compatible descriptions.

WARNING: SAMSIM was developed and was/is used for scientific purposes. It likely contains a few undetected bugs, can easily be crashed by using non-logical input settings, and some of the descriptions and comments may be outdated. Always check the plausibility of the model results!

Getting started:

- A number of testcases are implemented in SAMSIM. Testcases 1, 2, 3, and 4 are intended as standard testcases which should give a first time user a feel for the model capabilities and serve as a basis to set up custom testcases. To familiarize yourself with the model I suggest running testcases 1-3 and plotting the output with the python plotting scripts provided. The details of each testcase are commented in [mo\\_init.f90](#), and each plot script begins with a list of steps required.

Running SAMSIM the first times.

- Make sure that all .f90 files are located in the same folder with the makefile.
- Open the makefile with your editor of choice and choose the compiler and flags of choice.
- Open [SAMSIM.f90](#), set a testcase from 1-3, and edit the description string to fit your purpose.
- Use make to compile the code, which produces the executable samsim.x .
- Make sure a folder "output" is located in the folder with samsim.x .

- Execute SAMSIM by running `samsim.x` .
- Go into output folder
- Copy the plot script from `plotscripts` to output
- Follow the directions written in the `plotscripts` to plot the output.

Running testcase 4.

- In contrast to testcase 1-3, testcase four requires input files. Input data for testcase is provided in the input folder. Choose one of the subfolders from `input/ERA-interim/`, copy the `*.input` files into the folder with the code, and run the executable `.samsim.x` .

Following modules have a good documentation (both in the code and `refman.pdf`)

- [mo\\_heat\\_fluxes.f90](#)
- [mo\\_layer\\_dynamics.f90](#)
- [mo\\_init.f90](#)

Biogeochemical tracers can be activated with `bgc_flag=2`.

- Warning! This feature was implemented at the end of my PhD and not used much. As a result it has not been thoroughly tested.
- The model will track `Nbgc` number of individual tracer.
- Especially if you are interested in dissolved gases, you should first make yourself familiar with the `bgc_↔` advection subroutine in [mo\\_mass.f90](#).

Know issues/Tips and Tricks:

- If code changes have no effect, run "make clean" and then "make", for unknown reasons this is often needed when making changes to [mo\\_parameters.f90](#)
- When bug hunting increase `thick_0` and `dt`, this way the model runs faster, and the output is easier to sort through.
- Use `debug_flag= 2` to output data of each layer at each timestep. Be careful, the output size can become very large!
- Check `dat_settings` to keep track of runs, and use the description variable to keep track of experiments.
- Contact me :)



**Author**

Philipp Griewank

**COPYRIGHT**

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

**Revision History**

Started by Philipp Griewank 2014-05-05  
nothing changed here by Niels Fuchs, MPIMET (2017-03-01)  
License changed by Philipp Griewank 2018-05-22  
V2.0 finalized by Philipp Griewank 2018-08-09



## Chapter 2

# Modules Index

### 2.1 Modules List

Here is a list of all modules with brief descriptions:

<a href="#">mo_data</a>	Sets data and contains all flag descriptions . . . . .	9
<a href="#">mo_flood</a>	Computes the fluxes caused by liquid flooding the snow layer . . . . .	28
<a href="#">mo_flush</a>	Contains various subroutines for flushing . . . . .	30
<a href="#">mo_functions</a>	Module houses functions which have no home :( . . . . .	32
<a href="#">mo_grav_drain</a>	Computes the Salt fluxes caused by gravity drainage . . . . .	36
<a href="#">mo_grotz</a>	The most important module of SAMSIM . . . . .	38
<a href="#">mo_heat_fluxes</a>	Computes all heat fluxes . . . . .	40
<a href="#">mo_init</a>	Allocates Arrays and sets initial data for a given testcase for SAMSIM . . . . .	42
<a href="#">mo_layer_dynamics</a>	Mo_layer_dynamics contains all subroutines for the growth and shrinking of layer thickness . . . . .	44
<a href="#">mo_mass</a>	Regulates mass transfers and their results . . . . .	46
<a href="#">mo_output</a>	All things output . . . . .	48
<a href="#">mo_parameters</a>	Module determines physical constants to be used by the SAMSIM Seaice model . . . . .	52
<a href="#">mo_snow</a>	Module contains all things directly related to snow . . . . .	58
<a href="#">mo_testcase_specifics</a>	Module contains changes specific testcases require during the main timeloop . . . . .	61
<a href="#">mo_thermo_functions</a>	Contains subroutines and functions related to multi-phase thermodynamics . . . . .	64



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">mo_data.f90</a>	76
<a href="#">mo_flood.f90</a>	81
<a href="#">mo_flush.f90</a>	82
<a href="#">mo_functions.f90</a>	82
<a href="#">mo_grav_drain.f90</a>	83
<a href="#">mo_grotz.f90</a>	83
<a href="#">mo_heat_fluxes.f90</a>	83
<a href="#">mo_init.f90</a>	84
<a href="#">mo_layer_dynamics.f90</a>	84
<a href="#">mo_mass.f90</a>	84
<a href="#">mo_output.f90</a>	85
<a href="#">mo_parameters.f90</a>	86
<a href="#">mo_snow.f90</a>	87
<a href="#">mo_testcase_specifics.f90</a>	88
<a href="#">mo_thermo_functions.f90</a>	88
<a href="#">SAMSIM.f90</a>	89



## Chapter 4

# Module Documentation

### 4.1 mo\_data Module Reference

Sets data and contains all flag descriptions.

#### Variables

- `real(wp), dimension(:), allocatable h`  
*Enthalpy [J].*
- `real(wp), dimension(:), allocatable h_abs`  
*specific Enthalpy [J/kg]*
- `real(wp), dimension(:), allocatable q`  
*Heat in layer [J].*
- `real(wp), dimension(:), allocatable fl_q`  
*Heat flux between layers [J/s].*
- `real(wp), dimension(:), allocatable t`  
*Temperature [C].*
- `real(wp), dimension(:), allocatable s_bu`  
*Bulk Salinity [g/kg].*
- `real(wp), dimension(:), allocatable fl_s`  
*Salinity flux [(g/s)].*
- `real(wp), dimension(:), allocatable s_abs`  
*Absolute Salinity [g].*
- `real(wp), dimension(:), allocatable s_br`  
*Brine salinity [g/kg].*
- `real(wp), dimension(:), allocatable thick`  
*Layer thickness [m].*
- `real(wp), dimension(:), allocatable m`  
*Mass [kg].*
- `real(wp), dimension(:), allocatable fl_m`  
*Mass fluxes between layers [kg].*
- `real(wp), dimension(:), allocatable v_s`  
*Volume [m<sup>3</sup>] of solid.*
- `real(wp), dimension(:), allocatable v_l`  
*Volume [m<sup>3</sup>] of liquid.*

- `real(wp), dimension(:), allocatable v_g`  
*Volume [m<sup>3</sup>] of gas.*
- `real(wp), dimension(:), allocatable v_ex`  
*Volume of brine due expelled due to freezing [m<sup>3</sup>] of solid, gas & liquid.*
- `real(wp), dimension(:), allocatable phi`  
*Solid mass fraction.*
- `real(wp), dimension(:), allocatable psi_s`  
*Solid volume fraction.*
- `real(wp), dimension(:), allocatable psi_l`  
*Liquid volume fraction.*
- `real(wp), dimension(:), allocatable psi_g`  
*Gas volume fraction.*
- `real(wp), dimension(:), allocatable ray`  
*Rayleigh number of each layer.*
- `real(wp), dimension(:), allocatable perm`
- `real(wp), dimension(:), allocatable flush_v`
- `real(wp), dimension(:), allocatable flush_h`
- `real(wp), dimension(:), allocatable flush_v_old`
- `real(wp), dimension(:), allocatable flush_h_old`  
*Permeability [?].*
- `real(wp) dt`  
*Timestep [s].*
- `real(wp) thick_0`  
*Initial layer thickness [m].*
- `real(wp) time`  
*Time [s].*
- `real(wp) freeboard`  
*Height of ice surface above (or below) waterlevel [m].*
- `real(wp) t_freeze`  
*Freezing temperature [C].*
- `integer nlayer`  
*Number of layers.*
- `integer n_bottom`  
*Number of bottom layers.*
- `integer n_middle`  
*Number of middle layers.*
- `integer n_top`  
*Number of top layers.*
- `integer n_active`  
*Number of Layers active in the present.*
- `integer i`  
*Index, normally used for time.*
- `integer k`  
*Index, normally used for layer.*
- `integer styropor_flag`
- `real(wp) time_out`  
*Time between outputs [s].*
- `real(wp) time_total`  
*Time of simulation [s].*
- `integer i_time`  
*Number of timesteps.*



- integer [i\\_time\\_out](#)  
*Number of timesteps between each output.*
- integer [n\\_time\\_out](#)  
*Counts number of timesteps between output.*
- character \*12000 [format\\_t](#)
- character \*12000 [format\\_psi](#)
- character \*12000 [format\\_thick](#)
- character \*12000 [format\\_snow](#)
- character \*12000 [format\\_integer](#)
- character \*12000 [format\\_t2m\\_top](#)
- character \*12000 [format\\_bgc](#)
- character \*12000 [format\\_melt](#)  
*Format strings for output. Niels(2017) add: melt output.*
- character \*12000 [format\\_perm](#)  
*Niels(2017) add: permeability output.*
- real(wp) [t\\_bottom](#)  
*Temperature of water beneath the ice [C].*
- real(wp) [t\\_top](#)  
*Temperature at the surface [C].*
- real(wp) [s\\_bu\\_bottom](#)  
*Salinity beneath the ice [g/kg].*
- real(wp) [t2m](#)  
*Two meter Temperature [C].*
- real(wp) [fl\\_q\\_bottom](#)  
*Bottom heat flux [J\*s].*
- real(wp) [psi\\_s\\_snow](#)  
*Solid volume fraction of snow layer.*
- real(wp) [psi\\_l\\_snow](#)  
*Liquid volume fraction of snow layer.*
- real(wp) [psi\\_g\\_snow](#)  
*Gas volume fraction of snow layer.*
- real(wp) [phi\\_s](#)  
*Solid mass fraction of snow layer.*
- real(wp) [s\\_abs\\_snow](#)  
*Absolute salinity of snow layer [g].*
- real(wp) [h\\_abs\\_snow](#)  
*Absolute enthalpy of snow layer [J].*
- real(wp) [m\\_snow](#)  
*Mass of snow layer [kg].*
- real(wp) [t\\_snow](#)  
*Temperature of snow layer [C].*
- real(wp) [thick\\_snow](#)
- real(wp) [test](#)  
*Thickness of snow layer [m].*
- real(wp) [liquid\\_precip](#)  
*Liquid precip, [meter of water/s].*
- real(wp) [solid\\_precip](#)  
*Solid precip, [meter of water /s].*
- real(wp) [fl\\_q\\_snow](#)  
*flow of heat into the snow layer*
- real(wp) [energy\\_stored](#)

- Total amount of energy stored, control is freezing point temperature of S\_bu\_bottom [J].*

  - real(wp) [total\\_resist](#)  
*Thermal resistance of the whole column [].*
  - real(wp) [surface\\_water](#)  
*Percentage of water fraction in the top 5cm [%].*
  - real(wp) [freshwater](#)  
*Meters of freshwater stored in column [m].*
  - real(wp) [thickness](#)  
*Meters of ice [m].*
  - real(wp) [bulk\\_salinity](#)  
*Salt/Mass [ppt].*
  - real(wp) [thick\\_min](#)  
*Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.*
  - real(wp), save [t\\_test](#)  
*First guess for getT subroutine.*
  - real(wp) [albedo](#)  
*Amount of short wave radiation which is reflected at the top surface.*
  - real(wp) [fl\\_sw](#)  
*Incoming shortwave radiation [W/m\*\*2].*
  - real(wp) [fl\\_lw](#)  
*Incoming longwave radiation [W/m\*\*2].*
  - real(wp) [fl\\_sen](#)  
*Sensitive heat flux [W/m\*\*2].*
  - real(wp) [fl\\_lat](#)  
*Latent heat flux [W/m\*\*2].*
  - real(wp) [fl\\_rest](#)  
*Bundled longwave,sensitive and latent heat flux [W/m\*\*2].*
  - real(wp), dimension(:), allocatable [fl\\_rad](#)  
*Energy flux of absorbed sw radiation of each layer [J/s].*
  - real(wp) [grav\\_drain](#)  
*brine flux of gravity drainage between two outputs [kg/s]*
  - real(wp) [grav\\_salt](#)  
*salt flux moved by gravity drainage between two outputs [kg\*ppt/s]*
  - real(wp) [grav\\_temp](#)  
*average temperature of gravity drainage brine between two outputs [T]*
  - real(wp) [melt\\_thick](#)  
*thickness of fully liquid part of top layer [m]*
  - real(wp) [melt\\_thick\\_snow](#)
  - real(wp) [melt\\_thick\\_snow\\_old](#)  
*Niels(2017) add: thickness of excess fully liquid part from snow\_melt\_processes [m].*
  - real(wp), dimension(3) [melt\\_thick\\_output](#)  
*Niels, 2017 add: output field of surface liquid meltwater sizes.*
  - real(wp) [alpha\\_flux\\_instable](#)  
*Proportionality constant which determines energy flux by the temperature difference  $T_{top} > T_{2m}$  [W/C].*
  - real(wp) [alpha\\_flux\\_stable](#)  
*Proportionality constant which determines energy flux by the temperature difference  $T_{top} < T_{2m}$  [W/C].*
  - integer [atmoflux\\_flag](#)  
*1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in [mo\\_init](#)*
  - integer [grav\\_flag](#)  
*1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage*
  - integer [prescribe\\_flag](#)

- 1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)
- integer `grav_heat_flag`
  - 1: nothing happens, 2: compensates heatfluxes in `grav_flag = 2`
- integer `flush_heat_flag`
  - 1: nothing happens, 2: compensates heatfluxes in `flush_flag = 5`
- integer `turb_flag`
  - 1: No bottom turbulence, 2: Bottom mixing
- integer `salt_flag`
  - 1: Sea salt, 2: NaCL
- integer `boundflux_flag`
  - 1: top and bottom cooling plate, 2: top Notz fluxes, bottom cooling plate 3: top flux= $a \cdot (T - T_s)$
- integer `flush_flag`
  - 1: no flushing, 4: meltwater is removed artificially, 5: vert and horiz flushing, 6: simplified
- integer `flood_flag`
  - 1: no flooding, 2: normal flooding, 3: simple flooding
- integer `bottom_flag`
  - 1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests
- integer `debug_flag`
  - 1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)
- integer `precip_flag`
  - 0: solid and liquid precipitation, 1: phase determined by T2m
- integer `harmonic_flag`
  - 1: minimal permeability is used to calculate Rayleigh number, 2: harmonic mean is used for Rayleigh number
- integer `tank_flag`
  - 1: nothing, 2: `S_bu_bottom` and `bgc_bottom` are calculated as if the experiment is conducted in a tank
- integer `albedo_flag`
  - 1: simple albedo, 2: normal albedo, see `func_albedo` for details
- integer `lab_snow_flag`
  - Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.
- integer `freeboard_snow_flag`
  - Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.
- integer `snow_flush_flag`
  - Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "`k_snow_flush`".
- integer `snow_precip_flag`
  - Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.
- integer `length_input`
  - Sets the input length for `atmoflux_flag==2`, common value of 13169.
- real(wp), dimension(:), allocatable `tinput`
  - Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `precipinput`
  - Niels, 2017 add: used to read in precipitation for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `ocean_t_input`
  - Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `ocean_flux_input`
  - Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `styropor_input`
  - Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.
- real(wp), dimension(:), allocatable `ttop_input`

- Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as Ttop.*
- `real(wp), dimension(:), allocatable fl_sw_input`  
*Used to read in sw fluxes from ERA for atmoflux\_flag==2.*
  - `real(wp), dimension(:), allocatable fl_lw_input`  
*Used to read in lw fluxes from ERA for atmoflux\_flag==2.*
  - `real(wp), dimension(:), allocatable t2m_input`  
*Used to read in 2Tm from ERA for atmoflux\_flag==2.*
  - `real(wp), dimension(:), allocatable precip_input`  
*Used to read in precipitation from ERA for atmoflux\_flag==2.*
  - `real(wp), dimension(:), allocatable time_input`  
*Used to read in time from ERA for atmoflux\_flag==2.*
  - `integer time_counter`  
*Keeps track of input data.*
  - `integer bgc_flag`  
*1: no bgc, 2: bgc*
  - `integer n_bgc`  
*Number of chemicals.*
  - `real(wp), dimension(:, :), allocatable fl_brine_bgc`  
*Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.*
  - `real(wp), dimension(:, :), allocatable bgc_abs`  
*Absolute amount of chemicals [kmol] for each tracer.*
  - `real(wp), dimension(:, :), allocatable bgc_bu`  
*Bulk amounts of chemicals [kmol/kg].*
  - `real(wp), dimension(:, :), allocatable bgc_br`  
*Brine concentrations of chems [kmol/kg].*
  - `real(wp), dimension(:), allocatable bgc_bottom`  
*Bulk concentrations of chems below the ice [kmol/kg].*
  - `real(wp), dimension(:), allocatable bgc_total`  
*Total of chems, for lab experiments with a fixed total amount.*
  - `real(wp) m_total`  
*Total initial water mass, for lab experiments with a fixed total amount.*
  - `real(wp) s_total`  
*Total initial salt mass, for lab experiments with a fixed total amount.*
  - `real(wp) tank_depth`  
*water depth in meters, used to calculate concentrations below ice for tank experiments*
  - `character *3 flush_question = 'No!'`  
*Niels, 2017 add: used to indicate in stdout wether flushing occurs at this moment or not.*
  - `real(wp) melt_err = 0._wp`  
*Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.*
  - `integer length_input_lab`  
*Niels, 2017 add: used to allocate lab testcase input arrays in mo\_init, set value in testcases.*

#### 4.1.1 Detailed Description

Sets data and contains all flag descriptions.

All data needed by `mo_grotz` are set in this module. Most arrays are allocated after the needed dimension is specified for each testcase in `mo_init.f90`.

**Author**

Philipp Griewank

**COPYRIGHT**

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

**Revision History**

Initialized by Philipp Griewank, IMPRS (2010-07-14)  
Add several variables by Niels Fuchs, MPIMET (2017-03-01)

**4.1.2 Variable Documentation****4.1.2.1 real(wp) mo\_data::albedo**

Amount of short wave radiation which is reflected at the top surface.

**4.1.2.2 integer mo\_data::albedo\_flag**

1: simple albedo, 2: normal albedo, see func\_albedo for details

**4.1.2.3 real(wp) mo\_data::alpha\_flux\_instable**

Proportionality constant which determines energy flux by the temperature difference  $T_{\text{top}} > T_{\text{2m}}$  [W/C].

**4.1.2.4 real(wp) mo\_data::alpha\_flux\_stable**

Proportionality constant which determines energy flux by the temperature difference  $T_{\text{top}} < T_{\text{2m}}$  [W/C].

**4.1.2.5 integer mo\_data::atmoflux\_flag**

1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in [mo\\_init](#)

**4.1.2.6 real(wp), dimension(:,:), allocatable mo\_data::bgc\_abs**

Absolute amount of chemicals [kmol] for each tracer.

**4.1.2.7 real(wp), dimension(:), allocatable mo\_data::bgc\_bottom**

Bulk concentrations of chems below the ice [kmol/kg].

**4.1.2.8 real(wp), dimension(:, :), allocatable mo\_data::bgc\_br**

Brine concentrations of chems [kmol/kg].

**4.1.2.9 real(wp), dimension(:, :), allocatable mo\_data::bgc\_bu**

Bulk amounts of chemicals [kmol/kg].

**4.1.2.10 integer mo\_data::bgc\_flag**

1: no bgc, 2: bgc

**4.1.2.11 real(wp), dimension(:), allocatable mo\_data::bgc\_total**

Total of chems, for lab experiments with a fixed total amount.

**4.1.2.12 integer mo\_data::bottom\_flag**

1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests

**4.1.2.13 integer mo\_data::boundflux\_flag**

1: top and bottom cooling plate, 2: top Notz fluxes, bottom cooling plate 3: top flux= $a \cdot (T - T_s)$

**4.1.2.14 real(wp) mo\_data::bulk\_salin**

Salt/Mass [ppt].

**4.1.2.15 integer mo\_data::debug\_flag**

1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)

**4.1.2.16 real(wp) mo\_data::dt**

Timestep [s].

**4.1.2.17 real(wp) mo\_data::energy\_stored**

Total amount of energy stored, control is freezing point temperature of S\_bu\_bottom [J].

**4.1.2.18 real(wp), dimension(:, :), allocatable mo\_data::fl\_brine\_bgc**

Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.

**4.1.2.19 real(wp) mo\_data::fl\_lat**

Latent heat flux [W/m\*\*2].

**4.1.2.20 real(wp) mo\_data::fl\_lw**

Incoming longwave radiation [W/m\*\*2].

**4.1.2.21 real(wp), dimension(:), allocatable mo\_data::fl\_lw\_input**

Used to read in lw fluxes from ERA for atmoflux\_flag==2.

**4.1.2.22 real(wp), dimension(:), allocatable mo\_data::fl\_m**

Mass fluxes between layers [kg].

**4.1.2.23 real(wp), dimension(:), allocatable mo\_data::fl\_q**

Heat flux between layers [J/s].

**4.1.2.24 real(wp) mo\_data::fl\_q\_bottom**

Bottom heat flux [J\*s].

**4.1.2.25 real(wp) mo\_data::fl\_q\_snow**

flow of heat into the snow layer

**4.1.2.26 real(wp), dimension(:), allocatable mo\_data::fl\_rad**

Energy flux of absorbed sw radiation of each layer [J/s].

**4.1.2.27 real(wp) mo\_data::fl\_rest**

Bundled longwave, sensitive and latent heat flux [W/m\*\*2].

**4.1.2.28 real(wp), dimension(:), allocatable mo\_data::fl\_s**

Salinity flux [(g/s).

**4.1.2.29 real(wp) mo\_data::fl\_sen**

Sensitive heat flux [W/m\*\*2].

**4.1.2.30 real(wp) mo\_data::fl\_sw**

Incoming shortwave radiation [W/m\*\*2].

**4.1.2.31 real(wp), dimension(:), allocatable mo\_data::fl\_sw\_input**

Used to read in sw fluxes from ERA for atmoflux\_flag==2.

**4.1.2.32 integer mo\_data::flood\_flag**

1: no flooding, 2:normal flooding, 3:simple flooding

**4.1.2.33 integer mo\_data::flush\_flag**

1: no flushing, 4:meltwater is removed artificially, 5:vert and horiz flushing, 6: simplified

**4.1.2.34 real(wp), dimension(:), allocatable mo\_data::flush\_h****4.1.2.35 real(wp), dimension(:), allocatable mo\_data::flush\_h\_old**

Permeability [?].

**4.1.2.36 integer mo\_data::flush\_heat\_flag**

1: nothing happens, 2: compensates heatfluxes in flush\_flag = 5

**4.1.2.37 character\*3 mo\_data::flush\_question = 'No!'**

Niels, 2017 add: used to indicate in stdout whether flushing occurs at this moment or not.



4.1.2.38 `real(wp), dimension(:), allocatable mo_data::flush_v`

4.1.2.39 `real(wp), dimension(:), allocatable mo_data::flush_v_old`

4.1.2.40 `character*12000 mo_data::format_bgc`

4.1.2.41 `character*12000 mo_data::format_integer`

4.1.2.42 `character*12000 mo_data::format_melt`

Format strings for output. Niels(2017) add: melt output.

4.1.2.43 `character*12000 mo_data::format_perm`

Niels(2017) add: permeability output.

4.1.2.44 `character*12000 mo_data::format_psi`

4.1.2.45 `character*12000 mo_data::format_snow`

4.1.2.46 `character*12000 mo_data::format_t`

4.1.2.47 `character*12000 mo_data::format_t2m_top`

4.1.2.48 `character*12000 mo_data::format_thick`

4.1.2.49 `real(wp) mo_data::freeboard`

Height of ice surface above (or below) waterlevel [m].

4.1.2.50 `integer mo_data::freeboard_snow_flag`

Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.

4.1.2.51 `real(wp) mo_data::freshwater`

Meters of freshwater stored in column [m].

4.1.2.52 `real(wp) mo_data::grav_drain`

brine flux of gravity drainage between two outputs [kg/s]

**4.1.2.53 integer mo\_data::grav\_flag**

1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage

**4.1.2.54 integer mo\_data::grav\_heat\_flag**

1: nothing happens, 2: compensates heatfluxes in grav\_flag = 2

**4.1.2.55 real(wp) mo\_data::grav\_salt**

salt flux moved by gravity drainage between two outputs [kg\*ppt/s]

**4.1.2.56 real(wp) mo\_data::grav\_temp**

average temperature of gravity drainage brine between two outputs [T]

**4.1.2.57 real(wp), dimension(:), allocatable mo\_data::h**

Enthalpy [J].

**4.1.2.58 real(wp), dimension(:), allocatable mo\_data::h\_abs**

specific Enthalpy [J/kg]

**4.1.2.59 real(wp) mo\_data::h\_abs\_snow**

Absolute enthalpy of snow layer [J].

**4.1.2.60 integer mo\_data::harmonic\_flag**

1: minimal permeability is used to calculate Rayleigh number, 2:harmonic mean is used for Rayleigh number

**4.1.2.61 integer mo\_data::i**

Index, normally used for time.

**4.1.2.62 integer mo\_data::i\_time**

Number of timesteps.

**4.1.2.63 integer mo\_data::i\_time\_out**

Number of timesteps between each output.

**4.1.2.64 integer mo\_data::k**

Index, normally used for layer.

**4.1.2.65 integer mo\_data::lab\_snow\_flag**

Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.

**4.1.2.66 integer mo\_data::length\_input**

Sets the input length for `atmoflux_flag==2`, common value of 13169.

**4.1.2.67 integer mo\_data::length\_input\_lab**

Niels, 2017 add: used to allocate lab testcase input arrays in [mo\\_init](#), set value in testcases.

**4.1.2.68 real(wp) mo\_data::liquid\_precip**

Liquid precip, [meter of water/s].

**4.1.2.69 real(wp), dimension(:), allocatable mo\_data::m**

Mass [kg].

**4.1.2.70 real(wp) mo\_data::m\_snow**

Mass of snow layer [kg].

**4.1.2.71 real(wp) mo\_data::m\_total**

Total initial water mass, for lab experiments with a fixed total amount.

**4.1.2.72 real(wp) mo\_data::melt\_err = 0.\_wp**

Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.

#### 4.1.2.73 `real(wp) mo_data::melt_thick`

thickness of fully liquid part of top layer [m]

#### 4.1.2.74 `real(wp), dimension(3) mo_data::melt_thick_output`

Niels, 2017 add: output field of surface liquid meltwater sizes.

#### 4.1.2.75 `real(wp) mo_data::melt_thick_snow`

#### 4.1.2.76 `real(wp) mo_data::melt_thick_snow_old`

Niels(2017) add: thickness of excess fully liquid part from snow\_melt\_processes [m].

#### 4.1.2.77 `integer mo_data::n_active`

Number of Layers active in the present.

#### 4.1.2.78 `integer mo_data::n_bgc`

Number of chemicals.

#### 4.1.2.79 `integer mo_data::n_bottom`

Number of bottom layers.

#### 4.1.2.80 `integer mo_data::n_middle`

Number of middle layers.

#### 4.1.2.81 `integer mo_data::n_time_out`

Counts number of timesteps between output.

#### 4.1.2.82 `integer mo_data::n_top`

Number of top layers.

#### 4.1.2.83 `integer mo_data::nlayer`

Number of layers.

**4.1.2.84 real(wp), dimension(:), allocatable mo\_data::ocean\_flux\_input**

Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.

**4.1.2.85 real(wp), dimension(:), allocatable mo\_data::ocean\_t\_input**

Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.

**4.1.2.86 real(wp), dimension(:), allocatable mo\_data::perm****4.1.2.87 real(wp), dimension(:), allocatable mo\_data::phi**

Solid mass fraction.

**4.1.2.88 real(wp) mo\_data::phi\_s**

Solid mass fraction of snow layer.

**4.1.2.89 integer mo\_data::precip\_flag**

0: solid and liquid precipitation, 1: phase determined by T2m

**4.1.2.90 real(wp), dimension(:), allocatable mo\_data::precip\_input**

Used to read in precipitation from ERA for atmoflux\_flag==2.

**4.1.2.91 real(wp), dimension(:), allocatable mo\_data::precipinput**

Niels, 2017 add: used to read in precipitation for field experiment tests, dimension needs to be set in the code.

**4.1.2.92 integer mo\_data::prescribe\_flag**

1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)

**4.1.2.93 real(wp), dimension(:), allocatable mo\_data::psi\_g**

Gas volume fraction.

4.1.2.94 `real(wp) mo_data::psi_g_snow`

Gas volume fraction of snow layer.

4.1.2.95 `real(wp), dimension(:), allocatable mo_data::psi_l`

Liquid volume fraction.

4.1.2.96 `real(wp) mo_data::psi_l_snow`

Liquid volume fraction of snow layer.

4.1.2.97 `real(wp), dimension(:), allocatable mo_data::psi_s`

Solid volume fraction.

4.1.2.98 `real(wp) mo_data::psi_s_snow`

Solid volume fraction of snow layer.

4.1.2.99 `real(wp), dimension(:), allocatable mo_data::q`

Heat in layer [J].

4.1.2.100 `real(wp), dimension(:), allocatable mo_data::ray`

Rayleigh number of each layer.

4.1.2.101 `real(wp), dimension(:), allocatable mo_data::s_abs`

Absolute Salinity [g].

4.1.2.102 `real(wp) mo_data::s_abs_snow`

Absolute salinity of snow layer [g].

4.1.2.103 `real(wp), dimension(:), allocatable mo_data::s_br`

Brine salinity [g/kg].

4.1.2.104 `real(wp), dimension(:), allocatable mo_data::s_bu`

Bulk Salinity [g/kg].

4.1.2.105 `real(wp) mo_data::s_bu_bottom`

Salinity beneath the ice [g/kg].

4.1.2.106 `real(wp) mo_data::s_total`

Total initial salt mass, for lab experiments with a fixed total amount.

4.1.2.107 `integer mo_data::salt_flag`

1: Sea salt, 2: NaCL

4.1.2.108 `integer mo_data::snow_flush_flag`

Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "k\_snow\_flush".

4.1.2.109 `integer mo_data::snow_precip_flag`

Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.

4.1.2.110 `real(wp) mo_data::solid_precip`

Solid precip, [meter of water /s].

4.1.2.111 `integer mo_data::styropor_flag`

4.1.2.112 `real(wp), dimension(:), allocatable mo_data::styropor_input`

Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.

4.1.2.113 `real(wp) mo_data::surface_water`

Percentage of water fraction in the top 5cm [%].

4.1.2.114 `real(wp), dimension(:), allocatable mo_data::t`

Temperature [C].

4.1.2.115 `real(wp) mo_data::t2m`

Two meter Temperature [C].

4.1.2.116 `real(wp), dimension(:), allocatable mo_data::t2m_input`

Used to read in 2Tm from ERA for `atmoflux_flag==2`.

4.1.2.117 `real(wp) mo_data::t_bottom`

Temperature of water beneath the ice [C].

4.1.2.118 `real(wp) mo_data::t_freeze`

Freezing temperature [C].

4.1.2.119 `real(wp) mo_data::t_snow`

Temperature of snow layer [C].

4.1.2.120 `real(wp), save mo_data::t_test`

First guess for `getT` subroutine.

4.1.2.121 `real(wp) mo_data::t_top`

Temperature at the surface [C].

4.1.2.122 `real(wp) mo_data::tank_depth`

water depth in meters, used to calculate concentrations below ice for tank experiments

4.1.2.123 `integer mo_data::tank_flag`

1: nothing, 2: `S_bu_bottom` and `bgc_bottom` are calculated as if the experiment is conducted in a tank

4.1.2.124 `real(wp) mo_data::test`

Thickness of snow layer [m].



4.1.2.125 `real(wp), dimension(:), allocatable mo_data::thick`

Layer thickness [m].

4.1.2.126 `real(wp) mo_data::thick_0`

Initial layer thickness [m].

4.1.2.127 `real(wp) mo_data::thick_min`

Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.

4.1.2.128 `real(wp) mo_data::thick_snow`

4.1.2.129 `real(wp) mo_data::thickness`

Meters of ice [m].

4.1.2.130 `real(wp) mo_data::time`

Time [s].

4.1.2.131 `integer mo_data::time_counter`

Keeps track of input data.

4.1.2.132 `real(wp), dimension(:), allocatable mo_data::time_input`

Used to read in time from ERA for `atmoflux_flag==2`.

4.1.2.133 `real(wp) mo_data::time_out`

Time between outputs [s].

4.1.2.134 `real(wp) mo_data::time_total`

Time of simulation [s].

4.1.2.135 `real(wp), dimension(:), allocatable mo_data::tinput`

Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.

#### 4.1.2.136 `real(wp) mo_data::total_resist`

Thermal resistance of the whole column [].

#### 4.1.2.137 `real(wp), dimension(:), allocatable mo_data::ttop_input`

Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as Ttop.

#### 4.1.2.138 `integer mo_data::turb_flag`

1: No bottom turbulence, 2: Bottom mixing

#### 4.1.2.139 `real(wp), dimension(:), allocatable mo_data::v_ex`

Volume of brine due expelled due to freezing [m<sup>3</sup>] of solid, gas & liquid.

#### 4.1.2.140 `real(wp), dimension(:), allocatable mo_data::v_g`

Volume [m<sup>3</sup>] of gas.

#### 4.1.2.141 `real(wp), dimension(:), allocatable mo_data::v_l`

Volume [m<sup>3</sup>] of liquid.

#### 4.1.2.142 `real(wp), dimension(:), allocatable mo_data::v_s`

Volume [m<sup>3</sup>] of solid.

## 4.2 `mo_flood` Module Reference

Computes the fluxes caused by liquid flooding the snow layer.

### Functions/Subroutines

- subroutine, public `flood` (freeboard, psi\_s, psi\_l, S\_abs, H\_abs, m, T, thick, dt, Nlayer, N\_active, T\_bottom, S\_bu\_bottom, H\_abs\_snow, m\_snow, thick\_snow, psi\_g\_snow, debug\_flag, fl\_brine\_bgc)  
*Subroutine for calculating flooding.*
- subroutine, public `flood_simple` (freeboard, S\_abs, H\_abs, m, thick, T\_bottom, S\_bu\_bottom, H\_abs\_snow, m\_snow, thick\_snow, psi\_g\_snow, Nlayer, N\_active, debug\_flag)  
*Subroutine for calculating flooding.*

### 4.2.1 Detailed Description

Computes the fluxes caused by liquid flooding the snow layer.

Water floods the snow layer instantly transforming it to ice which is added to the top layer. As long as the negative freeboard is smaller than a certain parameter (*neg\_free*) the flood strength is limited by the harmonic mean permeability of the whole ice layer driven by the freeboard. When this parameter is exceeded, instant flooding is assumed. Based on Ted Maksyms work, brine is moved from the ocean to the snow without interacting with the ice in between. Very little of the process is well understood, so this parametrisation is ID mostly speculation. *Ratio\_flood* is a very important parameter, as it regulates how much wicking into the snow layer occurs during melting which dilutes the flooded snow. Ratio of two should lead to the snow pack being reduced twice as much as the top layer grows.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

#### Revision History

Copy and pasted into existence by Philipp Griewank, IMPRS (2011-01-21)

### 4.2.2 Function/Subroutine Documentation

**4.2.2.1** subroutine, public mo\_flood::flood ( real(wp), intent(in) *freeboard*, real(wp), dimension(nlayer), intent(in) *psi\_s*, real(wp), dimension(nlayer), intent(in) *psi\_l*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), dimension(nlayer), intent(inout) *thick*, real(wp), intent(in) *dt*, integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *m\_snow*, real(wp), intent(inout) *thick\_snow*, real(wp), intent(in) *psi\_g\_snow*, integer, intent(in) *debug\_flag*, real(wp), dimension(nlayer+1,nlayer+1), intent(inout), optional *fl\_brine\_bgc* )

Subroutine for calculating flooding.

Details explained in module description.

#### Revision History

Formed by Philipp Griewank, IMPRS (2011-01-21)  
Cleaned and commented by Philipp Griewank, (2014-04-19)

4.2.2.2 subroutine, public mo\_flood::flood\_simple ( real(wp), intent(in) *freeboard*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(inout) *thick*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *m\_snow*, real(wp), intent(inout) *thick\_snow*, real(wp), intent(in) *psi\_g\_snow*, integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, integer, intent(in) *debug\_flag* )

Subroutine for calculating flooding.

Simplified version of flood. Flooding occurs instantly to fill the negative freeboard until it reaches neg\_free with underlying ocean water.

#### Revision History

Formed by Philipp Griewank, IMPRS (2012-07-16) Added neg\_free limitation.

## 4.3 mo\_flush Module Reference

Contains various subroutines for flushing.

### Functions/Subroutines

- subroutine, public [flush3](#) (freeboard, psi\_l, thick, thick\_0, S\_abs, H\_abs, m, T, dt, Nlayer, N\_active, T\_bottom, S\_bu\_bottom, melt\_thick, debug\_flag, flush\_heat\_flag, melt\_err, perm, flush\_v, flush\_h, psi\_g, thick\_snow, rho\_l, snow\_flush\_flag, fl\_brine\_bgc)

*Subroutine for complex flushing.*

- subroutine, public [flush4](#) (psi\_l, thick, T, thick\_0, S\_abs, H\_abs, m, dt, Nlayer, N\_active, N\_top, N\_middle, N\_bottom, melt\_thick, debug\_flag)

*An alternative subroutine for calculating flushing.*

### 4.3.1 Detailed Description

Contains various subroutines for flushing.

Which subroutine is called is determined by flush\_flag.

#### Author

Philipp Griewank

### 4.3.2 Function/Subroutine Documentation

4.3.2.1 subroutine, public mo\_flush::flush3 ( real(wp), intent(in) *freeboard*, real(wp), dimension(nlayer), intent(inout) *psi\_l*, real(wp), dimension(nlayer), intent(inout) *thick*, real(wp), intent(in) *thick\_0*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), intent(in) *dt*, integer, intent(in) *Nlayer*, integer, intent(inout) *N\_active*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), intent(inout) *melt\_thick*, integer, intent(in) *debug\_flag*, integer, intent(in) *flush\_heat\_flag*, real(wp), intent(inout) *melt\_err*, real(wp), dimension(nlayer), intent(out) *perm*, real(wp), dimension(n\_active), intent(inout) *flush\_v*, real(wp), dimension(n\_active), intent(inout) *flush\_h*, real(wp), dimension(nlayer), intent(inout) *psi\_g*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *rho\_l*, integer, intent(in) *snow\_flush\_flag*, real(wp), dimension(nlayer+1,nlayer+1), intent(inout), optional *fl\_brine\_bgc* )

Subroutine for complex flushing.

Each layer splits the flushing brine into a fraction that moves downward, and a fraction that leaves the ice. A fraction of the top layer is considered melt water. This approach uses hydraulic resistivity  $R = \mu * \text{thick} / \text{perm}$ . The hydraulic head is assumed to be the freeboard. The vertical resistance  $R_v$  of each layer is a determined by its viscosity \* thickness divided by it's permeability. Additionally, each layer is given horizontal resistivity  $R_h$ . It is assumed that there is an average length horizontally which brine needs to flow to reach a drainage feature in the ice. We assume this length is a linear function of the ice thickness. The only tuning parameter is para\_flush\_horiz. The total resistance of layer i to the bottom is R.

For flush\_heat\_flag==2 the amount of heat which leaves by dynamics from the lowest layer is added to the lowest layer to keep results comparable to the other approaches. See PhD Griewank for details

#### Revision History

Invented by Philipp Griewank, IMPRS (2012-06-15)

Trying to add brine fluxes by Philipp Griewank, IMPRS (2014-02-01)

Changed: Permeability calculation (only for snow\_flush\_flag==1), hydraulic head and output data by Niels Fuchs, MPIMET (2017-03-01)

#### Parameters

in	<i>snow_flush_flag</i>	Niels, 2017 add: snow_flush_flag
in	<i>t</i>	Niels, 2017 add: moved psi_l -> INTENT(inout)
in, out	<i>psi_g</i>	Niels, 2017 add: psi_l, psi_g
in, out	<i>flush_v</i>	mass of vertically flushed brine of each layer [kg] !< Niels, 2017 add: inout
in, out	<i>flush_h</i>	mass of brine which leaves the ice of each layer [kg] !< Niels, 2017 add: inout
out	<i>perm</i>	Niels, 2017 add: out
in, out	<i>fl_brine_bgc</i>	Niels, 2017 add: if loop, enhanced the permeability, revise
in, out	<i>fl_brine_bgc</i>	Niels, 2017 add: psi_g to permeability calculation, improved the results but must be checked
in, out	<i>fl_brine_bgc</i>	Niels, 2017 add: melt thich is on top of the ice and therefore also part of the hydraulic head
in, out	<i>fl_brine_bgc</i>	Niels, 2017 add: melt_err, check how much meltwater vanishes in the line above

4.3.2.2 subroutine, public mo\_flush::flush4 ( real(wp), dimension(nlayer), intent(in) *psi\_l*, real(wp), dimension(nlayer), intent(inout) *thick*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), intent(in) *thick\_0*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), intent(in) *dt*, integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, integer, intent(in) *N\_top*, integer, intent(in) *N\_middle*, integer, intent(in) *N\_bottom*, real(wp), intent(inout) *melt\_thick*, integer, intent(in) *debug\_flag* )

An alternative subroutine for calculating flushing.

Simplified approach. Melt\_thick of top layer is simply removed with brine salinity. Salinity of a layer is reduced if the solid fraction is lower than that of the layer above it. Flushing stops as soon as a layer has a higher solid fraction than the layer below it.

#### Revision History

Invented by Philipp Griewank, IMPRS (2012-07-9)

## 4.4 mo\_functions Module Reference

Module houses functions which have no home :{.

### Functions/Subroutines

- real(wp) function [func\\_density](#) (T, S)  
*Calculates the physical density for given S and T.*
- real(wp) function [func\\_freeboard](#) (N\_active, Nlayer, psi\_s, psi\_g, m, thick, m\_snow, freeboard\_snow\_flag)  
*Calculates the freeboard of the 1d ice column.*
- real(wp) function [func\\_albedo](#) (thick\_snow, T\_snow, psi\_l, thick\_min, albedo\_flag)  
*Calculates the albedo.*
- real(wp) function [func\\_sat\\_o2](#) (T, S\_bu)  
*Calculates the oxygen saturation as a function of salinity and temperature.*
- real(wp) function [func\\_t\\_freeze](#) (S\_bu, salt\_flag)  
*Calculates the freezing temperature. Salt\_flag determines if either ocean salt or NaCl is used.*
- subroutine [sub\\_notzflux](#) (time, fl\_sw, fl\_rest)  
*Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.*
- subroutine [sub\\_input](#) (length\_input, fl\_sw\_input, fl\_lw\_input, T2m\_input, precip\_input, time\_input)  
*Reads in data for atmoflux\_flag ==2.*
- subroutine [sub\\_turb\\_flux](#) (T\_bottom, S\_bu\_bottom, T, S\_abs, m, dt, N\_bgc, bgc\_bottom, bgc\_abs)  
*Calculates salt and tracer mixing between lowest layer and underlying water.*
- subroutine [sub\\_melt\\_thick](#) (psi\_l, psi\_s, psi\_g, T, T\_freeze, T\_top, fl\_Q, thick\_snow, dt, melt\_thick, thick, thick\_min)  
*Calculates the thickness of the meltwater film.*
- subroutine [sub\\_melt\\_snow](#) (melt\_thick, thick, thick\_snow, H\_abs, H\_abs\_snow, m, m\_snow, psi\_g\_snow)  
*Calculates how the meltwater film interacts with snow.*

#### 4.4.1 Detailed Description

Module houses functions which have no home :{.

Created because I wanted to calculate the freeboard separately and didn't know where to put it.

#### Author

Philipp Griewank

### 4.4.2 Function/Subroutine Documentation

4.4.2.1 `real(wp) function mo_functions::func_albedo ( real(wp), intent(in) thick_snow, real(wp), intent(in) T_snow, real(wp), intent(in) psi_l, real(wp), intent(in) thick_min, integer, intent(in) albedo_flag )`

Calculates the albedo.

Calculates the albedo according to top conditions. This is not a good albedo scheme! It is only a quick approach. Non-continuous switching between wet and dry ice. Linear change from wet ice to water. Linear change from ice\_dry snow for snow thinner than 30cm.

$\text{psi\_l}(1) > 0.75$  water  $\text{psi\_l}(1) > 0.6$  linear change from wet ice to water  $\text{psi\_l}(1) > 0.2$  wet ice  $\text{psi\_l}(1) < 0.2 \rightarrow$  dry ice  $T\_snow = 0 \rightarrow$  wet snow  $T\_snow < 0 \rightarrow$  dry snow

#### Revision History

Built to spill by Philipp Griewank (2011-02-12)

4.4.2.2 `real(wp) function mo_functions::func_density ( real(wp), intent(in) T, real(wp), intent(in) S )`

Calculates the physical density for given S and T.

Although the model treats Salinity as a massless tracer, sometimes it is necessary to determine the exact density for specific purposes. First implemented to calculate simple turbulence between liquid layer and ocean. Uses following simplification of Frank J. Millero and Alain Poisson 1981:  $\text{Density} = \text{density\_0} + A*S + B*S**1.5$

#### Revision History

Started by Philipp Griewank (2011-02-24)

4.4.2.3 `real(wp) function mo_functions::func_freeboard ( integer, intent(in) N_active, integer, intent(in) Nlayer, real(wp), dimension(nlayer), intent(in) psi_s, real(wp), dimension(nlayer), intent(in) psi_g, real(wp), dimension(nlayer), intent(in) m, real(wp), dimension(nlayer), intent(in) thick, real(wp), intent(in) m_snow, integer, intent(in) freeboard_snow_flag )`

Calculates the freeboard of the 1d ice column.

The freeboard is calculated by first finding out which layer is at water level, and then finding out how deep the layer is submerged. For the correct freeboard the mass above water equals the buoyancy of the submerged part. Since the density of each layer is constant, step two can be calculated explicitly. The freeboard is the distance from the top of the ice to the water level. If snow pushes the ice underwater the freeboard becomes negative

#### Revision History

Built to spill by Philipp Griewank (2011-01-07)

Negative freeboard included by Philipp Griewank (2011-01-09)

Patched bug by Philipp Griewank (2011-03-10)

Add freeboard\_snow\_flag calculation of snow mass, check the code for further explanations by Niels Fuchs, MPIMET (2017-03-91)

#### 4.4.2.4 `real(wp) function mo_functions::func_sat_o2 ( real(wp), intent(in) T, real(wp), intent(in) S_bu )`

Calculates the oxygen saturation as a function of salinity and temperature.

Calculates the concentration of oxygen dissolved in freshwater and seawater in equilibrium with the atmosphere. The value should be  $\mu\text{mol/kg}$ . I switched to the solubility of nitrogen, oxygen and argon in water and sea water from Weiss R.F. 1970 because I couldn't get the other one to work out

##### Revision History

Written by Dr. Philipp Griewank (2014-02-25)

#### 4.4.2.5 `real(wp) function mo_functions::func_t_freeze ( real(wp), intent(in) S_bu, integer, intent(in) salt_flag )`

Calculates the freezing temperature. `Salt_flag` determines if either ocean salt or NaCl is used.

##### Revision History

Written to procrastinate by Philipp Griewank (2011-05-05)

#### 4.4.2.6 `subroutine mo_functions::sub_input ( integer, intent(in) length_input, real(wp), dimension(:), intent(out), allocatable fl_sw_input, real(wp), dimension(:), intent(out), allocatable fl_lw_input, real(wp), dimension(:), intent(out), allocatable T2m_input, real(wp), dimension(:), intent(out), allocatable precip_input, real(wp), dimension(:), intent(out), allocatable time_input )`

Reads in data for `atmoflux_flag == 2`.

Standard setup used for testcase 4 and all Griewank & Notz 2013/14 reanalysis forced runs is 4.5 years of three hourly values of shortwave incoming, longwave incoming, two meter T, and total precipitation. Data is read from ascii files and stored in long 1D arrays. ERA-interim derived input files in the standard length for various Arctic locations are located under `/input/ERA/`. Latent and sensible heat fluxes are not included, but could be added if needed.

##### Revision History

Moved here from [mo\\_grotz](#) by Philipp Griewank (2014-04-20)

#### 4.4.2.7 `subroutine mo_functions::sub_melt_snow ( real(wp), intent(inout) melt_thick, real(wp), intent(inout) thick, real(wp), intent(inout) thick_snow, real(wp), intent(inout) H_abs, real(wp), intent(inout) H_abs_snow, real(wp), intent(inout) m, real(wp), intent(inout) m_snow, real(wp), intent(inout) psi_g_snow )`

Calculates how the meltwater film interacts with snow.

Is activated when a thin snow layer (thinner than `thick_min`) is on top of meltwater. The snow is flooded and turned into ice.

##### Revision History

Put together by Philipp Griewank (2011-10-17)



4.4.2.8 subroutine mo\_functions::sub\_melt\_thick ( real(wp), intent(in) *psi\_l*, real(wp), intent(in) *psi\_s*, real(wp), intent(in) *psi\_g*, real(wp), intent(in) *T*, real(wp), intent(in) *T\_freeze*, real(wp), intent(in) *T\_top*, real(wp), intent(in) *fl\_Q*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *dt*, real(wp), intent(out) *melt\_thick*, real(wp), intent(inout) *thick*, real(wp), intent(in) *thick\_min* )

Calculates the thickness of the meltwater film.

If the top ice layer is being melted ( $T_{top} > T_{freeze}$ ) it is assumed that a thin meltwater film appears at the top. The thickness of this film is determined by the amount of incoming heat and diffusive transport. The incoming heat is an input ( $fl\_q(1)$ ) and the diffusive heat is  $(T(1) - T_{freeze})/R$ . See the thermodynamics section for  $R$ . The thickness of the meltlayer is determined by dividing the heat intake of the meltwater film by the amount of latent heat needed to melt the solid fraction of the top layer. If the solid fractions sinks below a given threshold ( $psi\_s\_top\_min$ ) a different approach is used. The melt thickness is then calculated by assuming that the ice below the meltwater film has a solid fraction of  $psi\_s\_top\_min$ . Although the thickness can be reduced, variations of mass, salinity and enthalpy are calculated in the flushing subroutine.

#### Revision History

Introduced by Philipp Griewank (2011-05-09)

4.4.2.9 subroutine mo\_functions::sub\_notzflux ( real(wp), intent(in) *time*, real(wp), intent(out) *fl\_sw*, real(wp), intent(out) *fl\_rest* )

Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.

Simplified version of the Untersteiner Fluxes. Returns only two fluxes as a function of time. Simplified Year, 12 months of 30 days. *fl\_sw* is set to zero for November till February Returns fluxes for day with day zero being 1. Jan. Depending on when the run starts the time should be modified when calling

#### Revision History

Ripped from Dirk by Philipp Griewank (2011-02-13)

4.4.2.10 subroutine mo\_functions::sub\_turb\_flux ( real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), intent(in) *T*, real(wp), intent(inout) *S\_abs*, real(wp), intent(in) *m*, real(wp), intent(in) *dt*, integer, intent(in) *N\_bgc*, real(wp), dimension(n\_bgc), intent(in), optional *bgc\_bottom*, real(wp), dimension(n\_bgc), intent(inout), optional *bgc\_abs* )

Calculates salt and tracer mixing between lowest layer and underlying water.

Very simple turbulence assumption which mixes the lowest layer with the underlying water. Based on assumption that there is a constant amount of turbulence  $A$ . This turbulence is amplified when the lowest layer is denser then the ocean mixed layer. And also dampened when the lowest layer is less dense then the mixed layer. Assumption;  $turb = A * \exp(B(\text{density\_layer} - \text{density\_ocean}))$   $A$  and  $B$  set in parameters.  $A = turb\_A$  ,  $B = turb\_B$

#### Revision History

Moved from grotz by Philipp Griewank (2014-04-2)

## 4.5 mo\_grav\_drain Module Reference

Computes the Salt fluxes caused by gravity drainage.

### Functions/Subroutines

- subroutine, public [fl\\_grav\\_drain](#) (S\_br, S\_bu, psi\_l, psi\_s, psi\_g, thick, S\_abs, H\_abs, T, m, dt, Nlayer, N\_active, ray, T\_bottom, S\_bu\_bottom, grav\_drain, grav\_temp, grav\_salt, grav\_heat\_flag, harmonic\_flag, fl\_brine\_bgc)

*Calculates fluxes caused by gravity drainage.*

- subroutine, public [fl\\_grav\\_drain\\_simple](#) (psi\_s, psi\_l, thick, S\_abs, S\_br, Nlayer, N\_active, ray, grav\_drain, harmonic\_flag)

*Calculates salinity to imitate the effects gravity drainage.*

### 4.5.1 Detailed Description

Computes the Salt fluxes caused by gravity drainage.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

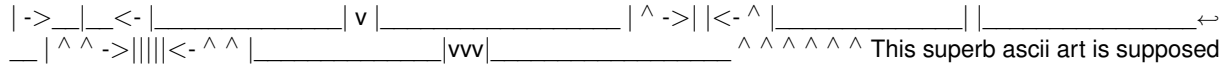
You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

### 4.5.2 Function/Subroutine Documentation

- 4.5.2.1 subroutine, public mo\_grav\_drain::fl\_grav\_drain ( real(wp), dimension(nlayer), intent(in) S\_br, real(wp), dimension(nlayer), intent(in) S\_bu, real(wp), dimension(nlayer), intent(in) psi\_l, real(wp), dimension(nlayer), intent(in) psi\_s, real(wp), dimension(nlayer), intent(in) psi\_g, real(wp), dimension(nlayer), intent(in) thick, real(wp), dimension(nlayer), intent(inout) S\_abs, real(wp), dimension(nlayer), intent(inout) H\_abs, real(wp), dimension(nlayer), intent(in) T, real(wp), dimension(nlayer), intent(inout) m, real(wp), intent(in) dt, integer, intent(in) Nlayer, integer, intent(in) N\_active, real(wp), dimension(nlayer-1), intent(out) ray, real(wp), intent(in) T\_bottom, real(wp), intent(in) S\_bu\_bottom, real(wp), intent(inout) grav\_drain, real(wp), intent(inout) grav\_temp, real(wp), intent(inout) grav\_salt, integer, intent(in) grav\_heat\_flag, integer, intent(in) harmonic\_flag, real(wp), dimension(nlayer+1,nlayer+1), intent(inout), optional fl\_brine\_bgc )

Calculates fluxes caused by gravity drainage.

If the Rayleigh number of a layer is higher then the critical value, brine leaves the layer by a theoretical brine channel. The discharged brine flows downward through all layers directly into the underlying ocean. To preserve mass the same amount of water flows upwards through all lower layers. In contrast to the downward flux the upward flux is assumed to be in thermal equilibrium thus moving salt and heat to each layer. The upward flux is a standard upwind advection. The downward flux of a layer over the timestep is  $= x \cdot (Ray - Ray\_crit) \cdot dt \cdot thick$ .

 This superb ascii art is supposed to show how the assumed fluxes flow downward through a brine channel and upwards through the layer.  $x$  and  $r$  are passed on to enable easy optimization. The effect of the upward moving brine is calculated in `mass_transfer`.

IMPORTANT: The height assumptions are special. The bottom of the ice edge is assumed to be at  $psi\_s(N\_active)/psi\_s\_min \cdot thick\_0$

The first approach assumed that brine drainage occurred between two layers but performed poorly.

If `grav_heat_flag` is set to 2 the amount of heat transported out of the ice will be compensated in the lowest layer

#### Revision History

created by Philipp Griewank, IMPRS (2010-08-27)  
 Completely revised to assume brine channels by Philipp Griewank, IMPRS (2010-11-05)  
`Mass_transfer` is used to advect  $H$  and  $S$  by Philipp Griewank, IMPRS (2010-11-05)  
 Added condition  $S\_br(k) > S\_br(k+1)$  by Philipp Griewank. IMPRS (2011-04-29)  
 Added harmonic mean for permeability by Philipp Griewank (2014-01-05)

#### Parameters

out	ray	Rayleigh number
-----	-----	-----------------

**4.5.2.2** subroutine, public `mo_grav_drain::fl_grav_drain_simple` ( `real(wp)`, `dimension(nlayer)`, `intent(in) psi_s`, `real(wp)`, `dimension(nlayer)`, `intent(in) psi_l`, `real(wp)`, `dimension(nlayer)`, `intent(in) thick`, `real(wp)`, `dimension(nlayer)`, `intent(inout) S_abs`, `real(wp)`, `dimension(nlayer)`, `intent(in) S_br`, `integer`, `intent(in) Nlayer`, `integer`, `intent(in) N_active`, `real(wp)`, `dimension(nlayer-1)`, `intent(out) ray`, `real(wp)`, `intent(inout) grav_drain`, `integer`, `intent(in) harmonic_flag` )

Calculates salinity to imitate the effects gravity drainage.

Based on the assumption that super critical Rayleigh numbers are quickly reduced below the critical Rayleigh number. Proposed as a very simplified parametrisation of gravity drainage. Includes no fluxes of any kind, instead bulk salinity is simply reduced when ever the Rayleigh number is above the critical values. The parametrization begins from the bottom layers and moves upward.

#### Revision History

created by Philipp Griewank, IMPRS (2012-01-01)

#### Parameters

out	ray	Rayleigh number
-----	-----	-----------------

## 4.6 mo\_grotz Module Reference

The most important module of SAMSIM.

### Functions/Subroutines

- subroutine [grotz](#) (testcase, description)

*Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by [mo\\_layer\\_dynamics](#).*

#### 4.6.1 Detailed Description

The most important module of SAMSIM.

The module [mo\\_grotz](#) contains the most important subroutine [grotz](#) (Named after GRiewank nOTZ). Mo\_grotz is called by [SAMSIM.f90](#). [SAMSIM.f90](#)'s only purpose is to set the testcase number and description string. Subroutine [grotz](#) contains the time loop, as well as the initialization, and calls all other branches of the model. This model was developed from scratch by Philipp Griewank during and after his PhD at Max Planck Institute of Meteorology from 2010-2014. The code is intended to be understandable and most subroutines, modules, functions, parameters, and global variables have doxygen compatible descriptions. In addition to the doxygen generated description, some python plotscripts are available to plot model output.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

## 4.6.2 Function/Subroutine Documentation

### 4.6.2.1 subroutine mo\_grotz::grotz ( integer, intent(in) *testcase*, character\*12000, intent(in) *description* )

Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by [mo\\_layer\\_dynamics](#).

The basic rundown of the time loop is:

1. Calculate the current ice/snow state and forcing, as well as gravity drainage and flooding
2. Apply all the fluxes, recalculate ice state
3. Flushing and layer dynamics

Here is the full rundown of what happens in [mo\\_grotz](#):

- Initialization: all fields are initialized for the given testcase, and the output is formatted
- Input and Forcing read in: Only if needed by the chosen testcase TIME LOOP BEGINS:
  - Calculate the total ice properties, total freshwater, thermal resistivity, energy, bulk salinity
  - Determine snow and rain rates
  - Calculate snow thermodynamics
  - Calculate inner ice thermodynamic fluxes
  - Calculate brine flux from expulsion
  - Raw output written out if debug\_flag is set to 2
  - Standard output written
  - Flooding parametrized
  - Lowest layer mixing with underlying water
  - Gravity drainage parametrized
  - Various testcase specifics
  - Calculating and applying the heat fluxes
  - After heatfluxes are applied new liquidus thermal equilibrium is calculated
  - Flushing is parametrized
  - Chemistry advection calculated
  - Layer Dynamics TIME LOOP ENDS -Final output, files closed, and fields deallocated

**IMPORTANT:** To get the correct freshwater amount make sure the freshwater is calculated using a salinity value to compare against.

Common errors leading to termination are: too small timestep, bad programming

#### Revision History

Basic thermodynamics and layer\_dynamics for fixed boundaries seem stable, backup made. by griewank (2010-08-10)  
 Add some more outputs, changed routine names and arguments with respect to newly introduces flags by Niels Fuchs, MPIMET (2017-03-01)  
 Added a bit of description with the run down of what happens by Philipp Griewank, Uni K (2018-08-08)

#### Parameters

in	description	String to describes simulation which is output into dat_settings
----	-------------	--

## 4.7 mo\_heat\_fluxes Module Reference

Computes all heat fluxes.

### Functions/Subroutines

- subroutine [sub\\_heat\\_fluxes](#) ()  
*Computes surface temperature and heatfluxes.*

#### 4.7.1 Detailed Description

Computes all heat fluxes.

Everything related to heat fluxes happens in `sub_heat_fluxes`, which is why it is a very crucial part of SAMSIM.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

## 4.7.2 Function/Subroutine Documentation

### 4.7.2.1 subroutine mo\_heat\_fluxes::sub\_heat\_fluxes ( )

Computes surface temperature and heatfluxes.

Major subroutine, calculates all atmospheric energy fluxes and applies both atmospheric and oceanic fluxes. Is one of the only subroutines to directly use [mo\\_data](#) because so many variables are needed.

There are three different ways to calculate atmospheric heat fluxes implemented which are defined using `boundflux_flag`.

- `Boundflux_flag`: 1 imitates top cooling plate by setting a fixed surface temperature, heat flux is derived from the T gradient from the surface to the top layer
- `Boundflux_flag`: 2 balances incoming and outgoing radiation to determine the surface temperature, heat flux is then calculated as in `boundflux_flag` 1. Some of the ice penetrates into the ice as is absorbed according to Beer's law. Optical properties are defined by the parameters `emissivity_ice`, `emissivity_snow`, `extinct`, and `penetr`.
- `Boundflux_flag`: 3 assumes the atmospheric heat flux is proportional to the difference between the top layer temperature and the air temperature.

For 1 and 2 the surface temperature in turn determines the atmospheric heat flux into the snow or ice. `Atmoflux_flag` is important for `boundflux_flag` 2, as it determines which atmospheric fluxes are used.

- `Atmoflux_flag`: 1 Mean climatology fluxes of Notz are used (see `sub_notz`)
- `Atmoflux_flag`: 2 Imported values are used, see `sub_input` for more info on reading in data.
- `Atmoflux_flag`: 3 Prescribed values are used (e.g. testcase 5).

Melting occurs when the surface T is above the melting temperature of the top layer

- `Boundflux_flag`: 1 atmospheric flux is limited by the parameter `max_flux_plate` which represents the maximum heating capacity of the plate
- `Boundflux_flag`: 2 the atmospheric heat flux is given by the difference between incoming and outgoing radiation
- `Boundflux_flag`: 3 works the same during melt and freezing, but a different proportionality parameter is used (`alpha_flux_stable`) because the air above the ice is assumed to be stably stratified.

`Boundflux_flag` 1 and 3 are not made to work with snow. If you need snow you'll have to implement snow cover yourself. For a detailed look at what is happening see the source code.

The snow layer is treated differently based on the snow thickness.

- If the snow layer is thinner than `thick_min/100` it is simply ignored.
- If the snow layer is thinner than `thick_min` but thicker than `thick_min/100` the snow and top ice layer are assumed to have the same temperature and are coupled using `snow_coupling`.
- If the snow layer is thicker than `thick_min` it is treated totally separately.

### Revision History

First version by Philipp Griewank (2014-04-02)  
Second version by Niels Fuchs (2017-02-02)

## 4.8 mo\_init Module Reference

Allocates Arrays and sets initial data for a given testcase for SAMSIM.

### Functions/Subroutines

- subroutine `init` (testcase)  
*Sets initial conditions according to which testcase is chosen.*
- subroutine `sub_allocate` (Nlayer, length\_input\_lab)  
*Allocates Arrays.*
- subroutine `sub_allocate_bgc` (Nlayer, N\_bgc)  
*Allocates BGC Arrays.*
- subroutine `sub_deallocate`  
*Deallocates Arrays.*

### 4.8.1 Detailed Description

Allocates Arrays and sets initial data for a given testcase for SAMSIM.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

### 4.8.2 Function/Subroutine Documentation

#### 4.8.2.1 subroutine mo\_init::init ( integer, intent(in) testcase )

Sets initial conditions according to which testcase is chosen.

For different initial conditions the Arrays are allocated and the initial values are set. Following must always be:

1. Nlayer = N\_top+N\_middle+N\_bottom
2. N\_active is set correctly, N\_active <= Nlayer
3. fl\_q\_bottom >= 0



4.  $T_{\text{bottom}} > \text{freezing point of ice}$  for  $S_{\text{bu\_bottom}}$
5. A too high  $dt$  for a too small  $\text{thick}_0$  leads to numerical thermodynamic instability. For a conservative guess  $dt$  [s] should be smaller than  $250000 * (dz \text{ [m]})^{**2}$

#### Testcase 1

- Testcase 1 is a replication of lab experiments conducted in tanks cooled from above by a cooling plate using the `boundflux_flag 1`.
- In this testcase the cooling plate Temperature  $T_{\text{top}}$  changes every 12 hours to imitate the experiments Dirk Notz conducted in his PhD.
- This testcase was used to optimize the free parameters of the gravity drainage parametrization (see Griewank Notz 2013/14).
- Can also be run with bgc tracers.

#### Testcase 2

- Testcase is an example of how to simulate ice growth and melt in cooling chambers.
- `Boundflux_flag 3` is used, which uses  $T_{2m}$  as the air temperature in the cooling chamber.
- The surface flux heat flux is proportional to the ice-air temperature difference ( $T_{\text{top}} - T_{2m}$ ).
- When reproducing cooling chamber experiments the alpha flux parameters need to be tuned, and a module in [mo\\_testcase\\_specifics](#) is needed to set  $T_{2m}$  over time.
- The heat flux in the water from below ( $fl\_q\_bottom$ ) for such experiments can be very hard to reproduce if the heat input is not carefully measured from all pumps or similar devices used.

#### Testcase 3

- Uses interpolated climate mean forcing from Notz and a constant oceanic heat flux ( $fl\_q\_bottom$ ) to grow idealized arctic sea ice.
- Is generally intended as a numerically cheap testcase to check for effects of code changes.
- Is also useful when runs over many years are needed.
- The amount of liquid and solid precipitation is set in `sub_test3` of `mo_testcase_specifics`.

#### Testcase 4

- Uses three hourly reanalysis forcing over 4.5 years.
- Is set up to start in July.
- Prescribes annual cycle of oceanic heat flux.
- Requires the proper input data to be copied into the executable folder (see `sub_input`).
- Is more computer intensive
- Was used a lot for Griewank & Notz 2013/2014

#### Revision History

First set up by Philipp Griewank, IMPRS (2010-07-22>)

#### 4.8.2.2 subroutine `mo_init::sub_allocate ( integer, intent(in) Nlayer, integer, intent(in), optional length_input_lab )`

Allocates Arrays.

For a given number of layers  $N_{\text{layers}}$  all arrays are allocated

## Parameters

in	<i>nlayer</i>	number of layers
in	<i>length_input_lab</i>	Niels, 2017 add: dimension of input arrays
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017
in	<i>length_input_lab</i>	Niels, 2017

4.8.2.3 subroutine mo\_init::sub\_allocate\_bgc ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_bgc* )

Allocates BGC Arrays.

## 4.8.2.4 subroutine mo\_init::sub\_deallocate ( )

Deallocates Arrays.

## 4.9 mo\_layer\_dynamics Module Reference

Mo\_layer\_dynamics contains all subroutines for the growth and shrinking of layer thickness.

### Functions/Subroutines

- subroutine, public [layer\\_dynamics](#) (phi, N\_active, Nlayer, N\_bottom, N\_middle, N\_top, m, S\_abs, H\_abs, thick, thick\_0, T\_bottom, S\_bu\_bottom, bottom\_flag, debug\_flag, melt\_thick\_output, N\_bgc, bgc\_abs, bgc\_bottom)

*Organizes the Semi-Adaptive grid SAMSIM uses.*

- subroutine, public [top\\_melt](#) (Nlayer, N\_active, N\_bottom, N\_middle, N\_top, thick\_0, m, S\_abs, H\_abs, thick, N\_bgc, bgc\_abs)
- subroutine, public [top\\_grow](#) (Nlayer, N\_active, N\_bottom, N\_middle, N\_top, thick\_0, m, S\_abs, H\_abs, thick, N\_bgc, bgc\_abs)

*Top grow subroutine.*

### 4.9.1 Detailed Description

Mo\_layer\_dynamics contains all subroutines for the growth and shrinking of layer thickness.

The middle layers have flexible thickness in contrast to the lower and upper layers which have static thickness. The details are provided in the separate subroutines.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

### 4.9.2 Function/Subroutine Documentation

**4.9.2.1** subroutine, public mo\_layer\_dynamics::layer\_dynamics ( real(wp), dimension(nlayer), intent(in) *phi*, integer, intent(inout) *N\_active*, integer, intent(in) *Nlayer*, integer, intent(in) *N\_bottom*, integer, intent(in) *N\_middle*, integer, intent(in) *N\_top*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *thick*, real(wp), intent(in) *thick\_0*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, integer, intent(in) *bottom\_flag*, integer, intent(in) *debug\_flag*, real(wp), intent(inout) *melt\_thick\_output*, integer, intent(in) *N\_bgc*, real(wp), dimension(nlayer,n\_bgc), intent(inout), optional *bgc\_abs*, real(wp), dimension(n\_bgc), intent(in), optional *bgc\_bottom* )

Organizes the Semi-Adaptive grid SAMSIM uses.

Modifies the grid and all core variables due to growth or melt. Calls the different subroutines according to current conditions. All subroutines can be called with or without biogeochemical tracers active, which is triggered by providing *bgc\_abs* when calling the subroutine. See Griewank PhD thesis for a full description of the grid.

Conditions under which following layer dynamics subroutines are called:

- *bottom\_melt*: lowest layer is ice free, second lowest layer has a solid fraction smaller than  $\phi_{s\_min}/2$ , and all *Nlayer* layers are active.
- *bottom\_melt\_simple*: lowest layer is ice free, second lowest layer has a solid fraction smaller than  $\phi_{s\_min}/2$ , and not all *Nlayer* layers are active.
- *bottom\_melt\_simple*: lowest layer is ice free, second lowest layer has a solid fraction smaller than  $\phi_{s\_min}/2$ , all *Nlayer* layers are active, and the thickness of the middle layers equals *thick\_0*
- *bottom\_growth\_simple*: lowest layer has a solid fraction higher than  $\psi_{s\_min}$ , and not all *Nlayer* layers are active
- *bottom\_growth*: lowest layer has a solid fraction higher than  $\psi_{s\_min}$ , and all *Nlayer* layers are active

- `top_grow`: top layer thicker than  $3/2 * \text{thick}_0$
- `top_melt`: top layer thinner than  $1/2 * \text{thick}_0$

If `debug_flag` is set to 2 the layer values will be written into the debug output (`thermoXX.dat`) before and after layer dynamics with a string to identify which subroutine was called

#### Revision History

created by Philipp Griewank, IMPRS (2010-07-29)

first complete and hopefully stable version by Philipp Griewank, IMPRS (2010-08-10)

#### Parameters

<code>in, out</code>	<code>melt_thick_output</code>	Niels, 2017 add: <code>melt_thick_output</code> !OBS: only 3rd element in standard <code>melt_thick_output</code> vector!
<code>in</code>	<code>bgc_bottom</code>	Niels, 2017 add: subtract top growth from melt thick output

**4.9.2.2** subroutine, public `mo_layer_dynamics::top_grow` ( integer, intent(in) *Nlayer*, integer, intent(inout) *N\_active*, integer, intent(in) *N\_bottom*, integer, intent(in) *N\_middle*, integer, intent(in) *N\_top*, real(wp), intent(in) *thick\_0*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *thick*, integer, intent(in) *N\_bgc*, real(wp), dimension(nlayer,n\_bgc), intent(inout), optional *bgc\_abs* )

Top grow subroutine.

Should be called when the top layer is thicker then  $1.5 * \text{thick}_0$ . If `N_active=Nlayer` middle layers are expanded by `thick_0/N_middle` and top layers are moved one down. IF `N_active<Nlayer` then `N_active=N_active+1` and all layers are shifted downwards.

#### Revision History

Started by Philipp Griewank, IMPRS (2011-05-10>)

**4.9.2.3** subroutine, public `mo_layer_dynamics::top_melt` ( integer, intent(in) *Nlayer*, integer, intent(inout) *N\_active*, integer, intent(in) *N\_bottom*, integer, intent(in) *N\_middle*, integer, intent(in) *N\_top*, real(wp), intent(in) *thick\_0*, real(wp), dimension(nlayer), intent(inout) *m*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *thick*, integer, intent(in) *N\_bgc*, real(wp), dimension(nlayer,n\_bgc), intent(inout), optional *bgc\_abs* )

## 4.10 mo\_mass Module Reference

Regulates mass transfers and their results.

#### Functions/Subroutines

- subroutine, public [mass\\_transfer](#) (*Nlayer*, *N\_active*, *T*, *H\_abs*, *S\_abs*, *S\_bu*, *T\_bottom*, *S\_bu\_bottom*, *fl\_m*)  
*Calculates the effects of mass transfers on H\_abs and S\_abs.*
- subroutine, public [expulsion\\_flux](#) (*thick*, *V\_ex*, *Nlayer*, *N\_active*, *psi\_g*, *fl\_m*, *m*)  
*Generates the fluxes caused by expulsion.*
- subroutine, public [bgc\\_advection](#) (*Nlayer*, *N\_active*, *N\_bgc*, *fl\_brine\_bgc*, *bgc\_abs*, *psi\_l*, *T*, *S\_abs*, *m*, *thick*, *bgc\_bottom*)  
*Calculates how the brine fluxes stored in fl\_brine\_bgc advect bgc tracers.*

### 4.10.1 Detailed Description

Regulates mass transfers and their results.

Ultimately all processes which involve a mass flux should be stored here.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

### 4.10.2 Function/Subroutine Documentation

**4.10.2.1** `subroutine, public mo_mass::bgc_advection ( integer, intent(in) Nlayer, integer, intent(in) N_active, integer, intent(in) N_bgc, real(wp), dimension(nlayer+1,nlayer+1), intent(in) fl_brine_bgc, real(wp), dimension(nlayer,n_bgc), intent(inout) bgc_abs, real(wp), dimension(nlayer), intent(in) psi_l, real(wp), dimension(nlayer), intent(in) T, real(wp), dimension(nlayer), intent(in) S_abs, real(wp), dimension(nlayer), intent(in) m, real(wp), dimension(nlayer), intent(in) thick, real(wp), dimension(n_bgc), intent(in) bgc_bottom )`

Calculates how the brine fluxes stored in `fl_brine_bgc` advect `bgc` tracers.

A very simple upwind strategy is employed. To avoid negative tracer densities, the maximum amount of advection is restricted to the current tracer content in a layer divided by three. Three is chosen as a limit as currently each layer can have a maximum of three flows leaving the layer (to the layer above, the layer below, and the lowest layer). The advection scheme is likely overly diffusive, but given the limitations we are working with (e.g. changing brine volumes) nothing more sophisticated can be applied easily.

For gases it might make sense to limit the brine density to saturation value in advecting brine, to take bubble formation into account. This needs to be specified in `bgc_advection`, and is a first attempt (both scientifically and code wise) which should be used with caution!

#### Revision History

Brought to life by Philipp Griewank, IMPRS (2014-02-10)

4.10.2.2 subroutine, public mo\_mass::expulsion\_flux ( real(wp), dimension(nlayer), intent(in) *thick*, real(wp), dimension(nlayer), intent(in) *V\_ex*, integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), dimension(nlayer), intent(inout) *psi\_g*, real(wp), dimension(nlayer+1), intent(out) *fl\_m*, real(wp), dimension(nlayer), intent(inout) *m* )

Generates the fluxes caused by expulsion.

Brine displaced by expansion of a freezing mushy layer lead to a mass, enthalpy and salt flux. This subroutine calculates the amount of brine which moves between the layers caused by *V\_ex* and how the mass in the layers changes. Vary basic assumptions are made. Brine always moves downward (negative), no horizontal movement are allowed and gas pockets can be filled. The upper boundary layer is not permeable but the bottom one is. This subroutine was started as a quick and dirty way to simulate the bottom freezing experiment described in Notz 2005 p. 85

#### Revision History

Brought to life by Philipp Griewank, IMPRS (2010-08-24)  
Simplified by Philipp Griewank, IMPRS (2010-11-27)

4.10.2.3 subroutine, public mo\_mass::mass\_transfer ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), dimension(nlayer), intent(inout) *H\_abs*, real(wp), dimension(nlayer), intent(inout) *S\_abs*, real(wp), dimension(nlayer), intent(in) *S\_bu*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), dimension(nlayer+1), intent(in) *fl\_m* )

Calculates the effects of mass transfers on *H\_abs* and *S\_abs*.

The effects of brine displaced by expulsion, flushing or drainage expansion lead to changes in mass, salt and enthalpy. This subroutine calculates the effects on *S\_abs* and *H\_abs*. A very simple upwind strategy is employed, Brine from below has *T* and *S\_br* of the lower layer, and brine from above *T* and *S\_br* of the upper layer. To avoid negative salinity, the maximum amount of advective salt is the total salt content of the layer. The amount of mass transferred is calculated in other subroutines.

This subroutine was started as a quick and dirty way to simulate the bottom freezing experiment described in Notz 2005 p. 85 IMPORTANT: Before this subroutine expelled brine was removed from the system and its effects were determined in subroutine expulsion. *S\_bu* must be up to date!

#### Revision History

Brought to life by Philipp Griewank, IMPRS (2010-08-24)  
Modified to work with all processes by Philipp Griewank, IMPRS (2010-11-27)

## 4.11 mo\_output Module Reference

All things output.

## Functions/Subroutines

- subroutine, public [output\\_settings](#) (description, testcase, N\_top, N\_bottom, Nlayer, fl\_q\_bottom, T\_bottom, S\_bu\_bottom, thick\_0, time\_out, time\_total, dt, boundflux\_flag, atmoflux\_flag, albedo\_flag, grav\_flag, flush\_flag, flood\_flag, grav\_heat\_flag, flush\_heat\_flag, harmonic\_flag, prescribe\_flag, salt\_flag, turb\_flag, bottom\_flag, tank\_flag, precip\_flag, bgc\_flag, N\_bgc, k\_snow\_flush)  
*Settings output.*
- subroutine, public [output](#) (Nlayer, T, psi\_s, psi\_l, thick, S\_bu, ray, format\_T, format\_psi, format\_thick, format\_snow, freeboard, thick\_snow, T\_snow, psi\_l\_snow, psi\_s\_snow, energy\_stored, freshwater, total\_resist, thickness, bulk\_salin, grav\_drain, grav\_salt, grav\_temp, T2m, T\_top, perm, format\_perm, flush\_v, flush\_h, psi\_g, melt\_thick\_output, format\_melt)  
*Standard output.*
- subroutine, public [output\\_bgc](#) (Nlayer, N\_active, bgc\_bottom, N\_bgc, bgc\_abs, psi\_l, thick, m, format\_bgc)  
*Standard bgc output.*
- subroutine, public [output\\_raw](#) (Nlayer, N\_active, time, T, thick, S\_bu, psi\_s, psi\_l, psi\_g)  
*Output for debugging purposes.*
- subroutine, public [output\\_raw\\_snow](#) (time, T\_snow, thick\_snow, S\_abs\_snow, m\_snow, psi\_s\_snow, psi\_l\_snow, psi\_g\_snow)  
*Output for debugging purposes.*
- subroutine, public [output\\_raw\\_lay](#) (Nlayer, N\_active, H\_abs, m, S\_abs, thick, string)  
*Output for debugging layer dynamics..*
- subroutine, public [output\\_begin](#) (Nlayer, debug\_flag, format\_T, format\_psi, format\_thick, format\_snow, format\_T2m\_top, format\_perm, format\_melt)  
*Output files are opened and format strings are created.*
- subroutine, public [output\\_begin\\_bgc](#) (Nlayer, N\_bgc, format\_bgc)  
*Output files for bgc are opened and format strings are created.*

### 4.11.1 Detailed Description

All things output.

Used to clean up root.f90 and make it easier to implement changes to the output.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

## 4.11.2 Function/Subroutine Documentation

4.11.2.1 subroutine, public mo\_output::output ( integer, intent(in) *Nlayer*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), dimension(nlayer), intent(in) *psi\_s*, real(wp), dimension(nlayer), intent(in) *psi\_l*, real(wp), dimension(nlayer), intent(in) *thick*, real(wp), dimension(nlayer), intent(in) *S\_bu*, real(wp), dimension(nlayer-1), intent(in) *ray*, character\*12000, intent(in) *format\_T*, character\*12000, intent(in) *format\_psi*, character\*12000, intent(in) *format\_thick*, character\*12000, intent(in) *format\_snow*, real(wp), intent(in) *freeboard*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *T\_snow*, real(wp), intent(in) *psi\_l\_snow*, real(wp), intent(in) *psi\_s\_snow*, real(wp), intent(in) *energy\_stored*, real(wp), intent(in) *freshwater*, real(wp), intent(in) *total\_resist*, real(wp), intent(in) *thickness*, real(wp), intent(in) *bulk\_salin*, real(wp), intent(in) *grav\_drain*, real(wp), intent(in) *grav\_salt*, real(wp), intent(in) *grav\_temp*, real(wp), intent(in) *T2m*, real(wp), intent(in) *T\_top*, real(wp), dimension(nlayer), intent(in) *perm*, character\*12000, intent(in) *format\_perm*, real(wp), dimension(nlayer), intent(in) *flush\_v*, real(wp), dimension(nlayer), intent(in) *flush\_h*, real(wp), dimension(nlayer), intent(in) *psi\_g*, real(wp), dimension(3), intent(in) *melt\_thick\_output*, character\*12000, intent(in) *format\_melt* )

Standard output.

For time=n\*time\_out data is exported.

### Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

### Parameters

in	<i>melt_thick_output</i>	Niels, 2017: 1: accumulated melt_thick, 2: accumulated melt_thick_snow, 3: accumulated top ice thickness variations (recheck 3: in <a href="#">mo_layer_dynamics</a> )
in	<i>format_melt</i>	Niels, 2017 add: output permeability
in	<i>format_melt</i>	Niels, 2017 add: output vertical flushing
in	<i>format_melt</i>	Niels, 2017 add: output horizontal flushing
in	<i>format_melt</i>	Niels, 2017 add: output gas fraction !OBS: not simulated physically in SAMSIM
in	<i>format_melt</i>	Niels, 2017

4.11.2.2 subroutine, public mo\_output::output\_begin ( integer, intent(in) *Nlayer*, integer, intent(in) *debug\_flag*, character\*12000, intent(out) *format\_T*, character\*12000, intent(out) *format\_psi*, character\*12000, intent(out) *format\_thick*, character\*12000, intent(out) *format\_snow*, character\*12000, intent(out) *format\_T2m\_top*, character\*12000, intent(out) *format\_perm*, character\*12000, intent(out) *format\_melt* )

Output files are opened and format strings are created.

Format strings are defined according to the number of layers used which define the output format. Files are opened.

### Revision History

created by Philipp Griewank, IMPRS (2010-10-11) moved by Philipp Griewank, IMPRS (2011-03-09)

4.11.2.3 subroutine, public mo\_output::output\_begin\_bgc ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_bgc*, character\*12000, intent(out) *format\_bgc* )

Output files for bgc are opened and format strings are created.

Same thing as out\_begin but for bgc Each tracer is outputted in bulk and in brine concentration in a separate file. Added ADJUSTL to the output strings because they got wierd



## Revision History

created by Dr. Philipp Griewank, MPI (2014-02-07) fix by Dr. Philipp Griewank, UniK (2018-05-18)

4.11.2.4 subroutine, public mo\_output::output\_bgc ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), dimension(n\_bgc), intent(in) *bgc\_bottom*, integer, intent(in) *N\_bgc*, real(wp), dimension(nlayer,n\_bgc), intent(in) *bgc\_abs*, real(wp), dimension(nlayer), intent(in) *psi\_l*, real(wp), dimension(nlayer), intent(in) *thick*, real(wp), dimension(nlayer), intent(in) *m*, character\*12000, intent(in) *format\_bgc* )

Standard bgc output.

For time=n\*time\_out data is exported.

## Revision History

created by Philipp Griewank, IMPRS (2014-02-06)

4.11.2.5 subroutine, public mo\_output::output\_raw ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), intent(in) *time*, real(wp), dimension(nlayer), intent(in) *T*, real(wp), dimension(nlayer), intent(in) *thick*, real(wp), dimension(nlayer), intent(in) *S\_bu*, real(wp), dimension(nlayer), intent(in) *psi\_s*, real(wp), dimension(nlayer), intent(in) *psi\_l*, real(wp), dimension(nlayer), intent(in) *psi\_g* )

Output for debugging purposes.

Data for each layer is written out each time step to aid in finding errors or understanding model behavior.

## Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

4.11.2.6 subroutine, public mo\_output::output\_raw\_layer ( integer, intent(in) *Nlayer*, integer, intent(in) *N\_active*, real(wp), dimension(nlayer), intent(in) *H\_abs*, real(wp), dimension(nlayer), intent(in) *m*, real(wp), dimension(nlayer), intent(in) *S\_abs*, real(wp), dimension(nlayer), intent(in) *thick*, character\*6, intent(in) *string* )

Output for debugging layer dynamics..

Is used when debug\_flag = 2 to track when which layer dynamics occur (see [mo\\_layer\\_dynamics](#)).

4.11.2.7 subroutine, public mo\_output::output\_raw\_snow ( real(wp), intent(in) *time*, real(wp), intent(in) *T\_snow*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *S\_abs\_snow*, real(wp), intent(in) *m\_snow*, real(wp), intent(in) *psi\_s\_snow*, real(wp), intent(in) *psi\_l\_snow*, real(wp), intent(in) *psi\_g\_snow* )

Output for debugging purposes.

Data of snow layer is written out at each time step to aid in finding errors or understanding model behavior.

## Revision History

created by Philipp Griewank, IMPRS (2010-10-11)

4.11.2.8 subroutine, public mo\_output::output\_settings ( character\*12000, intent(in) *description*, integer, intent(in) *testcase*, integer, intent(in) *N\_top*, integer, intent(in) *N\_bottom*, integer, intent(in) *Nlayer*, real(wp), intent(in) *fl\_q\_bottom*, real(wp), intent(in) *T\_bottom*, real(wp), intent(in) *S\_bu\_bottom*, real(wp), intent(in) *thick\_0*, real(wp), intent(in) *time\_out*, real(wp), intent(in) *time\_total*, real(wp), intent(in) *dt*, integer, intent(in) *boundflux\_flag*, integer, intent(in) *atmoflux\_flag*, integer, intent(in) *albedo\_flag*, integer, intent(in) *grav\_flag*, integer, intent(in) *flush\_flag*, integer, intent(in) *flood\_flag*, integer, intent(in) *grav\_heat\_flag*, integer, intent(in) *flush\_heat\_flag*, integer, intent(in) *harmonic\_flag*, integer, intent(in) *prescribe\_flag*, integer, intent(in) *salt\_flag*, integer, intent(in) *turb\_flag*, integer, intent(in) *bottom\_flag*, integer, intent(in) *tank\_flag*, integer, intent(in) *precip\_flag*, integer, intent(in) *bgc\_flag*, integer, intent(in) *N\_bgc*, real(wp), intent(in) *k\_snow\_flush* )

Settings output.

Writes important values to latter identify run.

#### Revision History

created by Philipp Griewank, IMPRS (2011-02-12)

#### Parameters

in	<i>description</i>	Niels, 2017
----	--------------------	-------------

## 4.12 mo\_parameters Module Reference

Module determines physical constants to be used by the SAMSIM Seaice model.

#### Variables

- integer, parameter **wp** = SELECTED\_REAL\_KIND(12, 307)  
*set working precision \_wp*
- real, parameter **pi** = 3.1415\_wp
- real, parameter **grav** = 9.8061\_wp  
*gravitational constant [m/s<sup>2</sup>]*
- real(**wp**), parameter **k\_s** = 2.2\_wp  
*solid heat conductivity [J / m s K] 2.2*
- real(**wp**), parameter **k\_l** = 0.523\_wp  
*liquid heat conductivity [J / m s K] 0.523*
- real(**wp**), parameter **c\_s** = 2020.0\_wp  
*solid heat capacity [J/ kg K]*
- real(**wp**), parameter **c\_s\_beta** = 7.6973\_wp  
*linear solid heat capacity approximation [J/ kg K<sup>2</sup>] c\_s = c\_s+c\_s\_beta\*T*
- real(**wp**), parameter **c\_l** = 3400.\_wp  
*liquid heat capacity [J/ kg K]*
- real(**wp**), parameter **rho\_s** = 920.\_wp  
*density of solid [kg / m<sup>3</sup>]*
- real(**wp**), parameter **rho\_l** = 1028.0\_wp  
*density of liquid [kg / m<sup>3</sup>]*
- real(**wp**), parameter **latent\_heat** = 333500.\_wp

- latent heat release [J/kg]*
- real(wp), parameter `zerok` = 273.15\_wp  
*Zero degrees Celsius in Kelvin [K].*
- real(wp), parameter `bbeta` = 0.8\_wp\*1e-3  
*concentration expansion coefficient [kg / (m<sup>3</sup> ppt)]*
- real(wp), parameter `mu` = 2.55\_wp\*1e-3  
*dynamic viscosity [kg / m s]*
- real(wp), parameter `kappa_l` = `k_l/rho_l/c_l`  
*heat diffusivity of water*
- real(wp), parameter `sigma` = 5.6704\_wp\*1e-8  
*Stefan Boltzmann constant [W/(m<sup>2</sup>\*K<sup>4</sup>)]*
- real(wp), parameter `psi_s_min` = 0.05\_wp  
*The amount of ice that the lowest layer can have before it counts as an ice layer.*
- real(wp), parameter `neg_free` = -0.05\_wp  
*The distance the freeboard can be below 0 before water starts flooding through cracks.*
- real(wp), parameter `x_grav` = 0.000584\_wp
- real(wp), parameter `ray_crit` = 4.89\_wp
- real(wp), parameter `para_flush_horiz` = 1.0\_wp  
*determines relationship of horizontal flow distance in during flushing (guess 1)*
- real(wp), parameter `para_flush_gamma` = 0.9\_wp  
*Strength of desalination per timestep (guess)*
- real(wp), parameter `psi_s_top_min` = 0.40\_wp  
*if psi\_s is below this value meltwater forms (guess) 0.4*
- real(wp), parameter `ratio_flood` = 1.50\_wp  
*Ratio of flooded to dissolve snow, plays an important role in subroutine flood.*
- real(wp), parameter `ref_salinity` = 34.\_wp  
*Reference salinity [g/kg] used to calculate freshwater column.*
- real(wp), parameter `rho_snow` = 330.\_wp  
*density of new snow [kg/m<sup>3</sup>], I< Niels, 2017 add: can be adjusted to lab values if they are measured*
- real(wp), parameter `gas_snow_ice` = 0.10\_wp  
*volume of gas percentage in new snow ice due to flooding, no longer used*
- real(wp), parameter `gas_snow_ice2` = 0.20\_wp  
*volume of gas percentage in new snow ice due to snow melting (Eicken 95)*
- real(wp), parameter `emissivity_ice` = 0.95\_wp  
*Emissivity of water and ice.*
- real(wp), parameter `emissivity_snow` = 1.00\_wp  
*Emissivity of Snow.*
- real(wp), parameter `penetr` = 0.30\_wp  
*Amount of penetrating sw radiation.*
- real(wp), parameter `extinc` = 2.00\_wp  
*Extinction coefficient of ice.*
- real(wp), parameter `turb_a` = 0.1\_wp\*0.05\_wp\*rho\_l/86400.\_wp  
*Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.*
- real(wp), parameter `turb_b` = 0.05\_wp  
*Exponential turbulence slope [m<sup>3</sup>/kg] WARNING no source, simple guess.*
- real(wp) `max_flux_plate` = 50.0  
*Maximal heating rate of a heating plate.*
- real(wp) `k_snow_flush` = 0.75\_wp  
*Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.*
- real(wp) `k_styropor` = 0.8\_wp  
*Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)*

### 4.12.1 Detailed Description

Module determines physical constants to be used by the SAMSIM Seaice model.

Many values are taken from Notz 2005, Table 5.2.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

#### Revision History

Started by Philipp Griewank 2010-07-08

add parameters: heat conductivity of styropor under special sea ice lab conditions and ratio of penetrating melt water by Niels Fuchs, MPIMET (2017-03-01)

### 4.12.2 Variable Documentation

#### 4.12.2.1 `real(wp), parameter mo_parameters::bbeta = 0.8_wp*1e-3`

concentration expansion coefficient [kg / (m<sup>3</sup> ppt)]

#### 4.12.2.2 `real(wp), parameter mo_parameters::c_l = 3400._wp`

liquid heat capacity [J/ kg K]

#### 4.12.2.3 `real(wp), parameter mo_parameters::c_s = 2020.0_wp`

solid heat capacity [J/ kg K]

#### 4.12.2.4 `real(wp), parameter mo_parameters::c_s_beta = 7.6973_wp`

linear solid heat capacity approximation [J/ kg K<sup>2</sup>]  $c_s = c_s + c_s\_beta * T$

4.12.2.5 `real(wp), parameter mo_parameters::emissivity_ice = 0.95_wp`

Emissivity of water and ice.

4.12.2.6 `real(wp), parameter mo_parameters::emissivity_snow = 1.00_wp`

Emissivity of Snow.

4.12.2.7 `real(wp), parameter mo_parameters::extinc = 2.00_wp`

Extinction coefficient of ice.

4.12.2.8 `real(wp), parameter mo_parameters::gas_snow_ice = 0.10_wp`

volume of gas percentage in new snow ice due to flooding, no longer used

4.12.2.9 `real(wp), parameter mo_parameters::gas_snow_ice2 = 0.20_wp`

volume of gas percentage in new snow ice due to snow melting (Eicken 95)

4.12.2.10 `real, parameter mo_parameters::grav = 9.8061_wp`

gravitational constant [m/s<sup>2</sup>]

4.12.2.11 `real(wp), parameter mo_parameters::k_l = 0.523_wp`

liquid heat conductivity [J / m s K] 0.523

4.12.2.12 `real(wp), parameter mo_parameters::k_s = 2.2_wp`

solid heat conductivity [J / m s K] 2.2

4.12.2.13 `real(wp) mo_parameters::k_snow_flush = 0.75_wp`

Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.

4.12.2.14 `real(wp) mo_parameters::k_styropor = 0.8_wp`

Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)

4.12.2.15 `real(wp), parameter mo_parameters::kappa_l = k_l/rho_l/c_l`

heat diffusivity of water

4.12.2.16 `real(wp), parameter mo_parameters::latent_heat = 333500._wp`

latent heat release [J/kg]

4.12.2.17 `real(wp) mo_parameters::max_flux_plate = 50.0`

Maximal heating rate of a heating plate.

4.12.2.18 `real(wp), parameter mo_parameters::mu = 2.55_wp*1e-3`

dynamic viscosity [kg /m s]

4.12.2.19 `real(wp), parameter mo_parameters::neg_free = -0.05_wp`

The distance the freeboard can be below 0 before water starts flooding through cracks.

4.12.2.20 `real(wp), parameter mo_parameters::para_flush_gamma = 0.9_wp`

Strength of desalination per timestep (guess)

4.12.2.21 `real(wp), parameter mo_parameters::para_flush_horiz = 1.0_wp`

determines relationship of horizontal flow distance in during flushing (guess 1)

4.12.2.22 `real(wp), parameter mo_parameters::penetr = 0.30_wp`

Amount of penetrating sw radiation.

4.12.2.23 `real, parameter mo_parameters::pi = 3.1415_wp`

4.12.2.24 `real(wp), parameter mo_parameters::psi_s_min = 0.05_wp`

The amount of ice that the lowest layer can have before it counts as an ice layer.

4.12.2.25 `real(wp), parameter mo_parameters::psi_s_top_min = 0.40_wp`

if psi\_s is below this value meltwater forms (guess) 0.4

4.12.2.26 `real(wp), parameter mo_parameters::ratio_flood = 1.50_wp`

Ratio of flooded to dissolve snow, plays an important role in subroutine flood.

4.12.2.27 `real(wp), parameter mo_parameters::ray_crit = 4.89_wp`

4.12.2.28 `real(wp), parameter mo_parameters::ref_salinity = 34._wp`

Reference salinity [g/kg] used to calculate freshwater column.

4.12.2.29 `real(wp), parameter mo_parameters::rho_l = 1028.0_wp`

density of liquid [kg / m<sup>3</sup>]

4.12.2.30 `real(wp), parameter mo_parameters::rho_s = 920._wp`

density of solid [kg / m<sup>3</sup>]

4.12.2.31 `real(wp), parameter mo_parameters::rho_snow = 330._wp`

density of new snow [kg/m<sup>3</sup>], !< Niels, 2017 add: can be adjusted to lab values if they are measured

4.12.2.32 `real(wp), parameter mo_parameters::sigma = 5.6704_wp*1e-8`

Stefan Boltzmann constant [W/(m<sup>2</sup>\*K<sup>4</sup>)].

4.12.2.33 `real(wp), parameter mo_parameters::turb_a = 0.1_wp*0.05_wp*rho_l/86400._wp`

Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.

4.12.2.34 `real(wp), parameter mo_parameters::turb_b = 0.05_wp`

Exponential turbulence slope [m<sup>3</sup>/kg] WARNING no source, simple guess.

4.12.2.35 `integer, parameter mo_parameters::wp = SELECTED_REAL_KIND(12, 307)`

set working precision \_wp

4.12.2.36 `real(wp), parameter mo_parameters::x_grav = 0.000584_wp`

4.12.2.37 `real(wp), parameter mo_parameters::zerok = 273.15_wp`

Zero degrees Celsius in Kelvin [K].

## 4.13 mo\_snow Module Reference

Module contains all things directly related to snow.

### Functions/Subroutines

- subroutine, public [snow\\_coupling](#) (H\_abs\_snow, phi\_s, T\_snow, H\_abs, H, phi, T, m\_snow, S\_abs\_snow, m, S\_bu)  
*Subroutine to couple a thin snow layer to the upper ice layer.*
- subroutine, public [snow\\_precip](#) (m\_snow, H\_abs\_snow, thick\_snow, psi\_s\_snow, dt, liquid\_precip\_in, T2m, solid\_precip\_in)  
*Subroutine for calculating precipitation on an existing snow cover.*
- subroutine, public [snow\\_precip\\_0](#) (H\_abs, S\_abs, m, T, dt, liquid\_precip\_in, T2m, solid\_precip\_in)  
*Subroutine for calculating precipitation into the ocean.*
- subroutine, public [snow\\_thermo](#) (psi\_l\_snow, psi\_s\_snow, psi\_g\_snow, thick\_snow, S\_abs\_snow, H\_abs\_snow, m\_snow, T\_snow, m, thick, H\_abs)  
*Subroutine for calculating snow thermodynamics.*
- subroutine, public [snow\\_thermo\\_meltwater](#) (psi\_l\_snow, psi\_s\_snow, psi\_g\_snow, thick\_snow, S\_abs\_snow, H\_abs\_snow, m\_snow, T\_snow, m, thick, H\_abs, melt\_thick\_snow)  
*Subroutine for calculating snow thermodynamics.*
- subroutine, public [sub\\_fl\\_q\\_0\\_snow\\_thin](#) (m\_snow, thick\_snow, T\_snow, psi\_s, psi\_l, psi\_g, thick, T\_bound, fl\_Q\_snow)  
*Determines conductive Heat flux for combined top ice and snow layer.*
- subroutine, public [sub\\_fl\\_q\\_snow](#) (m\_snow, thick\_snow, T\_snow, psi\_s\_2, psi\_l\_2, psi\_g\_2, thick\_2, T\_2, fl\_Q)  
*Determines conductive Heat flux between Snow and top ice layer.*
- subroutine, public [sub\\_fl\\_q\\_0\\_snow](#) (m\_snow, thick\_snow, T\_snow, T\_bound, fl\_Q)  
*Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner than thick\_min.*
- real(wp) function, public [func\\_k\\_snow](#) (m\_snow, thick\_snow)  
*Calculates the thermal conductivity of the snow layer as a function of the density.*

### 4.13.1 Detailed Description

Module contains all things directly related to snow.

#### Author

Philipp Griewank

### 4.13.2 Function/Subroutine Documentation

#### 4.13.2.1 real(wp) function, public mo\_snow::func\_k\_snow ( real(wp), intent(in) m\_snow, real(wp), intent(in) thick\_snow )

Calculates the thermal conductivity of the snow layer as a function of the density.

Based on the Sturm et al 1997 data fit for densities greater than 0.156 g/cm\*\*3. Warning, Sturm et al use g/cm\*\*3, I use kg/m\*\*3 Snow density probability functions can be included later to raise the effective conductivity. Warning!: added 0.15 to the thermal conductivity.

#### Revision History

Forged by Philipp Griewank (2010-12-13)



4.13.2.2 subroutine, public mo\_snow::snow\_coupling ( real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *phi\_s*, real(wp), intent(inout) *T\_snow*, real(wp), intent(inout) *H\_abs*, real(wp), intent(inout) *H*, real(wp), intent(inout) *phi*, real(wp), intent(inout) *T*, real(wp), intent(in) *m\_snow*, real(wp), intent(in) *S\_abs\_snow*, real(wp), intent(in) *m*, real(wp), intent(in) *S\_bu* )

Subroutine to couple a thin snow layer to the upper ice layer.

Subroutine is activated when *thick\_snow* < *thick\_min*. The enthalpies of the two layers are adjusted until both layers have the same temperatures. The following approach is used.

1. The enthalpies are adjusted so *T\_snow*=0, and *phi\_s*=1.
2. The temperatures are calculated.
3. If the ice temperature is greater 0 the balanced enthalpies are calculated directly. ELSE they are calculated iteratively.

#### Revision History

Written by Philipp Griewank, IMPRS (2011-01-20)

4.13.2.3 subroutine, public mo\_snow::snow\_precip ( real(wp), intent(inout) *m\_snow*, real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *thick\_snow*, real(wp), intent(inout) *psi\_s\_snow*, real(wp), intent(in) *dt*, real(wp), intent(in) *liquid\_precip\_in*, real(wp), intent(in) *T2m*, real(wp), intent(in), optional *solid\_precip\_in* )

Subroutine for calculating precipitation on an existing snow cover.

Can optionally deal with separate solid and liquid precipitation or a single liquid input. The 2 meter temperature determines the temperature of the precipitation. In case of single input the 2 meter temperature determines if snow or rain falls. Snow makes the thickness grow according to the density of new snow (*rho\_snow*), while rain falls into the snow without increasing snow depth. It is necessary to calculate the new *psi\_s\_snow* to ensure proper melting in *snow\_thermo*.

#### Revision History

Sired by Philipp Griewank, IMPRS (2010-12-14)

4.13.2.4 subroutine, public mo\_snow::snow\_precip\_0 ( real(wp), intent(inout) *H\_abs*, real(wp), intent(inout) *S\_abs*, real(wp), intent(in) *m*, real(wp), intent(in) *T*, real(wp), intent(in) *dt*, real(wp), intent(in) *liquid\_precip\_in*, real(wp), intent(in) *T2m*, real(wp), intent(in), optional *solid\_precip\_in* )

Subroutine for calculating precipitation into the ocean.

Can optionally deal with separate solid and liquid precipitation or a single liquid input. The 2 meter temperature determines the temperature of the precipitation. In case of single input the 2 meter temperature determines if snow or rain falls. It is important, that the mass, energy and salt leaving the upper layer must be outputted. This is not the case. Temp!

#### Revision History

Copy and Pasted by Philipp Griewank, IMPRS (2011-01-10)

4.13.2.5 subroutine, public mo\_snow::snow\_thermo ( real(wp), intent(inout) *psi\_l\_snow*, real(wp), intent(inout) *psi\_s\_snow*, real(wp), intent(inout) *psi\_g\_snow*, real(wp), intent(inout) *thick\_snow*, real(wp), intent(inout) *S\_abs\_snow*, real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *m\_snow*, real(wp), intent(inout) *T\_snow*, real(wp), intent(inout) *m*, real(wp), intent(inout) *thick*, real(wp), intent(inout) *H\_abs* )

Subroutine for calculating snow thermodynamics.

Behaves similar to mushy layer sea ice. Important differences are:

1. no expulsion, *thick\_snow* is raised if the volume expands.
2. The liquid fraction is limited.
3. When the liquid fraction exceeds it's limit the thickness of the snow layer is reduced. This is done as follows: Only applies if the fluid fraction is above the irreducible water content as defined in Coleuo-Lasaffre 98.  $thick\_snow = thick\_snow * (1 - wp - (psi\_s\_old - psi\_s\_snow) / psi\_s\_old)$  Warning: the formula for liquid water content in Coleuo-Lasaffre contains 2 typos When the water exceeds the limit water runs down to the bottom of the snow layer. The saturated lower layer is added to the top ice layer.

#### Revision History

Fabricated by Philipp Griewank, IMPRS (2010-12-14)

Major redo, water saturated bottom snow added to top ice layer by Philipp Griewank (2010-12-14)

#### Parameters

in, out	<i>h_abs</i>	Top ice layer variables
---------	--------------	-------------------------

4.13.2.6 subroutine, public mo\_snow::snow\_thermo\_meltwater ( real(wp), intent(inout) *psi\_l\_snow*, real(wp), intent(inout) *psi\_s\_snow*, real(wp), intent(inout) *psi\_g\_snow*, real(wp), intent(inout) *thick\_snow*, real(wp), intent(inout) *S\_abs\_snow*, real(wp), intent(inout) *H\_abs\_snow*, real(wp), intent(inout) *m\_snow*, real(wp), intent(inout) *T\_snow*, real(wp), intent(inout) *m*, real(wp), intent(inout) *thick*, real(wp), intent(inout) *H\_abs*, real(wp), intent(inout) *melt\_thick\_snow* )

Subroutine for calculating snow thermodynamics.

most of the physics are taken from [snow\\_thermo\(\)](#) based on lab observations: parts of the snow meltwater percolate directly into the ice

#### Revision History

introduced by Niels Fuchs (2016-10-13)

#### Parameters

in, out	<i>h_abs</i>	Top ice layer variables
---------	--------------	-------------------------

4.13.2.7 subroutine, public mo\_snow::sub\_fl\_q\_0\_snow ( real(wp), intent(in) *m\_snow*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *T\_snow*, real(wp), intent(in) *T\_bound*, real(wp), intent(out) *fl\_Q* )

Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner then *thick\_min*.

#### Revision History

first version by Philipp Griewank (2010-12-15)

Artificial limitation introduced by Philipp Griewank (2011-01-17)

#### Parameters

in	<i>t_bound</i>	T_bound temperature of boundary layer
----	----------------	---------------------------------------

4.13.2.8 subroutine, public mo\_snow::sub\_fl\_q\_0\_snow\_thin ( real(wp), intent(in) *m\_snow*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *T\_snow*, real(wp), intent(in) *psi\_s*, real(wp), intent(in) *psi\_l*, real(wp), intent(in) *psi\_g*, real(wp), intent(in) *thick*, real(wp), intent(in) *T\_bound*, real(wp), intent(out) *fl\_Q\_snow* )

Determines conductive Heat flux for combined top ice and snow layer.

When *thick\_snow* < *thick\_min*.

#### Revision History

first version by Philipp Griewank (2011-01-19)

4.13.2.9 subroutine, public mo\_snow::sub\_fl\_q\_snow ( real(wp), intent(in) *m\_snow*, real(wp), intent(in) *thick\_snow*, real(wp), intent(in) *T\_snow*, real(wp), intent(in) *psi\_s\_2*, real(wp), intent(in) *psi\_l\_2*, real(wp), intent(in) *psi\_g\_2*, real(wp), intent(in) *thick\_2*, real(wp), intent(in) *T\_2*, real(wp), intent(out) *fl\_Q* )

Determines conductive Heat flux between Snow and top ice layer.

Standard approach.

#### Revision History

first version by Philipp Griewank (2010-12-15)

## 4.14 mo\_testcase\_specifics Module Reference

Module contains changes specific testcases require during the main timeloop.

## Functions/Subroutines

- subroutine, public [sub\\_test1](#) (time, T\_top)  
*Subroutine for changing T\_top for testcase 1.*
- subroutine, public [sub\\_test2](#) (time, T2m)  
*Subroutine for changing T\_top for testcase 2.*
- subroutine, public [sub\\_test9](#) (time, T2m)  
*Subroutine for changing T2m for testcase 9.*
- subroutine, public [sub\\_test34](#) (time, T2m)  
*Subroutine for changing T2m for testcase 34.*
- subroutine, public [sub\\_test3](#) (time, liquid\_precip, solid\_precip)  
*Subroutine for setting snow for testcase 3.*
- subroutine, public [sub\\_test4](#) (time, fl\_q\_bottom)  
*Subroutine for setting snow for testcase 4.*
- subroutine, public [sub\\_test6](#) (time, T2m)  
*Subroutine for changing T\_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.*

### 4.14.1 Detailed Description

Module contains changes specific testcases require during the main timeloop.

Most settings related to the testcases are defined in [mo\\_init](#), but if changes to the code need to applied after the timestepping has begun they are located here. Changes were initially simply implemented in the main timeloop, but things got confusing.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

#### Revision History

Removed from [mo\\_grotz](#) by Philipp Griewank, IMPRS (2014-04-16)

### 4.14.2 Function/Subroutine Documentation

4.14.2.1 subroutine, public `mo_testcase_specifics::sub_test1 ( real(wp), intent(in) time, real(wp), intent(inout) T_top )`

Subroutine for changing T\_top for testcase 1.

#### Revision History

Formed by Philipp Griewank, IMPRS (2014-04-16)

#### 4.14.2.2 subroutine, public mo\_testcase\_specifics::sub\_test2 ( real(wp), intent(in) *time*, real(wp), intent(inout) *T2m* )

Subroutine for changing T\_top for testcase 2.

T2m is adjusted over time.

##### Revision History

Formed by Philipp Griewank, IMPRS (2014-04-17)

#### 4.14.2.3 subroutine, public mo\_testcase\_specifics::sub\_test3 ( real(wp), intent(in) *time*, real(wp), intent(inout) *liquid\_precip*, real(wp), intent(inout) *solid\_precip* )

Subroutine for setting snow for testcase 3.

Precipitation rates are set

##### Revision History

Formed by Philipp Griewank, (2014-04-18)

#### 4.14.2.4 subroutine, public mo\_testcase\_specifics::sub\_test34 ( real(wp), intent(in) *time*, real(wp), intent(inout) *T2m* )

Subroutine for changing T2m for testcase 34.

T2m is adjusted over time.

##### Revision History

adjusted by Niels Fuchs, MPI (2016-01-18)

#### 4.14.2.5 subroutine, public mo\_testcase\_specifics::sub\_test4 ( real(wp), intent(in) *time*, real(wp), intent(inout) *fl\_q\_bottom* )

Subroutine for setting snow for testcase 4.

##### Revision History

Formed by Philipp Griewank, (2014-04-18)

#### 4.14.2.6 subroutine, public mo\_testcase\_specifics::sub\_test6 ( real(wp), intent(in) *time*, real(wp), intent(inout) *T2m* )

Subroutine for changing T\_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.

##### Revision History

Formed by Philipp Griewank, IMPRS (2014-04-38)

#### 4.14.2.7 subroutine, public mo\_testcase\_specifics::sub\_test9 ( real(wp), intent(in) time, real(wp), intent(inout) T2m )

Subroutine for changing T2m for testcase 9.

T2m is adjusted over time.

#### Revision History

Formed by Niels Fuchs, MPI (2016-01-18)

## 4.15 mo\_thermo\_functions Module Reference

Contains subroutines and functions related to multi-phase thermodynamics.

### Functions/Subroutines

- subroutine, public [gett](#) (H, S\_bu, T\_in, T, phi, k)  
*Determines equilibrium Temperature of a layer for given S\_bu and H as well as solid fraction.*
- subroutine, public [expulsion](#) (phi, thick, m, psi\_s, psi\_l, psi\_g, V\_ex)  
*Determines Brine flux expelled from out of a layer due to freezing.*
- subroutine, public [sub\\_fl\\_q](#) (psi\_s\_1, psi\_l\_1, psi\_g\_1, thick\_1, T\_1, psi\_s\_2, psi\_l\_2, psi\_g\_2, thick\_2, T\_2, fl\_Q)  
*Determines conductive heat flux between two layers.*
- subroutine, public [sub\\_fl\\_q\\_0](#) (psi\_s, psi\_l, psi\_g, thick, T, T\_bound, direct\_flag, fl\_Q)  
*Determines conductive Heat flux between layer and boundary temperatures.*
- subroutine, public [sub\\_fl\\_q\\_styropor](#) (k\_styropor, fl\_Q)  
*Niels, 2017 add: Determines conductive Heat flux below styropor cover.*
- real(wp) function, public [func\\_s\\_br](#) (T, S\_bu)  
*Computes salinity of brine pockets for given temperature in Celsius of mushy layer.*
- real(wp) function, public [func\\_ddt\\_s\\_br](#) (T)  
*Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.*

### 4.15.1 Detailed Description

Contains subroutines and functions related to multi-phase thermodynamics.

See the subroutine and function descriptions for details.

#### Author

Philipp Griewank

#### COPYRIGHT

This file is part of SAMSIM.

SAMSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

SAMSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with SAMSIM. If not, see <http://www.gnu.org/licenses/>.

#### Revision History

Started by Philipp Griewank 2010-07-08

Add function for styropor cover by Niels Fuchs, MPIMET (2017-01-03)

Modified salinity functions by Philipp Griewank, Uni K (2018-08-01)

## 4.15.2 Function/Subroutine Documentation

4.15.2.1 subroutine, public mo\_thermo\_functions::expulsion ( real(wp), intent(in) *phi*, real(wp), intent(in) *thick*, real(wp), intent(inout) *m*, real(wp), intent(out) *psi\_s*, real(wp), intent(out) *psi\_l*, real(wp), intent(out) *psi\_g*, real(wp), intent(out) *V\_ex* )

Determines Brine flux expelled from out of a layer due to freezing.

If the volume of ice and brine exceed the Volume of the layer brine is expelled. The volume of the ejected brine is calculated and exported. The volume fractions are also calculated.

### Revision History

first version by Philipp Griewank, (2010-07-19)  
changes to mass, Enthalpy and Salinity are now computed in subroutine mass\_transfer by Philipp Griewank, (2010-08-24)

4.15.2.2 real(wp) function, public mo\_thermo\_functions::func\_ddt\_s\_br ( real(wp), intent(in) *T* )

Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.

Subroutine computes one ddT\_S\_br for one given T NaCl solutions and seawater produce slight variations. Which solution is used is specified by salt\_flag. Based on Notz 2005 p. 36  $ddT\_S\_br = c2 + 2 * c3 * T + 3 * c4 * T^2$

For T below -20 we simply use a linear extension based on "Composition of sea ice and its tensile strength". The actual salinity function is much more complicated and depends on the salt composition, but the linear fit is far better than using the polynomial fit.

The NaCl precipitates at -22, leading to ice/salt kristall mix below -22. Ideally the whole code would be modified to take the non-continous transition at -22 but given that there is currently little interest I can't be bothered to put in the effort.

### Revision History

First version by Philipp Griewank (2010-07-13)  
Added linear bit by Philipp Griewank (2018-07-22)

### Parameters

in	<i>t</i>	Temperature in Celsius
----	----------	------------------------

### Returns

derivative of Brine salinity

4.15.2.3 real(wp) function, public mo\_thermo\_functions::func\_s\_br ( real(wp), intent(in) *T*, real(wp), intent(in), optional *S\_bu* )

Computes salinity of brine pockets for given temperature in Celsius of mushy layer.

Subroutine computes one  $S_{br}$  for one given  $T$  in Celsius by third-order polynomial. NaCl solutions and seawater produce slight variations. Which solution is used is determined by `salt_flag`.  $S_{br} = c_1 + c_2 * T + c_3 * T^2 + c_4 * T^3$   
Based on Notz 2005 p. 36

For  $T$  below -20 we simply use a linear extension based on "Composition of sea ice and its tensile strength". The actual salinity function is much more complicated and depends on the salt composition, but the linear fit is far better than using the polynomial fit.

The NaCl precipitates at -22, leading to ice/salt kristall mix below -22. Ideally the whole code would be modified to take the non-continuous transition at -22 but given that there is currently little interest I can't be bothered to put in the effort.

#### Revision History

First version by Philipp Griewank (2010-07-12)  
Changed to go through 0 by Philipp Griewank (2014-05-07)  
Added linear bit by Philipp Griewank (2018-07-22)

#### Parameters

in	$t$	Temperature in Celsius
----	-----	------------------------

#### Returns

Brine salinity

4.15.2.4 subroutine, public `mo_thermo_functions::gett ( real(wp), intent(in) H, real(wp), intent(in) S_bu, real(wp), intent(in) T_in, real(wp), intent(out) T, real(wp), intent(out) phi, integer, intent(in), optional k )`

Determines equilibrium Temperature of a layer for given  $S_{bu}$  and  $H$  as well as solid fraction.

The temperature of a fully liquid layer is used to see if the resulting brine salinity is lower than the bulk salinity. After checking if the layer is a fluid or a mushy layer the temperature is calculated by solving  $f(T) = 0$  using the Newton method.  $f(T) = -\text{latent\_heat} - H + \text{latent\_heat} * S_{bu} / S_{br}(T) + c_s * T + c_{s\_beta} * T^2 / 2$   $f'(T) = c_s + c_{s\_beta} * T - \text{latent\_heat} * S_{bu} * S_{br}'(T) / S_{br}^2$  Described in Notz2005, subsubsection 5.6.1. See `func_S_br(T)` and `func_ddT_S_br(T)`. First guess  $T_0$  must be given, low first guess lead to overshooting which would lead to very high Temperatures. To avoid this, an if loop sets  $T$  to freezing  $T$  when  $T > 0$ . Freezing  $T$  is also calculated at the beginning using the Newton-Method. If  $S_{bu} < 0.001$  then it is treated as pure ice.

#### Revision History

first version by Philipp Griewank (2010-07-13)  
Freezing temperature is calculated and introduced if  $T$  goes above 0 by Philipp Griewank (2010-07-13)  
Added if loops to deal with saltless ice by Philipp Griewank (2010-11-27)

#### Parameters

in	$h$	Enthalpy [J/kg]
in	$s_{bu}$	Bulk Salinity [g/kg]
in	$t_{in}$	input Temperature for $T_0$ [C]
out	$t$	Temperature [C]
out	$\phi$	solid fraction



4.15.2.5 subroutine, public mo\_thermo\_functions::sub\_fl\_q ( real(wp), intent(in) *psi\_s\_1*, real(wp), intent(in) *psi\_l\_1*, real(wp), intent(in) *psi\_g\_1*, real(wp), intent(in) *thick\_1*, real(wp), intent(in) *T\_1*, real(wp), intent(in) *psi\_s\_2*, real(wp), intent(in) *psi\_l\_2*, real(wp), intent(in) *psi\_g\_2*, real(wp), intent(in) *thick\_2*, real(wp), intent(in) *T\_2*, real(wp), intent(out) *fl\_Q* )

Determines conductive heat flux between two layers.

Details can be found in Notz 2005, especially equation 5.7. The gas volume is assumed to have no thermal properties at all. First the thermal resistance *R* is calculated using the approximated thermal conductivity of the mushy layer (see Notz 2005 eq. 3.41.). Then the heat flux *Q* is simply  $(T_1 - T_2)/R$  "*\_1*" denotes the upper layer and "*\_2*" the lower layer. A positive heat flux is from lower to upper layer.

#### Revision History

First version by Philipp Griewank (2010-07-21)

4.15.2.6 subroutine, public mo\_thermo\_functions::sub\_fl\_q\_0 ( real(wp), intent(in) *psi\_s*, real(wp), intent(in) *psi\_l*, real(wp), intent(in) *psi\_g*, real(wp), intent(in) *thick*, real(wp), intent(in) *T*, real(wp), intent(in) *T\_bound*, integer *direct\_flag*, real(wp), intent(out) *fl\_Q* )

Determines conductive Heat flux between layer and boundary temperatures.

Details can be found in Notz 2005, especially equation 5.10 and 5.11. The gas volume is assumed to have no thermal properties. *direct\_flag* denotes if the boundary layer is above or below the layer. 1 : = layer above boundary  
-1: = layer below boundary

#### Revision History

first version by Philipp Griewank (2010-07-21)

#### Parameters

in	<i>t_bound</i>	<i>T_bound</i> temperature of boundary layer
----	----------------	--

4.15.2.7 subroutine, public mo\_thermo\_functions::sub\_fl\_q\_styropor ( real(wp), intent(in) *k\_styropor*, real(wp), intent(inout) *fl\_Q* )

Niels, 2017 add: Determines conductive Heat flux below styropor cover.

Standard approach.

#### Revision History

first version by Niels Fuchs, MPIMET (2017-01-03)



## Chapter 5

# File Documentation

### 5.1 gpl\_license.txt File Reference

#### Functions

- GNU GENERAL PUBLIC LICENSE June [Copyright](#) (C) 2007 Free Software Foundation

#### Variables

- GNU GENERAL PUBLIC LICENSE [Version](#)
- GNU GENERAL PUBLIC LICENSE June Inc< <https://www.gnu.org/licenses/gpl-3.0.html>:Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.  
Preamble The GNU General Public License is a free, copyleft license for software and other kinds of works. The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too. When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things. To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others. For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights. Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it. For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions. Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users. Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish

to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free. The precise terms and conditions for copying, distribution and modification follow.

**TERMS AND CONDITIONS**

**0. Definitions.** "This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks. "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work. A "covered work" means either the unmodified Program or a work based on the Program. To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well. To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying. An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

**1. Source Code.** The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work. A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language. The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it. The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work. The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work.

**2. Basic Permissions.** All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law. You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you. Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

**3. Protecting Users' Legal Rights From Anti-Circumvention Law.** No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures. When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid cir-

cumvention of technological measures.

**4. Conveying Verbatim Copies.** You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program. You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

**5. Conveying Modified Source Versions.** You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
- c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

**6. Conveying Non-Source Forms.** You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
- c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or inter-

ferred with solely because modification has been made. If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM). The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network. Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

**7. Additional Terms.** "Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions. When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission. Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
- d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying. If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms. Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

**8. Termination.** You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11). However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation. Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice. Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.9.

**9. Acceptance Not Required for Having Copies.** You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not

accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients. Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License. An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts. You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents. A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version". A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License. Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version. In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party. If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid. If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it. A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007. Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom. If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License. Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License

will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License. The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation. If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program. Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty. THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16. If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

### How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms. To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
< one line to give the program's name and a brief idea of what it does.
> Copyright(C) < year > < name of author > This program is free software without even the implied warranty
of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE See the GNU General Public License
for more details You should have received a copy of the GNU General Public License along with this program
If not
```

## 5.1.1 Function Documentation

### 5.1.1.1 GNU GENERAL PUBLIC LICENSE June Copyright ( C )

## 5.1.2 Variable Documentation



5.1.2.1 GNU GENERAL PUBLIC LICENSE June Inc<<https://www.gnu.org/licenses/gpl-3.0.html>: Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble The GNU General Public License is a free, copyleft license for software and other kinds of works. The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too. When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things. To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others. For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights. Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it. For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions. Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users. Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free. The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions. "This License" refers to version 3 of the GNU General Public License. "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks. "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations. To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work. A "covered work" means either the unmodified Program or a work based on the Program. To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well. To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying. An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code. The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work. A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language. The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it. The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data- or control flow between those subprograms and other parts of the work. The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source. The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions. All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from

## 5.1.2.2 GNU GENERAL PUBLIC LICENSE Version

## 5.2 mo\_data.f90 File Reference

## Modules

- module [mo\\_data](#)  
*Sets data and contains all flag descriptions.*

## Variables

- real(wp), dimension(:), allocatable [mo\\_data::h](#)  
*Enthalpy [J].*
- real(wp), dimension(:), allocatable [mo\\_data::h\\_abs](#)  
*specific Enthalpy [J/kg]*
- real(wp), dimension(:), allocatable [mo\\_data::q](#)  
*Heat in layer [J].*
- real(wp), dimension(:), allocatable [mo\\_data::fl\\_q](#)  
*Heat flux between layers [J/s].*
- real(wp), dimension(:), allocatable [mo\\_data::t](#)  
*Temperature [C].*
- real(wp), dimension(:), allocatable [mo\\_data::s\\_bu](#)  
*Bulk Salinity [g/kg].*
- real(wp), dimension(:), allocatable [mo\\_data::fl\\_s](#)  
*Salinity flux [(g/s)].*
- real(wp), dimension(:), allocatable [mo\\_data::s\\_abs](#)  
*Absolute Salinity [g].*
- real(wp), dimension(:), allocatable [mo\\_data::s\\_br](#)  
*Brine salinity [g/kg].*
- real(wp), dimension(:), allocatable [mo\\_data::thick](#)  
*Layer thickness [m].*
- real(wp), dimension(:), allocatable [mo\\_data::m](#)  
*Mass [kg].*
- real(wp), dimension(:), allocatable [mo\\_data::fl\\_m](#)  
*Mass fluxes between layers [kg].*
- real(wp), dimension(:), allocatable [mo\\_data::v\\_s](#)  
*Volume [m<sup>3</sup>] of solid.*
- real(wp), dimension(:), allocatable [mo\\_data::v\\_l](#)  
*Volume [m<sup>3</sup>] of liquid.*
- real(wp), dimension(:), allocatable [mo\\_data::v\\_g](#)  
*Volume [m<sup>3</sup>] of gas.*
- real(wp), dimension(:), allocatable [mo\\_data::v\\_ex](#)  
*Volume of brine due expelled due to freezing [m<sup>3</sup>] of solid, gas & liquid.*
- real(wp), dimension(:), allocatable [mo\\_data::phi](#)  
*Solid mass fraction.*
- real(wp), dimension(:), allocatable [mo\\_data::psi\\_s](#)  
*Solid volume fraction.*
- real(wp), dimension(:), allocatable [mo\\_data::psi\\_l](#)  
*Liquid volume fraction.*

- real(wp), dimension(:), allocatable `mo_data::psi_g`  
*Gas volume fraction.*
- real(wp), dimension(:), allocatable `mo_data::ray`  
*Rayleigh number of each layer.*
- real(wp), dimension(:), allocatable `mo_data::perm`
- real(wp), dimension(:), allocatable `mo_data::flush_v`
- real(wp), dimension(:), allocatable `mo_data::flush_h`
- real(wp), dimension(:), allocatable `mo_data::flush_v_old`
- real(wp), dimension(:), allocatable `mo_data::flush_h_old`  
*Permeability [?].*
- real(wp) `mo_data::dt`  
*Timestep [s].*
- real(wp) `mo_data::thick_0`  
*Initial layer thickness [m].*
- real(wp) `mo_data::time`  
*Time [s].*
- real(wp) `mo_data::freeboard`  
*Height of ice surface above (or below) waterlevel [m].*
- real(wp) `mo_data::t_freeze`  
*Freezing temperature [C].*
- integer `mo_data::nlayer`  
*Number of layers.*
- integer `mo_data::n_bottom`  
*Number of bottom layers.*
- integer `mo_data::n_middle`  
*Number of middle layers.*
- integer `mo_data::n_top`  
*Number of top layers.*
- integer `mo_data::n_active`  
*Number of Layers active in the present.*
- integer `mo_data::i`  
*Index, normally used for time.*
- integer `mo_data::k`  
*Index, normally used for layer.*
- integer `mo_data::styropor_flag`
- real(wp) `mo_data::time_out`  
*Time between outputs [s].*
- real(wp) `mo_data::time_total`  
*Time of simulation [s].*
- integer `mo_data::i_time`  
*Number of timesteps.*
- integer `mo_data::i_time_out`  
*Number of timesteps between each output.*
- integer `mo_data::n_time_out`  
*Counts number of timesteps between output.*
- character \*12000 `mo_data::format_t`
- character \*12000 `mo_data::format_psi`
- character \*12000 `mo_data::format_thick`
- character \*12000 `mo_data::format_snow`
- character \*12000 `mo_data::format_integer`
- character \*12000 `mo_data::format_t2m_top`

- character \*12000 [mo\\_data::format\\_bgc](#)
- character \*12000 [mo\\_data::format\\_melt](#)  
*Format strings for output. Niels(2017) add: melt output.*
- character \*12000 [mo\\_data::format\\_perm](#)  
*Niels(2017) add: permeability output.*
- real(wp) [mo\\_data::t\\_bottom](#)  
*Temperature of water beneath the ice [C].*
- real(wp) [mo\\_data::t\\_top](#)  
*Temperature at the surface [C].*
- real(wp) [mo\\_data::s\\_bu\\_bottom](#)  
*Salinity beneath the ice [g/kg].*
- real(wp) [mo\\_data::t2m](#)  
*Two meter Temperature [C].*
- real(wp) [mo\\_data::fl\\_q\\_bottom](#)  
*Bottom heat flux [J\*s].*
- real(wp) [mo\\_data::psi\\_s\\_snow](#)  
*Solid volume fraction of snow layer.*
- real(wp) [mo\\_data::psi\\_l\\_snow](#)  
*Liquid volume fraction of snow layer.*
- real(wp) [mo\\_data::psi\\_g\\_snow](#)  
*Gas volume fraction of snow layer.*
- real(wp) [mo\\_data::phi\\_s](#)  
*Solid mass fraction of snow layer.*
- real(wp) [mo\\_data::s\\_abs\\_snow](#)  
*Absolute salinity of snow layer [g].*
- real(wp) [mo\\_data::h\\_abs\\_snow](#)  
*Absolute enthalpy of snow layer [J].*
- real(wp) [mo\\_data::m\\_snow](#)  
*Mass of snow layer [kg].*
- real(wp) [mo\\_data::t\\_snow](#)  
*Temperature of snow layer [C].*
- real(wp) [mo\\_data::thick\\_snow](#)
- real(wp) [mo\\_data::test](#)  
*Thickness of snow layer [m].*
- real(wp) [mo\\_data::liquid\\_precip](#)  
*Liquid precip, [meter of water/s].*
- real(wp) [mo\\_data::solid\\_precip](#)  
*Solid precip, [meter of water /s].*
- real(wp) [mo\\_data::fl\\_q\\_snow](#)  
*flow of heat into the snow layer*
- real(wp) [mo\\_data::energy\\_stored](#)  
*Total amount of energy stored, control is freezing point temperature of S\_bu\_bottom [J].*
- real(wp) [mo\\_data::total\\_resist](#)  
*Thermal resistance of the whole column [].*
- real(wp) [mo\\_data::surface\\_water](#)  
*Percentage of water fraction in the top 5cm [%].*
- real(wp) [mo\\_data::freshwater](#)  
*Meters of freshwater stored in column [m].*
- real(wp) [mo\\_data::thickness](#)  
*Meters of ice [m].*
- real(wp) [mo\\_data::bulk\\_salin](#)

- Salt/Mass [ppt].*

  - real(wp) `mo_data::thick_min`

*Parameter for snow, determines when snow is in thermal equilibrium with the ice and when it is totally neglected.*
  - real(wp), save `mo_data::t_test`

*First guess for getT subroutine.*
  - real(wp) `mo_data::albedo`

*Amount of short wave radiation which is reflected at the top surface.*
  - real(wp) `mo_data::fl_sw`

*Incoming shortwave radiation [W/m\*\*2].*
  - real(wp) `mo_data::fl_lw`

*Incoming longwave radiation [W/m\*\*2].*
  - real(wp) `mo_data::fl_sen`

*Sensitive heat flux [W/m\*\*2].*
  - real(wp) `mo_data::fl_lat`

*Latent heat flux [W/m\*\*2].*
  - real(wp) `mo_data::fl_rest`

*Bundled longwave,sensitive and latent heat flux [W/m\*\*2].*
  - real(wp), dimension(:), allocatable `mo_data::fl_rad`

*Energy flux of absorbed sw radiation of each layer [J/s].*
  - real(wp) `mo_data::grav_drain`

*brine flux of gravity drainage between two outputs [kg/s]*
  - real(wp) `mo_data::grav_salt`

*salt flux moved by gravity drainage between two outputs [kg\*ppt/s]*
  - real(wp) `mo_data::grav_temp`

*average temperature of gravity drainage brine between two outputs [T]*
  - real(wp) `mo_data::melt_thick`

*thickness of fully liquid part of top layer [m]*
  - real(wp) `mo_data::melt_thick_snow`
  - real(wp) `mo_data::melt_thick_snow_old`

*Niels(2017) add: thickness of excess fully liquid part from snow\_melt\_processes [m].*
  - real(wp), dimension(3) `mo_data::melt_thick_output`

*Niels, 2017 add: output field of surface liquid meltwater sizes.*
  - real(wp) `mo_data::alpha_flux_instable`

*Proportionality constant which determines energy flux by the temperature difference  $T_{top} > T_{2m}$  [W/C].*
  - real(wp) `mo_data::alpha_flux_stable`

*Proportionality constant which determines energy flux by the temperature difference  $T_{top} < T_{2m}$  [W/C].*
  - integer `mo_data::atmoflux_flag`

*1: Use mean climatology of Notz, 2: Use imported reanalysis data, 3: use fixed values defined in `mo_init`*
  - integer `mo_data::grav_flag`

*1: no gravity drainage, 2: Gravity drainage, 3: Simple Drainage*
  - integer `mo_data::prescribe_flag`

*1: nothing happens, 2: prescribed Salinity profile is prescribed at each timestep (does not disable brine dynamics, just overwrites the salinity!)*
  - integer `mo_data::grav_heat_flag`

*1: nothing happens, 2: compensates heatfluxes in `grav_flag` = 2*
  - integer `mo_data::flush_heat_flag`

*1: nothing happens, 2: compensates heatfluxes in `flush_flag` = 5*
  - integer `mo_data::turb_flag`

*1: No bottom turbulence, 2: Bottom mixing*
  - integer `mo_data::salt_flag`

*1: Sea salt, 2: NaCL*

- integer `mo_data::boundflux_flag`  
1: top and bottom cooling plate, 2: top Notz fluxes, bottom cooling plate 3: top flux= $a \cdot (T - T_s)$
- integer `mo_data::flush_flag`  
1: no flushing, 4: meltwater is removed artificially, 5: vert and horiz flushing, 6: simplified
- integer `mo_data::flood_flag`  
1: no flooding, 2: normal flooding, 3: simple flooding
- integer `mo_data::bottom_flag`  
1: nothing changes, 2: deactivates all bottom layer dynamics, useful for some debugging and idealized tests
- integer `mo_data::debug_flag`  
1: no raw layer output, 2: each layer is output at every timestep (warning, file size can be very large)
- integer `mo_data::precip_flag`  
0: solid and liquid precipitation, 1: phase determined by T2m
- integer `mo_data::harmonic_flag`  
1: minimal permeability is used to calculate Rayleigh number, 2: harmonic mean is used for Rayleigh number
- integer `mo_data::tank_flag`  
1: nothing, 2: `S_bu_bottom` and `bgc_bottom` are calculated as if the experiment is conducted in a tank
- integer `mo_data::albedo_flag`  
1: simple albedo, 2: normal albedo, see `func_albedo` for details
- integer `mo_data::lab_snow_flag`  
Niels, 2017 add: 0: lab setup without snow covers, 1: lab setup include snow influence on heat fluxes.
- integer `mo_data::freeboard_snow_flag`  
Niels, 2017 add: 0: respect the mass of snow in the freeboard calculation, 1: don't.
- integer `mo_data::snow_flush_flag`  
Niels, 2017 add: 0: all meltwater from snow forms slush, 1: meltwater partly leads to flushing, ratio defined by "`k_snow_flush`".
- integer `mo_data::snow_precip_flag`  
Niels, 2017 add: 0: all precipitation is set to zero, 1: physical behaviour.
- integer `mo_data::length_input`  
Sets the input length for `atmoflux_flag==2`, common value of 13169.
- real(wp), dimension(:), allocatable `mo_data::tinput`  
Niels, 2017 add: used to read in top temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `mo_data::precipinput`  
Niels, 2017 add: used to read in precipitation for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `mo_data::ocean_t_input`  
Niels, 2017 add: used to read in ocean temperature for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `mo_data::ocean_flux_input`  
Niels, 2017 add: used to read in oceanic heat flux for field experiment tests, dimension needs to be set in the code.
- real(wp), dimension(:), allocatable `mo_data::styropor_input`  
Niels, 2017 add: if styropor is used in the lab on top of the ice to simulate snow heat fluxes.
- real(wp), dimension(:), allocatable `mo_data::ttop_input`  
Niels, 2017 add: used for testcase 111, comparison with greenland harp data, uppermost harp temperature is seen as `Ttop`.
- real(wp), dimension(:), allocatable `mo_data::fl_sw_input`  
Used to read in sw fluxes from ERA for `atmoflux_flag==2`.
- real(wp), dimension(:), allocatable `mo_data::fl_lw_input`  
Used to read in lw fluxes from ERA for `atmoflux_flag==2`.
- real(wp), dimension(:), allocatable `mo_data::t2m_input`  
Used to read in 2Tm from ERA for `atmoflux_flag==2`.
- real(wp), dimension(:), allocatable `mo_data::precip_input`  
Used to read in precipitation from ERA for `atmoflux_flag==2`.
- real(wp), dimension(:), allocatable `mo_data::time_input`

- Used to read in time from ERA for atmoflux\_flag==2.*
- integer `mo_data::time_counter`  
*Keeps track of input data.*
  - integer `mo_data::bgc_flag`  
*1: no bgc, 2: bgc*
  - integer `mo_data::n_bgc`  
*Number of chemicals.*
  - real(wp), dimension(:,:), allocatable `mo_data::fl_brine_bgc`  
*Brine fluxes in a matrix, [kg/s], first index is the layer of origin, and the second index is the layer of arrival.*
  - real(wp), dimension(:,:), allocatable `mo_data::bgc_abs`  
*Absolute amount of chemicals [kmol] for each tracer.*
  - real(wp), dimension(:,:), allocatable `mo_data::bgc_bu`  
*Bulk amounts of chemicals [kmol/kg].*
  - real(wp), dimension(:,:), allocatable `mo_data::bgc_br`  
*Brine concentrations of chems [kmol/kg].*
  - real(wp), dimension(:), allocatable `mo_data::bgc_bottom`  
*Bulk concentrations of chems below the ice [kmol/kg].*
  - real(wp), dimension(:), allocatable `mo_data::bgc_total`  
*Total of chems, for lab experiments with a fixed total amount.*
  - real(wp) `mo_data::m_total`  
*Total initial water mass, for lab experiments with a fixed total amount.*
  - real(wp) `mo_data::s_total`  
*Total initial salt mass, for lab experiments with a fixed total amount.*
  - real(wp) `mo_data::tank_depth`  
*water depth in meters, used to calculate concentrations below ice for tank experiments*
  - character \*3 `mo_data::flush_question` = 'No!'  
*Niels, 2017 add: used to indicate in stdout whether flushing occurs at this moment or not.*
  - real(wp) `mo_data::melt_err` = 0. \_wp  
*Niels, 2017 add: used to check how much meltwater vanishes in flushing routine.*
  - integer `mo_data::length_input_lab`  
*Niels, 2017 add: used to allocate lab testcase input arrays in `mo_init`, set value in testcases.*

## 5.3 mo\_flood.f90 File Reference

### Modules

- module `mo_flood`  
*Computes the fluxes caused by liquid flooding the snow layer.*

### Functions/Subroutines

- subroutine, public `mo_flood::flood` (freeboard, psi\_s, psi\_l, S\_abs, H\_abs, m, T, thick, dt, Nlayer, N\_active, T\_bottom, S\_bu\_bottom, H\_abs\_snow, m\_snow, thick\_snow, psi\_g\_snow, debug\_flag, fl\_brine\_bgc)  
*Subroutine for calculating flooding.*
- subroutine, public `mo_flood::flood_simple` (freeboard, S\_abs, H\_abs, m, thick, T\_bottom, S\_bu\_bottom, H\_abs\_snow, m\_snow, thick\_snow, psi\_g\_snow, Nlayer, N\_active, debug\_flag)  
*Subroutine for calculating flooding.*

## 5.4 mo\_flush.f90 File Reference

### Modules

- module [mo\\_flush](#)  
*Contains various subroutines for flushing.*

### Functions/Subroutines

- subroutine, public [mo\\_flush::flush3](#) (freeboard, psi\_l, thick, thick\_0, S\_abs, H\_abs, m, T, dt, Nlayer, N\_active, T\_bottom, S\_bu\_bottom, melt\_thick, debug\_flag, flush\_heat\_flag, melt\_err, perm, flush\_v, flush\_h, psi\_g, thick\_snow, rho\_l, snow\_flush\_flag, fl\_brine\_bgc)  
*Subroutine for complex flushing.*
- subroutine, public [mo\\_flush::flush4](#) (psi\_l, thick, T, thick\_0, S\_abs, H\_abs, m, dt, Nlayer, N\_active, N\_top, N\_middle, N\_bottom, melt\_thick, debug\_flag)  
*An alternative subroutine for calculating flushing.*

## 5.5 mo\_functions.f90 File Reference

### Modules

- module [mo\\_functions](#)  
*Module houses functions which have no home :(.*

### Functions/Subroutines

- real(wp) function [mo\\_functions::func\\_density](#) (T, S)  
*Calculates the physical density for given S and T.*
- real(wp) function [mo\\_functions::func\\_freeboard](#) (N\_active, Nlayer, psi\_s, psi\_g, m, thick, m\_snow, freeboard\_snow\_flag)  
*Calculates the freeboard of the 1d ice column.*
- real(wp) function [mo\\_functions::func\\_albedo](#) (thick\_snow, T\_snow, psi\_l, thick\_min, albedo\_flag)  
*Calculates the albedo.*
- real(wp) function [mo\\_functions::func\\_sat\\_o2](#) (T, S\_bu)  
*Calculates the oxygen saturation as a function of salinity and temperature.*
- real(wp) function [mo\\_functions::func\\_t\\_freeze](#) (S\_bu, salt\_flag)  
*Calculates the freezing temperature. Salt\_flag determines if either ocean salt or NaCl is used.*
- subroutine [mo\\_functions::sub\\_notzflux](#) (time, fl\_sw, fl\_rest)  
*Calculates the incoming shortwave and other fluxes according to p. 193-194 PhD Notz.*
- subroutine [mo\\_functions::sub\\_input](#) (length\_input, fl\_sw\_input, fl\_lw\_input, T2m\_input, precip\_input, time↔input)  
*Reads in data for atmoflux\_flag ==2.*
- subroutine [mo\\_functions::sub\\_turb\\_flux](#) (T\_bottom, S\_bu\_bottom, T, S\_abs, m, dt, N\_bgc, bgc\_bottom, bgc↔\_abs)  
*Calculates salt and tracer mixing between lowest layer and underlying water.*
- subroutine [mo\\_functions::sub\\_melt\\_thick](#) (psi\_l, psi\_s, psi\_g, T, T\_freeze, T\_top, fl\_Q, thick\_snow, dt, melt↔\_thick, thick, thick\_min)  
*Calculates the thickness of the meltwater film.*
- subroutine [mo\\_functions::sub\\_melt\\_snow](#) (melt\_thick, thick, thick\_snow, H\_abs, H\_abs\_snow, m, m\_snow, psi\_g\_snow)  
*Calculates how the meltwater film interacts with snow.*



## 5.6 mo\_grav\_drain.f90 File Reference

### Modules

- module [mo\\_grav\\_drain](#)  
*Computes the Salt fluxes caused by gravity drainage.*

### Functions/Subroutines

- subroutine, public [mo\\_grav\\_drain::fl\\_grav\\_drain](#) (S\_br, S\_bu, psi\_l, psi\_s, psi\_g, thick, S\_abs, H\_abs, T, m, dt, Nlayer, N\_active, ray, T\_bottom, S\_bu\_bottom, grav\_drain, grav\_temp, grav\_salt, grav\_heat\_flag, harmonic\_flag, fl\_brine\_bgc)  
*Calculates fluxes caused by gravity drainage.*
- subroutine, public [mo\\_grav\\_drain::fl\\_grav\\_drain\\_simple](#) (psi\_s, psi\_l, thick, S\_abs, S\_br, Nlayer, N\_active, ray, grav\_drain, harmonic\_flag)  
*Calculates salinity to imitate the effects gravity drainage.*

## 5.7 mo\_grotz.f90 File Reference

### Modules

- module [mo\\_grotz](#)  
*The most important module of SAMSIM.*

### Functions/Subroutines

- subroutine [mo\\_grotz::grotz](#) (testcase, description)  
*Main subroutine of SAMSIM, a 1D thermodynamic seaice model. A semi-adaptive grid is used which is managed by [mo\\_layer\\_dynamics](#).*

## 5.8 mo\_heat\_fluxes.f90 File Reference

### Modules

- module [mo\\_heat\\_fluxes](#)  
*Computes all heat fluxes.*

### Functions/Subroutines

- subroutine [mo\\_heat\\_fluxes::sub\\_heat\\_fluxes](#) ()  
*Computes surface temperature and heatfluxes.*

## 5.9 mo\_init.f90 File Reference

### Modules

- module [mo\\_init](#)  
*Allocates Arrays and sets initial data for a given testcase for SAMSIM.*

### Functions/Subroutines

- subroutine [mo\\_init::init](#) (testcase)  
*Sets initial conditions according to which testcase is chosen.*
- subroutine [mo\\_init::sub\\_allocate](#) (Nlayer, length\_input\_lab)  
*Allocates Arrays.*
- subroutine [mo\\_init::sub\\_allocate\\_bgc](#) (Nlayer, N\_bgc)  
*Allocates BGC Arrays.*
- subroutine [mo\\_init::sub\\_deallocate](#)  
*Deallocates Arrays.*

## 5.10 mo\_layer\_dynamics.f90 File Reference

### Modules

- module [mo\\_layer\\_dynamics](#)  
*Mo\_layer\_dynamics contains all subroutines for the growth and shrinking of layer thickness.*

### Functions/Subroutines

- subroutine, public [mo\\_layer\\_dynamics::layer\\_dynamics](#) (phi, N\_active, Nlayer, N\_bottom, N\_middle, N\_top, m, S\_abs, H\_abs, thick, thick\_0, T\_bottom, S\_bu\_bottom, bottom\_flag, debug\_flag, melt\_thick\_output, N\_bgc, bgc\_abs, bgc\_bottom)  
*Organizes the Semi-Adaptive grid SAMSIM uses.*
- subroutine, public [mo\\_layer\\_dynamics::top\\_melt](#) (Nlayer, N\_active, N\_bottom, N\_middle, N\_top, thick\_0, m, S\_abs, H\_abs, thick, N\_bgc, bgc\_abs)
- subroutine, public [mo\\_layer\\_dynamics::top\\_grow](#) (Nlayer, N\_active, N\_bottom, N\_middle, N\_top, thick\_0, m, S\_abs, H\_abs, thick, N\_bgc, bgc\_abs)  
*Top grow subroutine.*

## 5.11 mo\_mass.f90 File Reference

### Modules

- module [mo\\_mass](#)  
*Regulates mass transfers and their results.*

## Functions/Subroutines

- subroutine, public [mo\\_mass::mass\\_transfer](#) (Nlayer, N\_active, T, H\_abs, S\_abs, S\_bu, T\_bottom, S\_bu\_bottom, fl\_m)  
*Calculates the effects of mass transfers on H\_abs and S\_abs.*
- subroutine, public [mo\\_mass::expulsion\\_flux](#) (thick, V\_ex, Nlayer, N\_active, psi\_g, fl\_m, m)  
*Generates the fluxes caused by expulsion.*
- subroutine, public [mo\\_mass::bgc\\_advection](#) (Nlayer, N\_active, N\_bgc, fl\_brine\_bgc, bgc\_abs, psi\_l, T, S\_abs, m, thick, bgc\_bottom)  
*Calculates how the brine fluxes stored in fl\_brine\_bgc advect bgc tracers.*

## 5.12 mo\_output.f90 File Reference

### Modules

- module [mo\\_output](#)  
*All things output.*

### Functions/Subroutines

- subroutine, public [mo\\_output::output\\_settings](#) (description, testcase, N\_top, N\_bottom, Nlayer, fl\_q\_bottom, T\_bottom, S\_bu\_bottom, thick\_0, time\_out, time\_total, dt, boundflux\_flag, atmoflux\_flag, albedo\_flag, grav\_flag, flush\_flag, flood\_flag, grav\_heat\_flag, flush\_heat\_flag, harmonic\_flag, prescribe\_flag, salt\_flag, turb\_flag, bottom\_flag, tank\_flag, precip\_flag, bgc\_flag, N\_bgc, k\_snow\_flush)  
*Settings output.*
- subroutine, public [mo\\_output::output](#) (Nlayer, T, psi\_s, psi\_l, thick, S\_bu, ray, format\_T, format\_psi, format\_thick, format\_snow, freeboard, thick\_snow, T\_snow, psi\_l\_snow, psi\_s\_snow, energy\_stored, freshwater, total\_resist, thickness, bulk\_salin, grav\_drain, grav\_salt, grav\_temp, T2m, T\_top, perm, format\_perm, flush\_v, flush\_h, psi\_g, melt\_thick\_output, format\_melt)  
*Standard output.*
- subroutine, public [mo\\_output::output\\_bgc](#) (Nlayer, N\_active, bgc\_bottom, N\_bgc, bgc\_abs, psi\_l, thick, m, format\_bgc)  
*Standard bgc output.*
- subroutine, public [mo\\_output::output\\_raw](#) (Nlayer, N\_active, time, T, thick, S\_bu, psi\_s, psi\_l, psi\_g)  
*Output for debugging purposes.*
- subroutine, public [mo\\_output::output\\_raw\\_snow](#) (time, T\_snow, thick\_snow, S\_abs\_snow, m\_snow, psi\_s\_snow, psi\_l\_snow, psi\_g\_snow)  
*Output for debugging purposes.*
- subroutine, public [mo\\_output::output\\_raw\\_layer](#) (Nlayer, N\_active, H\_abs, m, S\_abs, thick, string)  
*Output for debugging layer dynamics..*
- subroutine, public [mo\\_output::output\\_begin](#) (Nlayer, debug\_flag, format\_T, format\_psi, format\_thick, format\_snow, format\_T2m\_top, format\_perm, format\_melt)  
*Output files are opened and format strings are created.*
- subroutine, public [mo\\_output::output\\_begin\\_bgc](#) (Nlayer, N\_bgc, format\_bgc)  
*Output files for bgc are opened and format strings are created.*

## 5.13 mo\_parameters.f90 File Reference

### Modules

- module `mo_parameters`

*Module determines physical constants to be used by the SAMSIM Seaice model.*

### Variables

- integer, parameter `mo_parameters::wp` = `SELECTED_REAL_KIND(12, 307)`  
*set working precision\_wp*
- real, parameter `mo_parameters::pi` = 3.1415\_wp
- real, parameter `mo_parameters::grav` = 9.8061\_wp  
*gravitational constant [m/s<sup>2</sup>]*
- real(wp), parameter `mo_parameters::k_s` = 2.2\_wp  
*solid heat conductivity [J / m s K] 2.2*
- real(wp), parameter `mo_parameters::k_l` = 0.523\_wp  
*liquid heat conductivity [J / m s K] 0.523*
- real(wp), parameter `mo_parameters::c_s` = 2020.0\_wp  
*solid heat capacity [J/ kg K]*
- real(wp), parameter `mo_parameters::c_s_beta` = 7.6973\_wp  
*linear solid heat capacity approximation [J/ kg K<sup>2</sup>]  $c_s = c_s + c_s\_beta * T$*
- real(wp), parameter `mo_parameters::c_l` = 3400.\_wp  
*liquid heat capacity [J/ kg K]*
- real(wp), parameter `mo_parameters::rho_s` = 920.\_wp  
*density of solid [kg / m<sup>3</sup>]*
- real(wp), parameter `mo_parameters::rho_l` = 1028.0\_wp  
*density of liquid [kg / m<sup>3</sup>]*
- real(wp), parameter `mo_parameters::latent_heat` = 333500.\_wp  
*latent heat release [J/kg]*
- real(wp), parameter `mo_parameters::zerok` = 273.15\_wp  
*Zero degrees Celsius in Kelvin [K].*
- real(wp), parameter `mo_parameters::bbeta` = 0.8\_wp\*1e-3  
*concentration expansion coefficient [kg / (m<sup>3</sup> ppt)]*
- real(wp), parameter `mo_parameters::mu` = 2.55\_wp\*1e-3  
*dynamic viscosity [kg / m s]*
- real(wp), parameter `mo_parameters::kappa_l` = `k_l/rho_l/c_l`  
*heat diffusivity of water*
- real(wp), parameter `mo_parameters::sigma` = 5.6704\_wp\*1e-8  
*Stefan Boltzmann constant [W/(m<sup>2</sup>\*K<sup>4</sup>)].*
- real(wp), parameter `mo_parameters::psi_s_min` = 0.05\_wp  
*The amount of ice that the lowest layer can have before it counts as an ice layer.*
- real(wp), parameter `mo_parameters::neg_free` = -0.05\_wp  
*The distance the freeboard can be below 0 before water starts flooding through cracks.*
- real(wp), parameter `mo_parameters::x_grav` = 0.000584\_wp
- real(wp), parameter `mo_parameters::ray_crit` = 4.89\_wp
- real(wp), parameter `mo_parameters::para_flush_horiz` = 1.0\_wp  
*determines relationship of horizontal flow distance in during flushing (guess 1)*
- real(wp), parameter `mo_parameters::para_flush_gamma` = 0.9\_wp  
*Strength of desalination per timestep (guess)*

- real(wp), parameter `mo_parameters::psi_s_top_min` = 0.40\_wp  
*if psi\_s is below this value meltwater forms (guess) 0.4*
- real(wp), parameter `mo_parameters::ratio_flood` = 1.50\_wp  
*Ratio of flooded to dissolve snow, plays an important role in subroutine flood.*
- real(wp), parameter `mo_parameters::ref_salinity` = 34.\_wp  
*Reference salinity [g/kg] used to calculate freshwater column.*
- real(wp), parameter `mo_parameters::rho_snow` = 330.\_wp  
*density of new snow [kg/m\*\*3], I< Niels, 2017 add: can be adjusted to lab values if they are measured*
- real(wp), parameter `mo_parameters::gas_snow_ice` = 0.10\_wp  
*volume of gas percentage in new snow ice due to flooding, no longer used*
- real(wp), parameter `mo_parameters::gas_snow_ice2` = 0.20\_wp  
*volume of gas percentage in new snow ice due to snow melting (Eicken 95)*
- real(wp), parameter `mo_parameters::emissivity_ice` = 0.95\_wp  
*Emissivity of water and ice.*
- real(wp), parameter `mo_parameters::emissivity_snow` = 1.00\_wp  
*Emissivity of Snow.*
- real(wp), parameter `mo_parameters::penetr` = 0.30\_wp  
*Amount of penetrating sw radiation.*
- real(wp), parameter `mo_parameters::extinc` = 2.00\_wp  
*Extinction coefficient of ice.*
- real(wp), parameter `mo_parameters::turb_a` = 0.1\_wp\*0.05\_wp\*rho\_l/86400.\_wp  
*Standard turbulence [kg/s] WARNING no source, just set so that 5cm of water are overturned each day.*
- real(wp), parameter `mo_parameters::turb_b` = 0.05\_wp  
*Exponential turbulence slope [m\*\*3/kg] WARNING no source, simple guess.*
- real(wp) `mo_parameters::max_flux_plate` = 50.0  
*Maximal heating rate of a heating plate.*
- real(wp) `mo_parameters::k_snow_flush` = 0.75\_wp  
*Niels, 2017 add: Percentage of excess liquid water content in the snow that is used for flushing instead of forming slush.*
- real(wp) `mo_parameters::k_styropor` = 0.8\_wp  
*Niels, 2017 add: heat conduction of styropor (empirical value to fit measurement data)*

## 5.14 mo\_snow.f90 File Reference

### Modules

- module `mo_snow`  
*Module contains all things directly related to snow.*

### Functions/Subroutines

- subroutine, public `mo_snow::snow_coupling` (H\_abs\_snow, phi\_s, T\_snow, H\_abs, H, phi, T, m\_snow, S\_↔  
abs\_snow, m, S\_bu)  
*Subroutine to couple a thin snow layer to the upper ice layer.*
- subroutine, public `mo_snow::snow_precip` (m\_snow, H\_abs\_snow, thick\_snow, psi\_s\_snow, dt, liquid\_↔  
precip\_in, T2m, solid\_precip\_in)  
*Subroutine for calculating precipitation on an existing snow cover.*
- subroutine, public `mo_snow::snow_precip_0` (H\_abs, S\_abs, m, T, dt, liquid\_precip\_in, T2m, solid\_precip\_in)  
*Subroutine for calculating precipitation into the ocean.*

- subroutine, public [mo\\_snow::snow\\_thermo](#) (psi\_l\_snow, psi\_s\_snow, psi\_g\_snow, thick\_snow, S\_abs\_snow, H\_abs\_snow, m\_snow, T\_snow, m, thick, H\_abs)  
*Subroutine for calculating snow thermodynamics.*
- subroutine, public [mo\\_snow::snow\\_thermo\\_meltwater](#) (psi\_l\_snow, psi\_s\_snow, psi\_g\_snow, thick\_snow, S\_abs\_snow, H\_abs\_snow, m\_snow, T\_snow, m, thick, H\_abs, melt\_thick\_snow)  
*Subroutine for calculating snow thermodynamics.*
- subroutine, public [mo\\_snow::sub\\_fl\\_q\\_0\\_snow\\_thin](#) (m\_snow, thick\_snow, T\_snow, psi\_s, psi\_l, psi\_g, thick, T\_bound, fl\_Q\_snow)  
*Determines conductive Heat flux for combined top ice and snow layer.*
- subroutine, public [mo\\_snow::sub\\_fl\\_q\\_snow](#) (m\_snow, thick\_snow, T\_snow, psi\_s\_2, psi\_l\_2, psi\_g\_2, thick\_2, T\_2, fl\_Q)  
*Determines conductive Heat flux between Snow and top ice layer.*
- subroutine, public [mo\\_snow::sub\\_fl\\_q\\_0\\_snow](#) (m\_snow, thick\_snow, T\_snow, T\_bound, fl\_Q)  
*Determines conductive Heat between snow layer and upper boundary layer. A limiting factor is added to increase stability of layers thinner than thick\_min.*
- real(wp) function, public [mo\\_snow::func\\_k\\_snow](#) (m\_snow, thick\_snow)  
*Calculates the thermal conductivity of the snow layer as a function of the density.*

## 5.15 mo\_testcase\_specifics.f90 File Reference

### Modules

- module [mo\\_testcase\\_specifics](#)  
*Module contains changes specific testcases require during the main timeloop.*

### Functions/Subroutines

- subroutine, public [mo\\_testcase\\_specifics::sub\\_test1](#) (time, T\_top)  
*Subroutine for changing T\_top for testcase 1.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test2](#) (time, T2m)  
*Subroutine for changing T\_top for testcase 2.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test9](#) (time, T2m)  
*Subroutine for changing T2m for testcase 9.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test34](#) (time, T2m)  
*Subroutine for changing T2m for testcase 34.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test3](#) (time, liquid\_precip, solid\_precip)  
*Subroutine for setting snow for testcase 3.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test4](#) (time, fl\_q\_bottom)  
*Subroutine for setting snow for testcase 4.*
- subroutine, public [mo\\_testcase\\_specifics::sub\\_test6](#) (time, T2m)  
*Subroutine for changing T\_top for testcase 6 which seeks to reproduce lab measurements of Roni Glud.*

## 5.16 mo\_thermo\_functions.f90 File Reference

### Modules

- module [mo\\_thermo\\_functions](#)  
*Contains subroutines and functions related to multi-phase thermodynamics.*

## Functions/Subroutines

- subroutine, public [mo\\_thermo\\_functions::gett](#) (H, S<sub>bu</sub>, T<sub>in</sub>, T, phi, k)  
*Determines equilibrium Temperature of a layer for given S<sub>bu</sub> and H as well as solid fraction.*
- subroutine, public [mo\\_thermo\\_functions::expulsion](#) (phi, thick, m, psi<sub>s</sub>, psi<sub>l</sub>, psi<sub>g</sub>, V<sub>ex</sub>)  
*Determines Brine flux expelled from out of a layer due to freezing.*
- subroutine, public [mo\\_thermo\\_functions::sub\\_fl\\_q](#) (psi<sub>s</sub><sub>1</sub>, psi<sub>l</sub><sub>1</sub>, psi<sub>g</sub><sub>1</sub>, thick<sub>1</sub>, T<sub>1</sub>, psi<sub>s</sub><sub>2</sub>, psi<sub>l</sub><sub>2</sub>, psi<sub>g</sub><sub>2</sub>, thick<sub>2</sub>, T<sub>2</sub>, fl\_Q)  
*Determines conductive heat flux between two layers.*
- subroutine, public [mo\\_thermo\\_functions::sub\\_fl\\_q\\_0](#) (psi<sub>s</sub>, psi<sub>l</sub>, psi<sub>g</sub>, thick, T, T<sub>bound</sub>, direct\_flag, fl\_Q)  
*Determines conductive Heat flux between layer and boundary temperatures.*
- subroutine, public [mo\\_thermo\\_functions::sub\\_fl\\_q\\_styropor](#) (k<sub>styropor</sub>, fl\_Q)  
*Niels, 2017 add: Determines conductive Heat flux below styropor cover.*
- real(wp) function, public [mo\\_thermo\\_functions::func\\_s\\_br](#) (T, S<sub>bu</sub>)  
*Computes salinity of brine pockets for given temperature in Celsius of mushy layer.*
- real(wp) function, public [mo\\_thermo\\_functions::func\\_ddt\\_s\\_br](#) (T)  
*Computes temperature derivative of brine pocket salinity for given temperature in Celsius of mushy layer.*

## 5.17 SAMSIM.f90 File Reference

### Functions/Subroutines

- program [samsim](#)

### 5.17.1 Function/Subroutine Documentation

#### 5.17.1.1 program samsim ( )





# Index

albedo  
    mo\_data, [13](#)  
albedo\_flag  
    mo\_data, [13](#)  
alpha\_flux\_instable  
    mo\_data, [13](#)  
alpha\_flux\_stable  
    mo\_data, [13](#)  
atmoflux\_flag  
    mo\_data, [13](#)  
  
bbeta  
    mo\_parameters, [50](#)  
bgc\_abs  
    mo\_data, [13](#)  
bgc\_advection  
    mo\_mass, [43](#)  
bgc\_bottom  
    mo\_data, [13](#)  
bgc\_br  
    mo\_data, [13](#)  
bgc\_bu  
    mo\_data, [14](#)  
bgc\_flag  
    mo\_data, [14](#)  
bgc\_total  
    mo\_data, [14](#)  
bottom\_flag  
    mo\_data, [14](#)  
boundflux\_flag  
    mo\_data, [14](#)  
bulk\_salin  
    mo\_data, [14](#)  
  
c\_l  
    mo\_parameters, [50](#)  
c\_s  
    mo\_parameters, [50](#)  
c\_s\_beta  
    mo\_parameters, [50](#)  
  
debug\_flag  
    mo\_data, [14](#)  
dt  
    mo\_data, [14](#)  
  
emissivity\_ice  
    mo\_parameters, [50](#)  
emissivity\_snow  
    mo\_parameters, [50](#)  
  
energy\_stored  
    mo\_data, [14](#)  
expulsion  
    mo\_thermo\_functions, [61](#)  
expulsion\_flux  
    mo\_mass, [43](#)  
extinc  
    mo\_parameters, [50](#)  
  
fl\_brine\_bgc  
    mo\_data, [14](#)  
fl\_grav\_drain  
    mo\_grav\_drain, [34](#)  
fl\_grav\_drain\_simple  
    mo\_grav\_drain, [35](#)  
fl\_lat  
    mo\_data, [15](#)  
fl\_lw  
    mo\_data, [15](#)  
fl\_lw\_input  
    mo\_data, [15](#)  
fl\_m  
    mo\_data, [15](#)  
fl\_q  
    mo\_data, [15](#)  
fl\_q\_bottom  
    mo\_data, [15](#)  
fl\_q\_snow  
    mo\_data, [15](#)  
fl\_rad  
    mo\_data, [15](#)  
fl\_rest  
    mo\_data, [15](#)  
fl\_s  
    mo\_data, [15](#)  
fl\_sen  
    mo\_data, [16](#)  
fl\_sw  
    mo\_data, [16](#)  
fl\_sw\_input  
    mo\_data, [16](#)  
flood  
    mo\_flood, [27](#)  
flood\_flag  
    mo\_data, [16](#)  
flood\_simple  
    mo\_flood, [27](#)  
flush3  
    mo\_flush, [28](#)  
flush4

- mo\_flush, 29
- flush\_flag
  - mo\_data, 16
- flush\_h
  - mo\_data, 16
- flush\_h\_old
  - mo\_data, 16
- flush\_heat\_flag
  - mo\_data, 16
- flush\_question
  - mo\_data, 16
- flush\_v
  - mo\_data, 16
- flush\_v\_old
  - mo\_data, 17
- format\_bgc
  - mo\_data, 17
- format\_integer
  - mo\_data, 17
- format\_melt
  - mo\_data, 17
- format\_perm
  - mo\_data, 17
- format\_psi
  - mo\_data, 17
- format\_snow
  - mo\_data, 17
- format\_t
  - mo\_data, 17
- format\_t2m\_top
  - mo\_data, 17
- format\_thick
  - mo\_data, 17
- freeboard
  - mo\_data, 17
- freeboard\_snow\_flag
  - mo\_data, 17
- freshwater
  - mo\_data, 17
- func\_albedo
  - mo\_functions, 30
- func\_ddt\_s\_br
  - mo\_thermo\_functions, 61
- func\_density
  - mo\_functions, 30
- func\_freeboard
  - mo\_functions, 31
- func\_k\_snow
  - mo\_snow, 55
- func\_s\_br
  - mo\_thermo\_functions, 61
- func\_sat\_o2
  - mo\_functions, 31
- func\_t\_freeze
  - mo\_functions, 31
- gas\_snow\_ice
  - mo\_parameters, 51
- gas\_snow\_ice2
  - mo\_parameters, 51
- gett
  - mo\_thermo\_functions, 62
- grav
  - mo\_parameters, 51
- grav\_drain
  - mo\_data, 17
- grav\_flag
  - mo\_data, 17
- grav\_heat\_flag
  - mo\_data, 18
- grav\_salt
  - mo\_data, 18
- grav\_temp
  - mo\_data, 18
- grotz
  - mo\_grotz, 36
- h
  - mo\_data, 18
- h\_abs
  - mo\_data, 18
- h\_abs\_snow
  - mo\_data, 18
- harmonic\_flag
  - mo\_data, 18
- i
  - mo\_data, 18
- i\_time
  - mo\_data, 18
- i\_time\_out
  - mo\_data, 18
- init
  - mo\_init, 39
- k
  - mo\_data, 19
- k\_l
  - mo\_parameters, 51
- k\_s
  - mo\_parameters, 51
- k\_snow\_flush
  - mo\_parameters, 51
- k\_styropor
  - mo\_parameters, 51
- kappa\_l
  - mo\_parameters, 51
- lab\_snow\_flag
  - mo\_data, 19
- latent\_heat
  - mo\_parameters, 51
- layer\_dynamics
  - mo\_layer\_dynamics, 41
- length\_input
  - mo\_data, 19
- length\_input\_lab
  - mo\_data, 19

- liquid\_precip
  - mo\_data, 19
- m
  - mo\_data, 19
- m\_snow
  - mo\_data, 19
- m\_total
  - mo\_data, 19
- mass\_transfer
  - mo\_mass, 44
- max\_flux\_plate
  - mo\_parameters, 51
- melt\_err
  - mo\_data, 19
- melt\_thick
  - mo\_data, 19
- melt\_thick\_output
  - mo\_data, 20
- melt\_thick\_snow
  - mo\_data, 20
- melt\_thick\_snow\_old
  - mo\_data, 20
- mo\_data, 7
  - albedo, 13
  - albedo\_flag, 13
  - alpha\_flux\_instable, 13
  - alpha\_flux\_stable, 13
  - atmoflux\_flag, 13
  - bgc\_abs, 13
  - bgc\_bottom, 13
  - bgc\_br, 13
  - bgc\_bu, 14
  - bgc\_flag, 14
  - bgc\_total, 14
  - bottom\_flag, 14
  - boundflux\_flag, 14
  - bulk\_salin, 14
  - debug\_flag, 14
  - dt, 14
  - energy\_stored, 14
  - fl\_brine\_bgc, 14
  - fl\_lat, 15
  - fl\_lw, 15
  - fl\_lw\_input, 15
  - fl\_m, 15
  - fl\_q, 15
  - fl\_q\_bottom, 15
  - fl\_q\_snow, 15
  - fl\_rad, 15
  - fl\_rest, 15
  - fl\_s, 15
  - fl\_sen, 16
  - fl\_sw, 16
  - fl\_sw\_input, 16
  - flood\_flag, 16
  - flush\_flag, 16
  - flush\_h, 16
  - flush\_h\_old, 16
  - flush\_heat\_flag, 16
  - flush\_question, 16
  - flush\_v, 16
  - flush\_v\_old, 17
  - format\_bgc, 17
  - format\_integer, 17
  - format\_melt, 17
  - format\_perm, 17
  - format\_psi, 17
  - format\_snow, 17
  - format\_t, 17
  - format\_t2m\_top, 17
  - format\_thick, 17
  - freeboard, 17
  - freeboard\_snow\_flag, 17
  - freshwater, 17
  - grav\_drain, 17
  - grav\_flag, 17
  - grav\_heat\_flag, 18
  - grav\_salt, 18
  - grav\_temp, 18
  - h, 18
  - h\_abs, 18
  - h\_abs\_snow, 18
  - harmonic\_flag, 18
  - i, 18
  - i\_time, 18
  - i\_time\_out, 18
  - k, 19
  - lab\_snow\_flag, 19
  - length\_input, 19
  - length\_input\_lab, 19
  - liquid\_precip, 19
  - m, 19
  - m\_snow, 19
  - m\_total, 19
  - melt\_err, 19
  - melt\_thick, 19
  - melt\_thick\_output, 20
  - melt\_thick\_snow, 20
  - melt\_thick\_snow\_old, 20
  - n\_active, 20
  - n\_bgc, 20
  - n\_bottom, 20
  - n\_middle, 20
  - n\_time\_out, 20
  - n\_top, 20
  - nlayer, 20
  - ocean\_flux\_input, 20
  - ocean\_t\_input, 21
  - perm, 21
  - phi, 21
  - phi\_s, 21
  - precip\_flag, 21
  - precip\_input, 21
  - precipinput, 21
  - prescribe\_flag, 21
  - psi\_g, 21

- psi\_g\_snow, 21
- psi\_l, 22
- psi\_l\_snow, 22
- psi\_s, 22
- psi\_s\_snow, 22
- q, 22
- ray, 22
- s\_abs, 22
- s\_abs\_snow, 22
- s\_br, 22
- s\_bu, 22
- s\_bu\_bottom, 23
- s\_total, 23
- salt\_flag, 23
- snow\_flush\_flag, 23
- snow\_precip\_flag, 23
- solid\_precip, 23
- styropor\_flag, 23
- styropor\_input, 23
- surface\_water, 23
- t, 23
- t2m, 23
- t2m\_input, 24
- t\_bottom, 24
- t\_freeze, 24
- t\_snow, 24
- t\_test, 24
- t\_top, 24
- tank\_depth, 24
- tank\_flag, 24
- test, 24
- thick, 24
- thick\_0, 25
- thick\_min, 25
- thick\_snow, 25
- thickness, 25
- time, 25
- time\_counter, 25
- time\_input, 25
- time\_out, 25
- time\_total, 25
- tinput, 25
- total\_resist, 25
- ttop\_input, 26
- turb\_flag, 26
- v\_ex, 26
- v\_g, 26
- v\_l, 26
- v\_s, 26
- mo\_data.f90, 65
- mo\_flood, 26
  - flood, 27
  - flood\_simple, 27
- mo\_flood.f90, 70
- mo\_flush, 28
  - flush3, 28
  - flush4, 29
- mo\_flush.f90, 71
- mo\_functions, 29
  - func\_albedo, 30
  - func\_density, 30
  - func\_freeboard, 31
  - func\_sat\_o2, 31
  - func\_t\_freeze, 31
  - sub\_input, 31
  - sub\_melt\_snow, 32
  - sub\_melt\_thick, 32
  - sub\_notzflux, 32
  - sub\_turb\_flux, 33
- mo\_functions.f90, 71
- mo\_grav\_drain, 33
  - fl\_grav\_drain, 34
  - fl\_grav\_drain\_simple, 35
- mo\_grav\_drain.f90, 72
- mo\_grotz, 35
  - grotz, 36
- mo\_grotz.f90, 72
- mo\_heat\_fluxes, 36
  - sub\_heat\_fluxes, 37
- mo\_heat\_fluxes.f90, 73
- mo\_init, 38
  - init, 39
  - sub\_allocate, 40
  - sub\_allocate\_bgc, 40
  - sub\_deallocate, 40
- mo\_init.f90, 73
- mo\_layer\_dynamics, 41
  - layer\_dynamics, 41
  - top\_grow, 42
  - top\_melt, 42
- mo\_layer\_dynamics.f90, 73
- mo\_mass, 43
  - bgc\_advection, 43
  - expulsion\_flux, 43
  - mass\_transfer, 44
- mo\_mass.f90, 74
- mo\_output, 44
  - output, 46
  - output\_begin, 46
  - output\_begin\_bgc, 46
  - output\_bgc, 47
  - output\_raw, 47
  - output\_raw\_lay, 47
  - output\_raw\_snow, 47
  - output\_settings, 47
- mo\_output.f90, 74
- mo\_parameters, 48
  - bbeta, 50
  - c\_l, 50
  - c\_s, 50
  - c\_s\_beta, 50
  - emissivity\_ice, 50
  - emissivity\_snow, 50
  - extinc, 50
  - gas\_snow\_ice, 51
  - gas\_snow\_ice2, 51

- grav, [51](#)
- k\_l, [51](#)
- k\_s, [51](#)
- k\_snow\_flush, [51](#)
- k\_styropor, [51](#)
- kappa\_l, [51](#)
- latent\_heat, [51](#)
- max\_flux\_plate, [51](#)
- mu, [52](#)
- neg\_free, [52](#)
- para\_flush\_gamma, [52](#)
- para\_flush\_horiz, [52](#)
- penetr, [52](#)
- pi, [52](#)
- psi\_s\_min, [52](#)
- psi\_s\_top\_min, [52](#)
- ratio\_flood, [52](#)
- ray\_crit, [52](#)
- ref\_salinity, [52](#)
- rho\_l, [52](#)
- rho\_s, [53](#)
- rho\_snow, [53](#)
- sigma, [53](#)
- turb\_a, [53](#)
- turb\_b, [53](#)
- wp, [53](#)
- x\_grav, [53](#)
- zerok, [53](#)
- mo\_parameters.f90, [75](#)
- mo\_snow, [53](#)
  - func\_k\_snow, [55](#)
  - snow\_coupling, [55](#)
  - snow\_precip, [55](#)
  - snow\_precip\_0, [55](#)
  - snow\_thermo, [56](#)
  - snow\_thermo\_meltwater, [56](#)
  - sub\_fl\_q\_0\_snow, [57](#)
  - sub\_fl\_q\_0\_snow\_thin, [57](#)
  - sub\_fl\_q\_snow, [57](#)
- mo\_snow.f90, [76](#)
- mo\_testcase\_specifics, [58](#)
  - sub\_test1, [58](#)
  - sub\_test2, [58](#)
  - sub\_test3, [59](#)
  - sub\_test34, [59](#)
  - sub\_test4, [59](#)
  - sub\_test6, [59](#)
  - sub\_test9, [59](#)
- mo\_testcase\_specifics.f90, [77](#)
- mo\_thermo\_functions, [60](#)
  - expulsion, [61](#)
  - func\_ddt\_s\_br, [61](#)
  - func\_s\_br, [61](#)
  - gett, [62](#)
  - sub\_fl\_q, [62](#)
  - sub\_fl\_q\_0, [62](#)
  - sub\_fl\_q\_styropor, [63](#)
- mo\_thermo\_functions.f90, [77](#)
- mu
  - mo\_parameters, [52](#)
- n\_active
  - mo\_data, [20](#)
- n\_bgc
  - mo\_data, [20](#)
- n\_bottom
  - mo\_data, [20](#)
- n\_middle
  - mo\_data, [20](#)
- n\_time\_out
  - mo\_data, [20](#)
- n\_top
  - mo\_data, [20](#)
- neg\_free
  - mo\_parameters, [52](#)
- nlayer
  - mo\_data, [20](#)
- ocean\_flux\_input
  - mo\_data, [20](#)
- ocean\_t\_input
  - mo\_data, [21](#)
- output
  - mo\_output, [46](#)
- output\_begin
  - mo\_output, [46](#)
- output\_begin\_bgc
  - mo\_output, [46](#)
- output\_bgc
  - mo\_output, [47](#)
- output\_raw
  - mo\_output, [47](#)
- output\_raw\_lay
  - mo\_output, [47](#)
- output\_raw\_snow
  - mo\_output, [47](#)
- output\_settings
  - mo\_output, [47](#)
- para\_flush\_gamma
  - mo\_parameters, [52](#)
- para\_flush\_horiz
  - mo\_parameters, [52](#)
- penetr
  - mo\_parameters, [52](#)
- perm
  - mo\_data, [21](#)
- phi
  - mo\_data, [21](#)
- phi\_s
  - mo\_data, [21](#)
- pi
  - mo\_parameters, [52](#)
- precip\_flag
  - mo\_data, [21](#)
- precip\_input
  - mo\_data, [21](#)

- precipinput
  - mo\_data, [21](#)
- prescribe\_flag
  - mo\_data, [21](#)
- psi\_g
  - mo\_data, [21](#)
- psi\_g\_snow
  - mo\_data, [21](#)
- psi\_l
  - mo\_data, [22](#)
- psi\_l\_snow
  - mo\_data, [22](#)
- psi\_s
  - mo\_data, [22](#)
- psi\_s\_min
  - mo\_parameters, [52](#)
- psi\_s\_snow
  - mo\_data, [22](#)
- psi\_s\_top\_min
  - mo\_parameters, [52](#)
- q
  - mo\_data, [22](#)
- ratio\_flood
  - mo\_parameters, [52](#)
- ray
  - mo\_data, [22](#)
- ray\_crit
  - mo\_parameters, [52](#)
- ref\_salinity
  - mo\_parameters, [52](#)
- rho\_l
  - mo\_parameters, [52](#)
- rho\_s
  - mo\_parameters, [53](#)
- rho\_snow
  - mo\_parameters, [53](#)
- s\_abs
  - mo\_data, [22](#)
- s\_abs\_snow
  - mo\_data, [22](#)
- s\_br
  - mo\_data, [22](#)
- s\_bu
  - mo\_data, [22](#)
- s\_bu\_bottom
  - mo\_data, [23](#)
- s\_total
  - mo\_data, [23](#)
- SAMSIM.f90, [78](#)
  - samsim, [78](#)
- salt\_flag
  - mo\_data, [23](#)
- samsim
  - SAMSIM.f90, [78](#)
- sigma
  - mo\_parameters, [53](#)
- snow\_coupling
  - mo\_snow, [55](#)
- snow\_flush\_flag
  - mo\_data, [23](#)
- snow\_precip
  - mo\_snow, [55](#)
- snow\_precip\_0
  - mo\_snow, [55](#)
- snow\_precip\_flag
  - mo\_data, [23](#)
- snow\_thermo
  - mo\_snow, [56](#)
- snow\_thermo\_meltwater
  - mo\_snow, [56](#)
- solid\_precip
  - mo\_data, [23](#)
- styropor\_flag
  - mo\_data, [23](#)
- styropor\_input
  - mo\_data, [23](#)
- sub\_allocate
  - mo\_init, [40](#)
- sub\_allocate\_bgc
  - mo\_init, [40](#)
- sub\_deallocate
  - mo\_init, [40](#)
- sub\_fl\_q
  - mo\_thermo\_functions, [62](#)
- sub\_fl\_q\_0
  - mo\_thermo\_functions, [62](#)
- sub\_fl\_q\_0\_snow
  - mo\_snow, [57](#)
- sub\_fl\_q\_0\_snow\_thin
  - mo\_snow, [57](#)
- sub\_fl\_q\_snow
  - mo\_snow, [57](#)
- sub\_fl\_q\_styropor
  - mo\_thermo\_functions, [63](#)
- sub\_heat\_fluxes
  - mo\_heat\_fluxes, [37](#)
- sub\_input
  - mo\_functions, [31](#)
- sub\_melt\_snow
  - mo\_functions, [32](#)
- sub\_melt\_thick
  - mo\_functions, [32](#)
- sub\_notzflux
  - mo\_functions, [32](#)
- sub\_test1
  - mo\_testcase\_specifics, [58](#)
- sub\_test2
  - mo\_testcase\_specifics, [58](#)
- sub\_test3
  - mo\_testcase\_specifics, [59](#)
- sub\_test34
  - mo\_testcase\_specifics, [59](#)
- sub\_test4
  - mo\_testcase\_specifics, [59](#)

sub\_test6  
    mo\_testcase\_specifics, [59](#)  
sub\_test9  
    mo\_testcase\_specifics, [59](#)  
sub\_turb\_flux  
    mo\_functions, [33](#)  
surface\_water  
    mo\_data, [23](#)  
  
t  
    mo\_data, [23](#)  
t2m  
    mo\_data, [23](#)  
t2m\_input  
    mo\_data, [24](#)  
t\_bottom  
    mo\_data, [24](#)  
t\_freeze  
    mo\_data, [24](#)  
t\_snow  
    mo\_data, [24](#)  
t\_test  
    mo\_data, [24](#)  
t\_top  
    mo\_data, [24](#)  
tank\_depth  
    mo\_data, [24](#)  
tank\_flag  
    mo\_data, [24](#)  
test  
    mo\_data, [24](#)  
thick  
    mo\_data, [24](#)  
thick\_0  
    mo\_data, [25](#)  
thick\_min  
    mo\_data, [25](#)  
thick\_snow  
    mo\_data, [25](#)  
thickness  
    mo\_data, [25](#)  
time  
    mo\_data, [25](#)  
time\_counter  
    mo\_data, [25](#)  
time\_input  
    mo\_data, [25](#)  
time\_out  
    mo\_data, [25](#)  
time\_total  
    mo\_data, [25](#)  
tinput  
    mo\_data, [25](#)  
top\_grow  
    mo\_layer\_dynamics, [42](#)  
top\_melt  
    mo\_layer\_dynamics, [42](#)  
total\_resist  
    mo\_data, [25](#)  
  
ttop\_input  
    mo\_data, [26](#)  
turb\_a  
    mo\_parameters, [53](#)  
turb\_b  
    mo\_parameters, [53](#)  
turb\_flag  
    mo\_data, [26](#)  
  
v\_ex  
    mo\_data, [26](#)  
v\_g  
    mo\_data, [26](#)  
v\_l  
    mo\_data, [26](#)  
v\_s  
    mo\_data, [26](#)  
  
wp  
    mo\_parameters, [53](#)  
  
x\_grav  
    mo\_parameters, [53](#)  
  
zerok  
    mo\_parameters, [53](#)