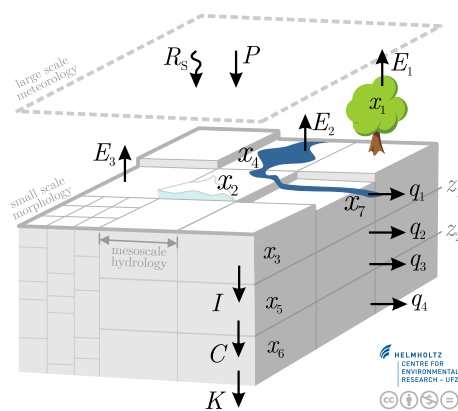


The mesoscale Hydrologic Model

mHM



Documentation for version 5.8

Edited by Luis Samaniego

in cooperation with

Johannes Brenner, Matthias Cuntz, Cüneyd M. Demirel, Maren Kaluza,
Rohini Kumar, Ben Langenberg, Juliane Mai, Oldrich Rakovec, David Schäfer,
Martin Schrön, Simon Stisen, Stephan Thober, Matthias Zink

Former mHM project collaborators

John Craven, Giovanni Dalmasso, Jude Musuuza, Vladyslav Prykhodko, Christoph Schneider,
Diana Spieler

The mHM project

www.ufz.de/mhm

©2005-2017

Helmholtz Centre for Environmental Research - UFZ



Contents

1	Introduction to mHM	1
1.1	Short Description	1
1.2	The Grid-based mHM Model	1
1.3	Model Formulation	3
1.4	The Multiscale Parameter Regionalization Technique	4
1.5	The Parameter Estimation Problem	5
1.6	Model Calibration	6
1.7	Helpful links	7
1.8	Test basin	7
1.9	Protocols	7
2	Getting Started	9
2.1	Receive mHM	9
2.2	Install NETCDF	9
2.2.1	Short Guide Install to NETCDF on Linux (example Ubuntu 16.04 with gfortran v5.4.0)	9
2.2.2	Short Guide Install to NETCDF on MacOS	10
2.3	Run mHM under CYGWIN on Windows	11
2.4	Compile mHM	15
2.5	Test mHM on Example Basin	16
2.6	Run your own Simulation	16
2.6.1	Main Configuration: Paths, Periods, Switches	16
2.6.2	Output Configuration: Time Steps, States, Fluxes	17
2.6.3	Regionalised Parameters: Initial Values and Ranges	17
2.7	Calibration and Optimization	17
2.7.1	The Optimization Routines	18
2.7.2	Calibration Settings for mHM	18
2.7.3	Final Calibration Results	19
3	Data Preparation for mHM	21
3.1	Getting Started	21
3.1.1	Meteorological variables	21
3.1.2	Morphological variables	22

3.1.3	Land Cover	22
3.1.4	Gauging Station Information	22
3.1.5	Optional Data	23
3.2	Preparation of the Forcings	23
3.2.1	Extract Data to the Size of a Catchment	23
3.2.2	Extact Data to the Time Period of Interest	23
3.2.3	Data Types	23
3.2.4	Variable Names	24
3.2.5	The Header File	24
3.2.6	NODATA values	24
3.3	Preparation of the LatLon Grid	24
3.4	Preparation of the Morphological Data	25
3.5	A possible GIS workflow	26
3.5.1	General considerations	26
3.5.2	Slope map	29
3.5.3	Aspect map	29
3.5.4	Fill DEM sinks	30
3.5.5	Flow direction and flow accumulation	30
3.5.5.1	Spatial Analyst	30
3.5.5.2	Arc Hydro Tools	31
3.5.6	Gauges map	32
3.5.7	Watershed delineation	34
3.5.8	Mask the datasets	35
3.5.9	Write the ascii grids	35
3.6	Land Cover Data	35
3.7	Table Data	36
3.7.1	The soil look-up table	36
3.7.2	The hydrogeolgy look-up table	36
3.7.3	The LAI look-up table	37
3.7.4	The gauge files	37
3.8	Post-GIS preparation	38
4	Visualizing Model Output	39
5	Calibration Options	41
5.1	Calibration of Parameters on Observations	41
5.1.1	Parameters	41
5.1.2	Methods	41
5.1.3	Functions	41
5.1.4	Data	41
5.1.5	Output	42

6 Coding and Documentation Style	43
7 The details about the test basin	47
8 Protocols for setting up a new mHM basin	49
8.1 Check for possible input data errors using mHM outputs	49
8.2 Protocol for generating a restart file	50
8.3 Protocol for determining the warm-up period for a new basin	50
9 mHM Dependencies	51
10 Publications using mHM	53
11 mHM RELEASE NOTES	57
12 Modules Index	65
12.1 Modules List	65
13 Data Type Index	69
13.1 Data Types List	69
14 File Index	73
14.1 File List	73
15 Module Documentation	75
15.1 dummy Module Reference	75
15.2 mo_anneal Module Reference	75
15.2.1 Function/Subroutine Documentation	75
15.2.1.1 anneal_dp()	75
15.2.1.2 dchange_dp()	76
15.2.1.3 generate_neighborhood_weight_dp()	77
15.2.1.4 gettemperature_dp()	77
15.2.1.5 pargen_anneal_dp()	78
15.2.1.6 pargen_dds_dp()	79
15.3 mo_append Module Reference	79
15.3.1 Detailed Description	80
15.3.2 Function/Subroutine Documentation	80
15.3.2.1 append_char_3d()	80
15.3.2.2 append_char_m_m()	81
15.3.2.3 append_char_v_s()	81
15.3.2.4 append_char_v_v()	81
15.3.2.5 append_dp_3d()	81
15.3.2.6 append_dp_m_m()	81
15.3.2.7 append_dp_v_s()	81

15.3.2.8	<code>append_dp_v_v()</code>	81
15.3.2.9	<code>append_i4_3d()</code>	82
15.3.2.10	<code>append_i4_m_m()</code>	82
15.3.2.11	<code>append_i4_v_s()</code>	82
15.3.2.12	<code>append_i4_v_v()</code>	82
15.3.2.13	<code>append_i8_3d()</code>	82
15.3.2.14	<code>append_i8_m_m()</code>	82
15.3.2.15	<code>append_i8_v_s()</code>	83
15.3.2.16	<code>append_i8_v_v()</code>	83
15.3.2.17	<code>append_lgt_3d()</code>	83
15.3.2.18	<code>append_lgt_m_m()</code>	83
15.3.2.19	<code>append_lgt_v_s()</code>	83
15.3.2.20	<code>append_lgt_v_v()</code>	83
15.3.2.21	<code>append_sp_3d()</code>	83
15.3.2.22	<code>append_sp_m_m()</code>	84
15.3.2.23	<code>append_sp_v_s()</code>	84
15.3.2.24	<code>append_sp_v_v()</code>	84
15.3.2.25	<code>paste_char_m_m()</code>	84
15.3.2.26	<code>paste_char_m_s()</code>	84
15.3.2.27	<code>paste_char_m_v()</code>	84
15.3.2.28	<code>paste_dp_m_m()</code>	84
15.3.2.29	<code>paste_dp_m_s()</code>	85
15.3.2.30	<code>paste_dp_m_v()</code>	85
15.3.2.31	<code>paste_i4_m_m()</code>	85
15.3.2.32	<code>paste_i4_m_s()</code>	85
15.3.2.33	<code>paste_i4_m_v()</code>	85
15.3.2.34	<code>paste_i8_m_m()</code>	85
15.3.2.35	<code>paste_i8_m_s()</code>	86
15.3.2.36	<code>paste_i8_m_v()</code>	86
15.3.2.37	<code>paste_lgt_m_m()</code>	86
15.3.2.38	<code>paste_lgt_m_s()</code>	86
15.3.2.39	<code>paste_lgt_m_v()</code>	86
15.3.2.40	<code>paste_sp_m_m()</code>	86
15.3.2.41	<code>paste_sp_m_s()</code>	86
15.3.2.42	<code>paste_sp_m_v()</code>	87
15.4	<code>mo_canopy_interc</code> Module Reference	87
15.4.1	Detailed Description	87
15.4.2	Function/Subroutine Documentation	87
15.4.2.1	<code>canopy_interc()</code>	87
15.5	<code>mo_common_variables</code> Module Reference	89

15.5.1 Detailed Description	89
15.5.2 Variable Documentation	89
15.5.2.1 alma_convention	89
15.5.2.2 dds_r	90
15.5.2.3 global_parameters	90
15.5.2.4 global_parameters_name	90
15.5.2.5 mcmc_error_params	90
15.5.2.6 mcmc_opti	90
15.5.2.7 nerror_model	90
15.5.2.8 niterations	90
15.5.2.9 nprocesses	90
15.5.2.10 opti_function	91
15.5.2.11 opti_method	91
15.5.2.12 optimize	91
15.5.2.13 optimize_restart	91
15.5.2.14 processmatrix	91
15.5.2.15 sa_temp	91
15.5.2.16 sce_ngs	91
15.5.2.17 sce_npg	92
15.5.2.18 sce_nps	92
15.5.2.19 seed	92
15.6 mo_constants Module Reference	92
15.6.1 Detailed Description	94
15.6.2 Variable Documentation	94
15.6.2.1 cp0_dp	94
15.6.2.2 cp0_sp	95
15.6.2.3 deg2rad_dp	95
15.6.2.4 deg2rad_sp	95
15.6.2.5 eps_dp	95
15.6.2.6 eps_sp	95
15.6.2.7 euler	95
15.6.2.8 euler_d	95
15.6.2.9 gravity_dp	96
15.6.2.10 gravity_sp	96
15.6.2.11 nerr	96
15.6.2.12 nin	96
15.6.2.13 nnml	96
15.6.2.14 nout	96
15.6.2.15 p0_dp	96
15.6.2.16 p0_sp	97

15.6.2.17	pi	97
15.6.2.18	pi_d	97
15.6.2.19	pi_dp	97
15.6.2.20	pi_sp	97
15.6.2.21	pio2	97
15.6.2.22	pio2_d	97
15.6.2.23	pio2_dp	98
15.6.2.24	pio2_sp	98
15.6.2.25	psychro_dp	98
15.6.2.26	psychro_sp	98
15.6.2.27	rad2deg_dp	98
15.6.2.28	rad2deg_sp	98
15.6.2.29	radiusearth_dp	98
15.6.2.30	radiusearth_sp	99
15.6.2.31	rho0_dp	99
15.6.2.32	rho0_sp	99
15.6.2.33	secday_dp	99
15.6.2.34	secday_sp	99
15.6.2.35	sigma_dp	99
15.6.2.36	sigma_sp	99
15.6.2.37	solarconst_dp	99
15.6.2.38	solarconst_sp	100
15.6.2.39	specheatet_dp	100
15.6.2.40	specheatet_sp	100
15.6.2.41	sqrt2	100
15.6.2.42	sqrt2_d	100
15.6.2.43	sqrt2_dp	100
15.6.2.44	sqrt2_sp	100
15.6.2.45	t0_dp	101
15.6.2.46	t0_sp	101
15.6.2.47	twopi	101
15.6.2.48	twopi_d	101
15.6.2.49	twopi_dp	101
15.6.2.50	twopi_sp	101
15.6.2.51	twothird_dp	101
15.6.2.52	twothird_sp	102
15.7	mo_corr Module Reference	102
15.7.1	Function/Subroutine Documentation	103
15.7.1.1	arth_dp()	103
15.7.1.2	arth_i4()	103

15.7.1.3	arth_sp()	103
15.7.1.4	autocoeffk_1d_dp()	103
15.7.1.5	autocoeffk_1d_sp()	103
15.7.1.6	autocoeffk_dp()	104
15.7.1.7	autocoeffk_sp()	104
15.7.1.8	autocorr_1d_dp()	104
15.7.1.9	autocorr_1d_sp()	104
15.7.1.10	autocorr_dp()	104
15.7.1.11	autocorr_sp()	104
15.7.1.12	corr_dp()	105
15.7.1.13	corr_sp()	105
15.7.1.14	crosscoeffk_dp()	105
15.7.1.15	crosscoeffk_sp()	105
15.7.1.16	crosscorr_dp()	105
15.7.1.17	crosscorr_sp()	105
15.7.1.18	four1_dp()	106
15.7.1.19	four1_sp()	106
15.7.1.20	fourrow_dp()	106
15.7.1.21	fourrow_sp()	106
15.7.1.22	realft_dp()	106
15.7.1.23	realft_sp()	107
15.7.1.24	swap_1d_dpc()	107
15.7.1.25	swap_1d_spc()	107
15.7.1.26	zroots_unity_dp()	107
15.7.1.27	zroots_unity_sp()	108
15.7.2	Variable Documentation	108
15.7.2.1	npar2_arth	108
15.7.2.2	npar_arth	108
15.8	mo_dds Module Reference	109
15.8.1	Detailed Description	109
15.8.2	Function/Subroutine Documentation	109
15.8.2.1	dds()	109
15.8.2.2	mdds()	111
15.8.2.3	neigh_value()	112
15.9	mo_errormeasures Module Reference	113
15.9.1	Function/Subroutine Documentation	114
15.9.1.1	bias_dp_1d()	114
15.9.1.2	bias_dp_2d()	114
15.9.1.3	bias_dp_3d()	114
15.9.1.4	bias_sp_1d()	115

15.9.1.5	<code>bias_sp_2d()</code>	115
15.9.1.6	<code>bias_sp_3d()</code>	115
15.9.1.7	<code>kge_dp_1d()</code>	115
15.9.1.8	<code>kge_dp_2d()</code>	115
15.9.1.9	<code>kge_dp_3d()</code>	115
15.9.1.10	<code>kge_sp_1d()</code>	115
15.9.1.11	<code>kge_sp_2d()</code>	116
15.9.1.12	<code>kge_sp_3d()</code>	116
15.9.1.13	<code>kgenocorr_dp_1d()</code>	116
15.9.1.14	<code>kgenocorr_dp_2d()</code>	116
15.9.1.15	<code>kgenocorr_dp_3d()</code>	116
15.9.1.16	<code>kgenocorr_sp_1d()</code>	116
15.9.1.17	<code>kgenocorr_sp_2d()</code>	117
15.9.1.18	<code>kgenocorr_sp_3d()</code>	117
15.9.1.19	<code>lnnse_dp_1d()</code>	117
15.9.1.20	<code>lnnse_dp_2d()</code>	117
15.9.1.21	<code>lnnse_dp_3d()</code>	117
15.9.1.22	<code>lnnse_sp_1d()</code>	117
15.9.1.23	<code>lnnse_sp_2d()</code>	118
15.9.1.24	<code>lnnse_sp_3d()</code>	118
15.9.1.25	<code>mae_dp_1d()</code>	118
15.9.1.26	<code>mae_dp_2d()</code>	118
15.9.1.27	<code>mae_dp_3d()</code>	119
15.9.1.28	<code>mae_sp_1d()</code>	119
15.9.1.29	<code>mae_sp_2d()</code>	120
15.9.1.30	<code>mae_sp_3d()</code>	120
15.9.1.31	<code>mse_dp_1d()</code>	120
15.9.1.32	<code>mse_dp_2d()</code>	121
15.9.1.33	<code>mse_dp_3d()</code>	122
15.9.1.34	<code>mse_sp_1d()</code>	122
15.9.1.35	<code>mse_sp_2d()</code>	123
15.9.1.36	<code>mse_sp_3d()</code>	124
15.9.1.37	<code>nse_dp_1d()</code>	124
15.9.1.38	<code>nse_dp_2d()</code>	125
15.9.1.39	<code>nse_dp_3d()</code>	125
15.9.1.40	<code>nse_sp_1d()</code>	125
15.9.1.41	<code>nse_sp_2d()</code>	125
15.9.1.42	<code>nse_sp_3d()</code>	125
15.9.1.43	<code>rmse_dp_1d()</code>	125
15.9.1.44	<code>rmse_dp_2d()</code>	126

15.9.1.45 rmse_dp_3d()	126
15.9.1.46 rmse_sp_1d()	127
15.9.1.47 rmse_sp_2d()	127
15.9.1.48 rmse_sp_3d()	127
15.9.1.49 sae_dp_1d()	128
15.9.1.50 sae_dp_2d()	128
15.9.1.51 sae_dp_3d()	129
15.9.1.52 sae_sp_1d()	130
15.9.1.53 sae_sp_2d()	130
15.9.1.54 sae_sp_3d()	131
15.9.1.55 sse_dp_1d()	132
15.9.1.56 sse_dp_2d()	132
15.9.1.57 sse_dp_3d()	133
15.9.1.58 sse_sp_1d()	134
15.9.1.59 sse_sp_2d()	134
15.9.1.60 sse_sp_3d()	135
15.10mo_file Module Reference	136
15.10.1 Detailed Description	138
15.10.2 Variable Documentation	138
15.10.2.1 file_aspect	138
15.10.2.2 file_config	138
15.10.2.3 file_daily_discharge	139
15.10.2.4 file_defoutput	139
15.10.2.5 file_dem	139
15.10.2.6 file_facc	139
15.10.2.7 file_fdir	139
15.10.2.8 file_geolut	139
15.10.2.9 file_hydrogeoclass	139
15.10.2.10file_lai_binary_end	139
15.10.2.11file_lai_header	140
15.10.2.12file_laiclass	140
15.10.2.13file_lailut	140
15.10.2.14file_main	140
15.10.2.15file_meteo_binary_end	140
15.10.2.16file_meteo_header	140
15.10.2.17file_namelist	140
15.10.2.18file_namelist_param	141
15.10.2.19file_opti	141
15.10.2.20file_opti_nml	141
15.10.2.21file_slope	141

15.10.2.22	file_soil_database	141
15.10.2.23	file_soil_database_1	141
15.10.2.24	file_soilclass	142
15.10.2.25	uaspect	142
15.10.2.26	uconfig	142
15.10.2.27	udaily_discharge	142
15.10.2.28	udefoutput	142
15.10.2.29	udem	142
15.10.2.30	udischarge	142
15.10.2.31	ufacc	142
15.10.2.32	ufdir	143
15.10.2.33	ugeolut	143
15.10.2.34	uhydrogeoclass	143
15.10.2.35	ulai	143
15.10.2.36	ulai_header	143
15.10.2.37	ulaiclass	143
15.10.2.38	ulailut	143
15.10.2.39	ulcoverclass	144
15.10.2.40	umeteo	144
15.10.2.41	umeteo_header	144
15.10.2.42	unamelist	144
15.10.2.43	unamelist_param	144
15.10.2.44	uopti	144
15.10.2.45	uopti_nml	144
15.10.2.46	uslope	145
15.10.2.47	usoil_database	145
15.10.2.48	usoilclass	145
15.10.2.49	utws	145
15.10.2.50	version	145
15.10.2.51	version_date	145
15.11	mo_finish Module Reference	145
15.11.1	Detailed Description	146
15.11.2	Function/Subroutine Documentation	146
15.11.2.1	finish()	146
15.12	mo_global_variables Module Reference	147
15.12.1	Detailed Description	151
15.12.2	Variable Documentation	151
15.12.2.1	basin	151
15.12.2.2	basin_avg_tws_obs	151
15.12.2.3	basin_avg_tws_sim	151

15.12.2.4 c2tstu	151
15.12.2.5 contact	152
15.12.2.6 conventions	152
15.12.2.7 dirabsvappressure	152
15.12.2.8 dircommonfiles	152
15.12.2.9 dirconfigout	152
15.12.2.10direvapotranspiration	152
15.12.2.11dirgridded_lai	152
15.12.2.12dircover	152
15.12.2.13dirmaxtemperature	153
15.12.2.14dirmintemperature	153
15.12.2.15dirmorpho	153
15.12.2.16dirnetradiation	153
15.12.2.17dirneutrons	153
15.12.2.18dirout	153
15.12.2.19dirprecipitation	153
15.12.2.20dirreferenceet	153
15.12.2.21dirrestartin	154
15.12.2.22dirrestartout	154
15.12.2.23dirsoil_moisture	154
15.12.2.24dirtemperature	154
15.12.2.25dirwindspeed	154
15.12.2.26evalper	154
15.12.2.27evap_coeff	154
15.12.2.28day_pet	154
15.12.2.29day_prec	155
15.12.2.30day_temp	155
15.12.2.31filelatlon	155
15.12.2.32filetws	155
15.12.2.33night_pet	155
15.12.2.34night_prec	155
15.12.2.35night_temp	155
15.12.2.36fracsealed_cityarea	155
15.12.2.37geounitkar	155
15.12.2.38geounitlist	156
15.12.2.39history	156
15.12.2.40horizonddepth_mhm	156
15.12.2.41iflag_cordinate_sys	156
15.12.2.42iflag_soildb	156
15.12.2.43inputformat_gridded_lai	156

15.12.2.44inputformat_meteo_forcings	156
15.12.2.450_areacell	156
15.12.2.460_asp	157
15.12.2.470_basin	157
15.12.2.480_cellcoor	157
15.12.2.490_elev	157
15.12.2.500_geounit	157
15.12.2.510_gridded_lai	157
15.12.2.520_id	157
15.12.2.530_latitude	157
15.12.2.540_lcover	158
15.12.2.550_lcover_lai	158
15.12.2.560_longitude	158
15.12.2.570_mask	158
15.12.2.580_ncells	158
15.12.2.590_slope	158
15.12.2.600_slope_emp	158
15.12.2.610_soilid	158
15.12.2.621_absvappress	159
15.12.2.631_aeroresist	159
15.12.2.641_aetcanopy	159
15.12.2.651_aetsealed	159
15.12.2.661_aetsoil	159
15.12.2.671_alpha	159
15.12.2.681_areacell	159
15.12.2.691_baseflow	160
15.12.2.701_cellcoor	160
15.12.2.711_degday	160
15.12.2.721_degdayinc	160
15.12.2.731_degdaymax	160
15.12.2.741_degdaynopre	160
15.12.2.751_downbound_l0	160
15.12.2.761_et	161
15.12.2.771_et_mask	161
15.12.2.781_fasp	161
15.12.2.791_fastrunoff	161
15.12.2.801_fforest	161
15.12.2.811_fperm	161
15.12.2.821_froots	161
15.12.2.831_fsealed	162

15.12.2.841_harsamcoeff	162
15.12.2.851_id	162
15.12.2.861_infilsoil	162
15.12.2.871_inter	162
15.12.2.881_jarvis_thresh_c1	162
15.12.2.891_karstloss	162
15.12.2.901_kbaseflow	163
15.12.2.911_kfastflow	163
15.12.2.921_kperco	163
15.12.2.931_kslowflow	163
15.12.2.941_latitude	163
15.12.2.951_leftbound_l0	163
15.12.2.961_longitude	163
15.12.2.971_maxinter	164
15.12.2.981_melt	164
15.12.2.991_ncells	164
15.12.2.100_netrad	164
15.12.2.101_neutrons	164
15.12.2.102_neutronsdata	164
15.12.2.103_neutronsdata_mask	164
15.12.2.104_ntcells_l0	165
15.12.2.105_percol	165
15.12.2.106_pet	165
15.12.2.107_pet_calc	165
15.12.2.108_pet_weights	165
15.12.2.109_petlaicorfactor	165
15.12.2.110_pre	165
15.12.2.111_pre_weights	165
15.12.2.112_preeffect	166
15.12.2.113_prietayalpha	166
15.12.2.114_rain	166
15.12.2.115_rect_latitude	166
15.12.2.116_rect_longitude	166
15.12.2.117_rightbound_l0	166
15.12.2.118_runoffseal	166
15.12.2.119_satstw	167
15.12.2.120_sealedthresh	167
15.12.2.121_sealstw	167
15.12.2.122_slowrunoff	167
15.12.2.123_sm	167

15.12.2.124_sm_mask	167
15.12.2.125_snow	167
15.12.2.126_snowpack	168
15.12.2.127_soilmoist	168
15.12.2.128_soilmoistexp	168
15.12.2.129_soilmoistfc	168
15.12.2.130_soilmoistsat	168
15.12.2.131_surfresist	168
15.12.2.132_temp	168
15.12.2.133_temp_weights	169
15.12.2.134_tempthresh	169
15.12.2.135_throughfall	169
15.12.2.136_tmax	169
15.12.2.137_tmin	169
15.12.2.138_total_runoff	169
15.12.2.139_unsatstw	169
15.12.2.140_unsatthresh	169
15.12.2.141_upbound_l0	170
15.12.2.142_wiltingpoint	170
15.12.2.143_windspeed	170
15.12.2.144ilut	170
15.12.2.145iunitlist	170
15.12.2.146ts	170
15.12.2.147filename	170
15.12.2.148yearid	170
15.12.2.149vel0	171
15.12.2.150vel1	171
15.12.2.151vel2	171
15.12.2.152ns	171
15.12.2.153hm_details	171
15.12.2.154basins	171
15.12.2.155neutron_integral_afast	172
15.12.2.156geounits	172
15.12.2.157aiclass	172
15.12.2.158coverscene	172
15.12.2.159measperday_tws	172
15.12.2.160soilhorizons_mhm	172
15.12.2.161soilhorizons_sm_input	172
15.12.2.162soiltypes	172
15.12.2.163timesteps_l1_et	173

15.12.2.164	timesteps_l1_neutrons	173
15.12.2.165	timesteps_l1_sm	173
15.12.2.166	stepday	173
15.12.2.167	outputfluxstate	173
15.12.2.168	perform_mpr	173
15.12.2.169	project_details	173
15.12.2.170	read_meteo_weights	173
15.12.2.171	read_restart	174
15.12.2.172	readper	174
15.12.2.173	resolutionhydrology	174
15.12.2.174	resolutionrouting	174
15.12.2.175	routingstates	174
15.12.2.176	setup_description	174
15.12.2.177	warmp	174
15.12.2.178	simulation_type	174
15.12.2.179	soildb	175
15.12.2.180	stagedepth	175
15.12.2.181	timestep	175
15.12.2.182	timestep_et_input	175
15.12.2.183	timestep_lai_input	175
15.12.2.184	timestep_model_inputs	175
15.12.2.185	timestep_model_outputs	175
15.12.2.186	timestep_neutrons_input	175
15.12.2.187	timestep_sm_input	176
15.12.2.188	warmingdays	176
15.12.2.189	warmp	176
15.12.2.190	write_restart	176
15.12.2.191	xcoor	176
15.12.2.192	ycoor	176
15.13	mo_init_states Module Reference	176
15.13.1	Detailed Description	177
15.13.2	Function/Subroutine Documentation	177
15.13.2.1	calculate_grid_properties()	177
15.13.2.2	get_basin_info()	179
15.13.2.3	variables_alloc()	180
15.13.2.4	variables_default_init()	182
15.14	mo_julian Module Reference	183
15.14.1	Detailed Description	185
15.14.2	Function/Subroutine Documentation	185
15.14.2.1	caldat()	185

15.14.2.2 caldat360()	187
15.14.2.3 caldat365()	188
15.14.2.4 caldatjulian()	190
15.14.2.5 date2dec()	192
15.14.2.6 date2dec360()	194
15.14.2.7 date2dec365()	195
15.14.2.8 date2decjulian()	197
15.14.2.9 dec2date()	199
15.14.2.10dec2date360()	201
15.14.2.11dec2date365()	203
15.14.2.12dec2datejulian()	205
15.14.2.13jday()	207
15.14.2.14jday360()	209
15.14.2.15jday365()	210
15.14.2.16jdayjulian()	211
15.14.2.17ndays()	213
15.14.2.18ndyin()	213
15.14.2.19selectcalendar()	214
15.14.2.20setcalendarinteger()	215
15.14.2.21setcalendarstring()	216
15.14.3 Variable Documentation	217
15.14.3.1 calendar	217
15.15mo_kind Module Reference	217
15.15.1 Detailed Description	217
15.15.2 Variable Documentation	218
15.15.2.1 dp	218
15.15.2.2 dpc	218
15.15.2.3 i1	218
15.15.2.4 i2	218
15.15.2.5 i4	219
15.15.2.6 i8	219
15.15.2.7 lgt	219
15.15.2.8 sp	219
15.15.2.9 spc	220
15.16mo_linfit Module Reference	220
15.16.1 Detailed Description	220
15.16.2 Function/Subroutine Documentation	220
15.16.2.1 linfit_dp()	220
15.16.2.2 linfit_sp()	221
15.17mo_mcmc Module Reference	221

15.17.1 Detailed Description	221
15.17.2 Function/Subroutine Documentation	221
15.17.2.1 generatenewparameterset_dp()	222
15.17.2.2 mcmc_dp()	222
15.17.2.3 mcmc_stddev_dp()	223
15.17.2.4 pargen_dp()	223
15.17.2.5 pargennorm_dp()	224
15.18mo_message Module Reference	224
15.18.1 Detailed Description	224
15.18.2 Function/Subroutine Documentation	225
15.18.2.1 message()	225
15.18.3 Variable Documentation	226
15.18.3.1 message_text	226
15.19mo_meteo_forcings Module Reference	226
15.19.1 Detailed Description	227
15.19.2 Function/Subroutine Documentation	227
15.19.2.1 chunk_config()	227
15.19.2.2 chunk_size()	228
15.19.2.3 is_read()	230
15.19.2.4 meteo_forcings_wrapper()	231
15.19.2.5 meteo_weights_wrapper()	234
15.19.2.6 prepare_meteo_forcings_data()	235
15.20mo_mhm Module Reference	237
15.20.1 Detailed Description	238
15.20.2 Function/Subroutine Documentation	238
15.20.2.1 mhm()	238
15.21mo_mhm_constants Module Reference	243
15.21.1 Detailed Description	245
15.21.2 Variable Documentation	245
15.21.2.1 bulkdens_orgmatter	245
15.21.2.2 c1_initstatesm	245
15.21.2.3 cosmic_alpha	246
15.21.2.4 cosmic_bd	246
15.21.2.5 cosmic_l1	246
15.21.2.6 cosmic_l2	246
15.21.2.7 cosmic_l3	246
15.21.2.8 cosmic_l4	246
15.21.2.9 cosmic_n	246
15.21.2.10cosmic_vwclat	246
15.21.2.11dayhours	247

15.21.2.12daysecs	247
15.21.2.13desilets_a0	247
15.21.2.14desilets_a1	247
15.21.2.15desilets_a2	247
15.21.2.16duffiedelta1	247
15.21.2.17duffiedelta2	247
15.21.2.18duffiedr	248
15.21.2.19field_cap_c1	248
15.21.2.20field_cap_c2	248
15.21.2.21h2odens	248
15.21.2.22harsamconst	248
15.21.2.23karman	248
15.21.2.24ks_c	248
15.21.2.25ai_factor_surfresi	249
15.21.2.26ai_offset_surfresi	249
15.21.2.27max_surfresist	249
15.21.2.28maxgeounit	249
15.21.2.29maxnlcovers	249
15.21.2.30maxnobasins	249
15.21.2.31maxnosoilhorizons	249
15.21.2.32ncolpars	249
15.21.2.33nlcover_class	250
15.21.2.34nodata_dp	250
15.21.2.35nodata_i4	250
15.21.2.36noutflxstate	250
15.21.2.37p1_initstatefluxes	250
15.21.2.38p2_initstatefluxes	250
15.21.2.39p3_initstatefluxes	251
15.21.2.40p4_initstatefluxes	251
15.21.2.41p5_initstatefluxes	251
15.21.2.42pwp_c	251
15.21.2.43pwp_matpot_thetar	251
15.21.2.44satpressureslope1	251
15.21.2.45stboltzmann	251
15.21.2.46tetens_c1	251
15.21.2.47tetens_c2	252
15.21.2.48tetens_c3	252
15.21.2.49vgenuchten_sandtresh	252
15.21.2.50vgenuchtenn_c1	252
15.21.2.51vgenuchtenn_c10	252

15.21.2.52	genuchtenn_c11	252
15.21.2.53	genuchtenn_c12	252
15.21.2.54	genuchtenn_c13	252
15.21.2.55	genuchtenn_c14	253
15.21.2.56	genuchtenn_c15	253
15.21.2.57	genuchtenn_c16	253
15.21.2.58	genuchtenn_c17	253
15.21.2.59	genuchtenn_c18	253
15.21.2.60	genuchtenn_c2	253
15.21.2.61	genuchtenn_c3	253
15.21.2.62	genuchtenn_c4	253
15.21.2.63	genuchtenn_c5	253
15.21.2.64	genuchtenn_c6	254
15.21.2.65	genuchtenn_c7	254
15.21.2.66	genuchtenn_c8	254
15.21.2.67	genuchtenn_c9	254
15.21.2.68	windmeasheight	254
15.21.2.69	yeardays	254
15.21.2.70	yearmonths	254
15.21.2.71	yearmonths_i4	254
15.22	mo_mhm_eval Module Reference	255
15.22.1	Detailed Description	255
15.22.2	Function/Subroutine Documentation	255
15.22.2.1	mhm_eval()	255
15.23	mo_moment Module Reference	258
15.23.1	Function/Subroutine Documentation	259
15.23.1.1	absdev_dp()	259
15.23.1.2	absdev_sp()	260
15.23.1.3	average_dp()	260
15.23.1.4	average_sp()	260
15.23.1.5	central_moment_dp()	260
15.23.1.6	central_moment_sp()	260
15.23.1.7	central_moment_var_dp()	260
15.23.1.8	central_moment_var_sp()	260
15.23.1.9	correlation_dp()	261
15.23.1.10	correlation_sp()	261
15.23.1.11	covariance_dp()	261
15.23.1.12	covariance_sp()	261
15.23.1.13	kurtosis_dp()	261
15.23.1.14	kurtosis_sp()	261

15.23.1.15	mean_dp()	262
15.23.1.16	mean_sp()	262
15.23.1.17	mixed_central_moment_dp()	262
15.23.1.18	mixed_central_moment_sp()	262
15.23.1.19	mixed_central_moment_var_dp()	262
15.23.1.20	mixed_central_moment_var_sp()	262
15.23.1.21	moment_dp()	263
15.23.1.22	moment_sp()	263
15.23.1.23	skewness_dp()	263
15.23.1.24	skewness_sp()	263
15.23.1.25	stddev_dp()	263
15.23.1.26	stddev_sp()	264
15.23.1.27	variance_dp()	264
15.23.1.28	variance_sp()	264
15.24	mo_mpr_pet Module Reference	264
15.24.1	Detailed Description	264
15.24.2	Function/Subroutine Documentation	265
15.24.2.1	bulksurface_resistance()	265
15.24.2.2	pet_correctbyasp()	267
15.24.2.3	pet_correctbylai()	268
15.24.2.4	priestley_taylor_alpha()	271
15.25	mo_mpr_runoff Module Reference	272
15.25.1	Detailed Description	273
15.25.2	Function/Subroutine Documentation	273
15.25.2.1	mpr_runoff()	273
15.26	mo_mpr_smhorizons Module Reference	275
15.26.1	Detailed Description	275
15.26.2	Function/Subroutine Documentation	276
15.26.2.1	mpr_smhorizons()	276
15.27	mo_mpr_soilmoist Module Reference	279
15.27.1	Detailed Description	280
15.27.2	Function/Subroutine Documentation	280
15.27.2.1	field_cap()	280
15.27.2.2	genuchten()	281
15.27.2.3	hydro_cond()	283
15.27.2.4	mpr_sm()	284
15.27.2.5	pwp()	288
15.28	mo_mrm_constants Module Reference	289
15.28.1	Variable Documentation	290
15.28.1.1	deltah	290

15.28.1.2 given_ts	290
15.28.1.3 hoursecs	290
15.28.1.4 maxnlcovers	290
15.28.1.5 maxnobasins	290
15.28.1.6 maxnogauges	290
15.28.1.7 ncolpars	291
15.28.1.8 nodata_dp	291
15.28.1.9 nodata_i4	291
15.28.1.10noutflxstate	291
15.28.1.11nroutingstates	291
15.28.1.12p1_initstatefluxes	291
15.28.1.13rout_space_weight	292
15.29mo_mrm_eval Module Reference	292
15.29.1 Detailed Description	292
15.29.2 Function/Subroutine Documentation	292
15.29.2.1 mrm_eval()	292
15.30mo_mrm_file Module Reference	294
15.30.1 Detailed Description	296
15.30.2 Variable Documentation	296
15.30.2.1 file_config	296
15.30.2.2 file_daily_discharge	296
15.30.2.3 file_defoutput	296
15.30.2.4 file_dem	296
15.30.2.5 file_facc	296
15.30.2.6 file_fdir	297
15.30.2.7 file_gaugeloc	297
15.30.2.8 file_main	297
15.30.2.9 file_mrm_output	297
15.30.2.10file_namelist_mrm	297
15.30.2.11file_namelist_param_mrm	297
15.30.2.12file_opti	298
15.30.2.13file_opti_nml	298
15.30.2.14ncfile_discharge	298
15.30.2.15uconfig	298
15.30.2.16udaily_discharge	298
15.30.2.17udefoutput	298
15.30.2.18udem	298
15.30.2.19udischarge	299
15.30.2.20ufacc	299
15.30.2.21ufdir	299

15.30.2.22	ugauoloc	299
15.30.2.23	ulcoverclass	299
15.30.2.24	unamelist_mrm	299
15.30.2.25	unamelist_param	300
15.30.2.26	uopti	300
15.30.2.27	uopti_nml	300
15.30.2.28	version	300
15.30.2.29	version_date	300
15.31	mo_mrm_global_variables Module Reference	300
15.31.1	Detailed Description	303
15.31.2	Variable Documentation	303
15.31.2.1	basin_mrm	303
15.31.2.2	contact	303
15.31.2.3	conventions	303
15.31.2.4	dircommonfiles	304
15.31.2.5	dirconfigout	304
15.31.2.6	dirgauges	304
15.31.2.7	dirlcover	304
15.31.2.8	dirmorpho	304
15.31.2.9	dirout	304
15.31.2.10	dirrestartin	304
15.31.2.11	dirrestartout	304
15.31.2.12	dirtotalrunoff	305
15.31.2.13	evalper	305
15.31.2.14	filelatlon	305
15.31.2.15	fracsealed_cityarea	305
15.31.2.16	gauge	305
15.31.2.17	history	305
15.31.2.18	flag_cordinate_sys	305
15.31.2.19	nflowgauge	306
15.31.2.20	s_start	306
15.31.2.21	l0_areacell	306
15.31.2.22	l0_basin	306
15.31.2.23	l0_cellcoor	306
15.31.2.24	l0_dracell	306
15.31.2.25	l0_drasc	306
15.31.2.26	l0_elev_mrm	307
15.31.2.27	l0_elev_read	307
15.31.2.28	l0_facc	307
15.31.2.29	l0_fdir	307

15.31.2.300_floodplain	307
15.31.2.310_gaugeloc	307
15.31.2.320_id	307
15.31.2.330_inflowgaugeloc	307
15.31.2.340_l11_id	308
15.31.2.350_latitude	308
15.31.2.360_lcover_mrm	308
15.31.2.370_lcover_read	308
15.31.2.380_longitude	308
15.31.2.390_mask_mrm	308
15.31.2.400_ncells	308
15.31.2.410_streamnet	308
15.31.2.4211_afloodplain	309
15.31.2.4311_areacell	309
15.31.2.4411_c1	309
15.31.2.4511_c2	309
15.31.2.4611_cellcoor	309
15.31.2.4711_colout	309
15.31.2.4811_downbound_l0	309
15.31.2.4911_downbound_l1	310
15.31.2.5011_fcol	310
15.31.2.5111_fdir	310
15.31.2.5211_fracpimp	310
15.31.2.5311_fromn	310
15.31.2.5411_frow	310
15.31.2.5511_id	310
15.31.2.5611_k	311
15.31.2.5711_l1_id	311
15.31.2.5811_label	311
15.31.2.5911_leftbound_l0	311
15.31.2.6011_leftbound_l1	311
15.31.2.6111_length	311
15.31.2.6211_ncells	311
15.31.2.6311_netperm	312
15.31.2.6411_noutlets	312
15.31.2.6511_qmod	312
15.31.2.6611_qout	312
15.31.2.6711_qtin	312
15.31.2.6811_qtr	312
15.31.2.6911_rect_latitude	312

15.31.2.7011_rect_longitude	313
15.31.2.7111_rightbound_l0	313
15.31.2.7211_rightbound_l1	313
15.31.2.7311_rorder	313
15.31.2.7411_rowout	313
15.31.2.7511_sink	313
15.31.2.7611_slope	313
15.31.2.7711_tcol	314
15.31.2.7811_ton	314
15.31.2.7911_trow	314
15.31.2.8011_tsrout	314
15.31.2.8111_upbound_l0	314
15.31.2.8211_upbound_l1	314
15.31.2.8311_xi	314
15.31.2.8411_areacell	315
15.31.2.8511_id	315
15.31.2.8611_l1_id	315
15.31.2.8711_ncells	315
15.31.2.8811_total_runoff_in	315
15.31.2.89cfilename	315
15.31.2.90cyearid	315
15.31.2.91level0	315
15.31.2.92level1	316
15.31.2.93level11	316
15.31.2.94level110	316
15.31.2.95mhm_details	316
15.31.2.96mrm_coupling_mode	316
15.31.2.97mrm_runoff	316
15.31.2.98nbasins	316
15.31.2.99ngaugestotal	317
15.31.2.100inflowgaugestotal	317
15.31.2.101lcoverscene	317
15.31.2.102measperday	317
15.31.2.103stepday	317
15.31.2.104outputflxstate_mrm	317
15.31.2.105perform_mpr	318
15.31.2.106project_details	318
15.31.2.107read_restart	318
15.31.2.108adper	318
15.31.2.109solutionhydrology	318

15.31.2.110 resolutionrouting	318
15.31.2.111 setup_description	318
15.31.2.112 warmer	318
15.31.2.113 simulation_type	319
15.31.2.114 timestep	319
15.31.2.115 timestep_model_inputs	319
15.31.2.116 timestep_model_outputs_mrm	319
15.31.2.117 warmingdays_mrm	319
15.31.2.118 warmer	319
15.31.2.119 write_restart	319
15.32mo_mrm_init Module Reference	320
15.32.1 Detailed Description	320
15.32.2 Function/Subroutine Documentation	320
15.32.2.1 config_output()	320
15.32.2.2 l0_check_input_routing()	321
15.32.2.3 mrm_init()	322
15.32.2.4 mrm_init_param()	324
15.32.2.5 mrm_update_param()	325
15.32.2.6 print_startup_message()	326
15.32.2.7 variables_alloc_routing()	327
15.32.2.8 variables_default_init_routing()	328
15.33mo_mrm_mpr Module Reference	328
15.33.1 Detailed Description	329
15.33.2 Function/Subroutine Documentation	329
15.33.2.1 reg_rout()	329
15.34mo_mrm_net_startup Module Reference	330
15.34.1 Detailed Description	331
15.34.2 Function/Subroutine Documentation	332
15.34.2.1 celllength()	332
15.34.2.2 get_distance_two_lat_lon_points()	332
15.34.2.3 l11_flow_direction()	333
15.34.2.4 l11_fraction_sealed_floodplain()	335
15.34.2.5 l11_link_location()	337
15.34.2.6 l11_routing_order()	338
15.34.2.7 l11_set_drain_outlet_gauges()	339
15.34.2.8 l11_set_network_topology()	340
15.34.2.9 l11_stream_features()	341
15.34.2.10 l11_variable_init()	342
15.34.2.11 movedownonecell()	343
15.34.2.12 moveup()	344

15.35mo_mrm_objective_function_runoff Module Reference	344
15.35.1 Detailed Description	345
15.35.2 Function/Subroutine Documentation	346
15.35.2.1 eval()	346
15.35.2.2 extract_runoff()	348
15.35.2.3 loglikelihood()	350
15.35.2.4 loglikelihood_evin2013_2()	353
15.35.2.5 loglikelihood_stddev()	355
15.35.2.6 loglikelihood_trend_no_autocorr()	357
15.35.2.7 multi_objective_ae_fdc_lsv_nse_djf()	359
15.35.2.8 multi_objective_lnnse_highflow_lnnse_lowflow()	361
15.35.2.9 multi_objective_lnnse_highflow_lnnse_lowflow_2()	363
15.35.2.10multi_objective_nse_lnnse()	366
15.35.2.11multi_objective_runoff()	368
15.35.2.12objective_equal_nse_lnnse()	369
15.35.2.13objective_kge()	372
15.35.2.14objective_lnnse()	374
15.35.2.15objective_multiple_gauges_kge_power6()	376
15.35.2.16objective_nse()	379
15.35.2.17objective_power6_nse_lnnse()	381
15.35.2.18objective_sse()	383
15.35.2.19parameter_regularization()	385
15.35.2.20single_objective_runoff()	385
15.36mo_mrm_read_config Module Reference	387
15.36.1 Detailed Description	387
15.36.2 Function/Subroutine Documentation	387
15.36.2.1 in_bound()	387
15.36.2.2 read_mrm_config()	388
15.36.2.3 read_mrm_config_coupling()	389
15.36.2.4 read_mrm_routing_params()	390
15.37mo_mrm_read_data Module Reference	391
15.37.1 Detailed Description	392
15.37.2 Function/Subroutine Documentation	392
15.37.2.1 mrm_l0_variable_init()	392
15.37.2.2 mrm_l1_variable_init()	393
15.37.2.3 mrm_read_discharge()	394
15.37.2.4 mrm_read_l0_data()	395
15.37.2.5 mrm_read_total_runoff()	396
15.37.2.6 rotate_fdir_variable()	398
15.38mo_mrm_read_latlon Module Reference	398

15.38.1 Detailed Description	398
15.38.2 Function/Subroutine Documentation	398
15.38.2.1 read_latlon()	398
15.39mo_mrm_restart Module Reference	399
15.39.1 Detailed Description	400
15.39.2 Function/Subroutine Documentation	400
15.39.2.1 mrm_read_restart_config()	400
15.39.2.2 mrm_read_restart_states()	401
15.39.2.3 mrm_write_restart()	403
15.40mo_mrm_routing Module Reference	404
15.40.1 Detailed Description	404
15.40.2 Function/Subroutine Documentation	405
15.40.2.1 add_inflow()	405
15.40.2.2 l11_routing()	406
15.40.2.3 l11_runoff_acc()	408
15.40.2.4 mrm_routing()	409
15.41mo_mrm_signatures Module Reference	413
15.41.1 Detailed Description	414
15.41.2 Function/Subroutine Documentation	414
15.41.2.1 autocorrelation()	414
15.41.2.2 flowdurationcurve()	415
15.41.2.3 limb_densities()	416
15.41.2.4 maximummonthlyflow()	417
15.41.2.5 moments()	418
15.41.2.6 peakdistribution()	420
15.41.2.7 runoffratio()	421
15.41.2.8 zeroflowratio()	422
15.42mo_mrm_tools Module Reference	423
15.42.1 Detailed Description	423
15.42.2 Function/Subroutine Documentation	423
15.42.2.1 calculate_grid_properties()	423
15.42.2.2 get_basin_info_mrm()	425
15.43mo_mrm_write Module Reference	427
15.43.1 Detailed Description	428
15.43.2 Function/Subroutine Documentation	428
15.43.2.1 mrm_write()	428
15.43.2.2 mrm_write_optfile()	429
15.43.2.3 mrm_write_optinamelist()	430
15.43.2.4 mrm_write_output_fluxes()	431
15.43.2.5 write_configfile()	433

15.43.2.6 write_daily_obs_sim_discharge()	434
15.43.3 Variable Documentation	435
15.43.3.1 average_counter	436
15.43.3.2 day_counter	436
15.43.3.3 month_counter	436
15.43.3.4 nc	436
15.43.3.5 year_counter	436
15.44mo_mrm_write_fluxes_states Module Reference	436
15.44.1 Detailed Description	437
15.44.2 Function/Subroutine Documentation	437
15.44.2.1 close()	437
15.44.2.2 createoutputfile()	438
15.44.2.3 fluxesunit()	439
15.44.2.4 geocoordinates()	440
15.44.2.5 mapcoordinates()	441
15.44.2.6 newoutputdataset()	441
15.44.2.7 newoutputvariable()	442
15.44.2.8 updatedataset()	443
15.44.2.9 updatevariable()	443
15.44.2.10 writetimestep()	444
15.44.2.11 writevariableattributes()	445
15.44.2.12 writevariabletimestep()	445
15.45mo_multi_param_reg Module Reference	446
15.45.1 Detailed Description	446
15.45.2 Function/Subroutine Documentation	447
15.45.2.1 aerodynamical_resistance()	447
15.45.2.2 baseflow_param()	449
15.45.2.3 canopy_intercept_param()	450
15.45.2.4 iper_thres_runoff()	452
15.45.2.5 karstic_layer()	453
15.45.2.6 mpr()	456
15.45.2.7 snow_acc_melt_param()	461
15.46mo_ncread Module Reference	463
15.46.1 Function/Subroutine Documentation	464
15.46.1.1 check()	464
15.46.1.2 get_info()	465
15.46.1.3 get_ncdim()	467
15.46.1.4 get_ncdimatt()	469
15.46.1.5 get_ncvar_0d_dp()	469
15.46.1.6 get_ncvar_0d_i1()	469

15.46.1.7	get_ncvar_0d_i4()	470
15.46.1.8	get_ncvar_0d_sp()	470
15.46.1.9	get_ncvar_1d_dp()	471
15.46.1.10	get_ncvar_1d_i1()	471
15.46.1.11	get_ncvar_1d_i4()	472
15.46.1.12	get_ncvar_1d_sp()	472
15.46.1.13	get_ncvar_2d_dp()	473
15.46.1.14	get_ncvar_2d_i1()	473
15.46.1.15	get_ncvar_2d_i4()	474
15.46.1.16	get_ncvar_2d_sp()	474
15.46.1.17	get_ncvar_3d_dp()	475
15.46.1.18	get_ncvar_3d_i1()	475
15.46.1.19	get_ncvar_3d_i4()	476
15.46.1.20	get_ncvar_3d_sp()	476
15.46.1.21	get_ncvar_4d_dp()	477
15.46.1.22	get_ncvar_4d_i1()	477
15.46.1.23	get_ncvar_4d_i4()	478
15.46.1.24	get_ncvar_4d_sp()	478
15.46.1.25	get_ncvar_5d_dp()	479
15.46.1.26	get_ncvar_5d_i1()	479
15.46.1.27	get_ncvar_5d_i4()	480
15.46.1.28	get_ncvar_5d_sp()	480
15.46.1.29	get_ncvaratt()	481
15.46.1.30	ncclose()	482
15.46.1.31	ncopen()	482
15.47	mo_ncwrite Module Reference	483
15.47.1	Function/Subroutine Documentation	484
15.47.1.1	check()	484
15.47.1.2	close_netcdf()	485
15.47.1.3	create_netcdf()	487
15.47.1.4	dump_netcdf_1d_dp()	488
15.47.1.5	dump_netcdf_1d_i4()	488
15.47.1.6	dump_netcdf_1d_sp()	489
15.47.1.7	dump_netcdf_2d_dp()	489
15.47.1.8	dump_netcdf_2d_i4()	490
15.47.1.9	dump_netcdf_2d_sp()	490
15.47.1.10	dump_netcdf_3d_dp()	491
15.47.1.11	dump_netcdf_3d_i4()	491
15.47.1.12	dump_netcdf_3d_sp()	492
15.47.1.13	dump_netcdf_4d_dp()	492

15.47.1.14	dump_netcdf_4d_i4()	493
15.47.1.15	dump_netcdf_4d_sp()	493
15.47.1.16	dump_netcdf_5d_dp()	494
15.47.1.17	dump_netcdf_5d_i4()	494
15.47.1.18	dump_netcdf_5d_sp()	495
15.47.1.19	open_netcdf()	495
15.47.1.20	var2nc_1d_dp()	497
15.47.1.21	var2nc_1d_i4()	498
15.47.1.22	var2nc_1d_sp()	499
15.47.1.23	var2nc_2d_dp()	499
15.47.1.24	var2nc_2d_i4()	500
15.47.1.25	var2nc_2d_sp()	501
15.47.1.26	var2nc_3d_dp()	501
15.47.1.27	var2nc_3d_i4()	502
15.47.1.28	var2nc_3d_sp()	503
15.47.1.29	var2nc_4d_dp()	503
15.47.1.30	var2nc_4d_i4()	504
15.47.1.31	var2nc_4d_sp()	505
15.47.1.32	var2nc_5d_dp()	505
15.47.1.33	var2nc_5d_i4()	506
15.47.1.34	var2nc_5d_sp()	507
15.47.1.35	write_dynamic_netcdf()	507
15.47.1.36	write_static_netcdf()	508
15.47.2	Variable Documentation	508
15.47.2.1	dnc	508
15.47.2.2	gatt	508
15.47.2.3	maxlen	509
15.47.2.4	nattdim	509
15.47.2.5	ndims	509
15.47.2.6	ngatt	509
15.47.2.7	nmaxatt	509
15.47.2.8	nmaxdim	509
15.47.2.9	nvars	509
15.47.2.10	v	509
15.48	mo_netcdf Module Reference	510
15.48.1	Detailed Description	512
15.48.2	Function/Subroutine Documentation	513
15.48.2.1	check()	513
15.48.2.2	close()	513
15.48.2.3	equalncdimensions()	514

15.48.2.4	getdata1df32()	514
15.48.2.5	getdata1df64()	514
15.48.2.6	getdata1di16()	515
15.48.2.7	getdata1di32()	515
15.48.2.8	getdata1di8()	516
15.48.2.9	getdata2df32()	516
15.48.2.10	getdata2df64()	517
15.48.2.11	getdata2di16()	517
15.48.2.12	getdata2di32()	518
15.48.2.13	getdata2di8()	518
15.48.2.14	getdata3df32()	519
15.48.2.15	getdata3df64()	519
15.48.2.16	getdata3di16()	520
15.48.2.17	getdata3di32()	520
15.48.2.18	getdata3di8()	521
15.48.2.19	getdata4df32()	521
15.48.2.20	getdata4df64()	522
15.48.2.21	getdata4di16()	522
15.48.2.22	getdata4di32()	523
15.48.2.23	getdata4di8()	523
15.48.2.24	getdata5df32()	524
15.48.2.25	getdata5df64()	524
15.48.2.26	getdata5di16()	525
15.48.2.27	getdata5di32()	525
15.48.2.28	getdata5di8()	526
15.48.2.29	getdatascalarf32()	526
15.48.2.30	getdatascalarf64()	527
15.48.2.31	getdatascalari16()	527
15.48.2.32	getdatascalari32()	528
15.48.2.33	getdatascalari8()	528
15.48.2.34	getdimensionbyid()	529
15.48.2.35	getdimensionbyname()	529
15.48.2.36	getdimensionlength()	530
15.48.2.37	getdimensionname()	530
15.48.2.38	getdtypefrominteger()	530
15.48.2.39	getdtypefromstring()	531
15.48.2.40	getglobalattributechar()	531
15.48.2.41	getglobalattributef32()	532
15.48.2.42	getglobalattributef64()	532
15.48.2.43	getglobalattributei16()	532

15.48.2.44	<code>getglobalattributei32()</code>	533
15.48.2.45	<code>getglobalattributei8()</code>	533
15.48.2.46	<code>getnodimensions()</code>	534
15.48.2.47	<code>getnovariables()</code>	534
15.48.2.48	<code>getreaddatashape()</code>	534
15.48.2.49	<code>getunlimiteddimension()</code>	535
15.48.2.50	<code>getvariableattributechar()</code>	536
15.48.2.51	<code>getvariableattributef32()</code>	536
15.48.2.52	<code>getvariableattributef64()</code>	537
15.48.2.53	<code>getvariableattributei16()</code>	537
15.48.2.54	<code>getvariableattributei32()</code>	537
15.48.2.55	<code>getvariableattributei8()</code>	538
15.48.2.56	<code>getvariablebyname()</code>	538
15.48.2.57	<code>getvariabledimensions()</code>	539
15.48.2.58	<code>getvariabledtype()</code>	539
15.48.2.59	<code>getvariablefillvaluef32()</code>	539
15.48.2.60	<code>getvariablefillvaluef64()</code>	540
15.48.2.61	<code>getvariablefillvaluei16()</code>	540
15.48.2.62	<code>getvariablefillvaluei32()</code>	540
15.48.2.63	<code>getvariablefillvaluei8()</code>	540
15.48.2.64	<code>getvariableids()</code>	540
15.48.2.65	<code>getvariablename()</code>	540
15.48.2.66	<code>getvariables()</code>	541
15.48.2.67	<code>getvariablesshape()</code>	541
15.48.2.68	<code>hasattribute()</code>	541
15.48.2.69	<code>hasdimension()</code>	541
15.48.2.70	<code>hasvariable()</code>	541
15.48.2.71	<code>initncdataset()</code>	542
15.48.2.72	<code>initncdimension()</code>	542
15.48.2.73	<code>initncvariable()</code>	542
15.48.2.74	<code>isdatasetunlimited()</code>	542
15.48.2.75	<code>isunlimiteddimension()</code>	543
15.48.2.76	<code>isunlimitedvariable()</code>	543
15.48.2.77	<code>newncdataset()</code>	543
15.48.2.78	<code>newncdimension()</code>	543
15.48.2.79	<code>newncvariable()</code>	543
15.48.2.80	<code>setdata1df32()</code>	543
15.48.2.81	<code>setdata1df64()</code>	544
15.48.2.82	<code>setdata1di16()</code>	544
15.48.2.83	<code>setdata1di32()</code>	545

15.48.2.84	setdata1di8()	545
15.48.2.85	setdata2df32()	546
15.48.2.86	setdata2df64()	546
15.48.2.87	setdata2di16()	547
15.48.2.88	setdata2di32()	547
15.48.2.89	setdata2di8()	548
15.48.2.90	setdata3df32()	548
15.48.2.91	setdata3df64()	549
15.48.2.92	setdata3di16()	549
15.48.2.93	setdata3di32()	550
15.48.2.94	setdata3di8()	550
15.48.2.95	setdata4df32()	551
15.48.2.96	setdata4df64()	551
15.48.2.97	setdata4di16()	552
15.48.2.98	setdata4di32()	552
15.48.2.99	setdata4di8()	553
15.48.2.100	setdata5df32()	553
15.48.2.101	setdata5df64()	554
15.48.2.102	setdata5di16()	554
15.48.2.103	setdata5di32()	555
15.48.2.104	setdata5di8()	555
15.48.2.105	setdatascalarf32()	556
15.48.2.106	setdatascalarf64()	556
15.48.2.107	setdatascalar16()	557
15.48.2.108	setdatascalar32()	557
15.48.2.109	setdatascalar8()	557
15.48.2.110	setdimension()	558
15.48.2.111	setglobalattributechar()	558
15.48.2.112	setglobalattribute32()	559
15.48.2.113	setglobalattribute64()	559
15.48.2.114	setglobalattribute16()	559
15.48.2.115	setglobalattribute32()	560
15.48.2.116	setglobalattribute8()	560
15.48.2.117	setvariableattributechar()	561
15.48.2.118	setvariableattribute32()	561
15.48.2.119	setvariableattribute64()	561
15.48.2.120	setvariableattribute16()	562
15.48.2.121	setvariableattribute32()	562
15.48.2.122	setvariableattribute8()	563
15.48.2.123	setvariablefillvalue32()	563

15.48.2.124	setvariablefillvaluef64()	563
15.48.2.125	setvariablefillvaluei16()	563
15.48.2.126	setvariablefillvaluei32()	563
15.48.2.127	setvariablefillvaluei8()	564
15.48.2.128	setvariablewithids()	564
15.48.2.129	setvariablewithnames()	565
15.48.2.130	setvariablewithtypes()	565
15.49	mo_neutrons Module Reference	566
15.49.1	Detailed Description	566
15.49.2	Function/Subroutine Documentation	566
15.49.2.1	approx_mon_int()	567
15.49.2.2	approx_mon_int_eps()	567
15.49.2.3	approx_mon_int_steps()	568
15.49.2.4	cosmic()	568
15.49.2.5	desiletsn0()	570
15.49.2.6	intgrandfast()	572
15.49.2.7	lookupintegral()	572
15.49.2.8	oldintegration()	573
15.49.2.9	tabularintegralafast()	573
15.50	mo_nml Module Reference	574
15.50.1	Detailed Description	575
15.50.2	Function/Subroutine Documentation	575
15.50.2.1	close_nml()	575
15.50.2.2	open_nml()	576
15.50.2.3	position_nml()	577
15.50.3	Variable Documentation	579
15.50.3.1	length_error	579
15.50.3.2	missing	579
15.50.3.3	nunitnml	579
15.50.3.4	positioned	579
15.50.3.5	read_error	579
15.51	mo_objective_function Module Reference	579
15.51.1	Detailed Description	580
15.51.2	Function/Subroutine Documentation	580
15.51.2.1	extract_basin_avg_tws()	581
15.51.2.2	objective()	582
15.51.2.3	objective_et_kge_catchment_avg()	584
15.51.2.4	objective_kge_q_et()	587
15.51.2.5	objective_kge_q_rmse_et()	589
15.51.2.6	objective_kge_q_rmse_tws()	591

15.51.2.7 objective_kge_q_sm_corr()	593
15.51.2.8 objective_neutrons_kge_catchment_avg()	595
15.51.2.9 objective_sm_corr()	598
15.51.2.10 objective_sm_kge_catchment_avg()	601
15.51.2.11 objective_sm_pd()	604
15.51.2.12 objective_sm_sse_standard_score()	607
15.52 mo_optimization Module Reference	610
15.52.1 Detailed Description	610
15.52.2 Function/Subroutine Documentation	610
15.52.2.1 optimization()	610
15.53 mo_orderpack Module Reference	613
15.53.1 Detailed Description	615
15.53.2 Function/Subroutine Documentation	615
15.53.2.1 d_ctrper()	615
15.53.2.2 d_fndnth()	615
15.53.2.3 d_indmed()	615
15.53.2.4 d_indnth()	616
15.53.2.5 d_inspar()	616
15.53.2.6 d_inssor()	616
15.53.2.7 d_med()	616
15.53.2.8 d_median()	617
15.53.2.9 d_mrgref()	617
15.53.2.10 d_mrgrnk()	617
15.53.2.11 d_mulcnt()	617
15.53.2.12 d_nearless()	617
15.53.2.13 d_rapknr()	617
15.53.2.14 d_refpar()	618
15.53.2.15 d_refsor()	618
15.53.2.16 d_rinpar()	618
15.53.2.17 d_rnkpar()	618
15.53.2.18 d_subsor()	619
15.53.2.19 d_uniinv()	619
15.53.2.20 d_unipar()	619
15.53.2.21 d_unirnk()	619
15.53.2.22 d_unista()	619
15.53.2.23 d_valmed()	620
15.53.2.24 d_valnth()	620
15.53.2.25 d_ctrper()	620
15.53.2.26 d_fndnth()	620
15.53.2.27 d_indmed()	620

15.53.2.28_indnth()	620
15.53.2.29_inspar()	621
15.53.2.30_inssor()	621
15.53.2.31i_med()	621
15.53.2.32_median()	622
15.53.2.33_mrgref()	622
15.53.2.34_mrgnrnk()	622
15.53.2.35_mulcnt()	622
15.53.2.36_nearless()	622
15.53.2.37_rapknr()	622
15.53.2.38_refpar()	622
15.53.2.39_refsor()	623
15.53.2.40_rinpar()	623
15.53.2.41i_rnkpar()	623
15.53.2.42_subsor()	623
15.53.2.43_uniinv()	624
15.53.2.44_unipar()	624
15.53.2.45_unirnk()	624
15.53.2.46_unista()	624
15.53.2.47_valmed()	624
15.53.2.48_valnth()	625
15.53.2.49_ctrper()	625
15.53.2.50_fndnth()	625
15.53.2.51r_indmed()	625
15.53.2.52_indnth()	625
15.53.2.53_inspar()	625
15.53.2.54_inssor()	626
15.53.2.55r_med()	626
15.53.2.56r_median()	626
15.53.2.57r_mrgref()	627
15.53.2.58r_mrgnrnk()	627
15.53.2.59r_mulcnt()	627
15.53.2.60r_nearless()	627
15.53.2.61r_rapknr()	627
15.53.2.62r_refpar()	627
15.53.2.63r_refsor()	627
15.53.2.64r_rinpar()	628
15.53.2.65r_rnkpar()	628
15.53.2.66r_subsor()	628
15.53.2.67r_uniinv()	629

15.53.2.68	<code>_unipar()</code>	629
15.53.2.69	<code>_unirnk()</code>	629
15.53.2.70	<code>_unista()</code>	629
15.53.2.71	<code>_valmed()</code>	629
15.53.2.72	<code>_valnth()</code>	629
15.53.2.73	<code>sort_index_dp()</code>	629
15.53.2.74	<code>sort_index_i4()</code>	630
15.53.2.75	<code>sort_index_sp()</code>	630
15.53.3	Variable Documentation	630
15.53.3.1	<code>idont</code>	630
15.54	<code>mo_percentile</code> Module Reference	630
15.54.1	Function/Subroutine Documentation	630
15.54.1.1	<code>median_dp()</code>	631
15.54.1.2	<code>median_sp()</code>	631
15.54.1.3	<code>n_element_dp()</code>	631
15.54.1.4	<code>n_element_sp()</code>	631
15.54.1.5	<code>percentile_0d_dp()</code>	631
15.54.1.6	<code>percentile_0d_sp()</code>	631
15.54.1.7	<code>percentile_1d_dp()</code>	632
15.54.1.8	<code>percentile_1d_sp()</code>	632
15.54.1.9	<code>qmedian_dp()</code>	632
15.54.1.10	<code>qmedian_sp()</code>	632
15.55	<code>mo_pet</code> Module Reference	632
15.55.1	Detailed Description	633
15.55.2	Function/Subroutine Documentation	633
15.55.2.1	<code>extraterr_rad_approx()</code>	633
15.55.2.2	<code>pet_hargreaves()</code>	635
15.55.2.3	<code>pet_penman()</code>	637
15.55.2.4	<code>pet_priestly()</code>	639
15.55.2.5	<code>sat_vap_pressure()</code>	641
15.55.2.6	<code>slope_satpressure()</code>	642
15.56	<code>mo_prepare_gridded_lai</code> Module Reference	644
15.56.1	Detailed Description	644
15.56.2	Function/Subroutine Documentation	645
15.56.2.1	<code>prepare_gridded_daily_lai_data()</code>	645
15.56.2.2	<code>prepare_gridded_mean_monthly_lai_data()</code>	646
15.57	<code>mo_read_config</code> Module Reference	647
15.57.1	Detailed Description	647
15.57.2	Function/Subroutine Documentation	647
15.57.2.1	<code>in_bound()</code>	648

15.57.2.2 read_config()	648
15.58mo_read_forcing_nc Module Reference	649
15.58.1 Detailed Description	650
15.58.2 Function/Subroutine Documentation	650
15.58.2.1 get_time()	650
15.58.2.2 read_forcing_nc()	652
15.58.2.3 read_weights_nc()	654
15.59mo_read_latlon Module Reference	656
15.59.1 Detailed Description	656
15.59.2 Function/Subroutine Documentation	656
15.59.2.1 read_latlon()	657
15.60mo_read_lut Module Reference	658
15.60.1 Detailed Description	658
15.60.2 Function/Subroutine Documentation	658
15.60.2.1 read_geoformation_lut()	658
15.60.2.2 read_lai_lut()	659
15.61mo_read_meteo Module Reference	660
15.61.1 Detailed Description	660
15.61.2 Function/Subroutine Documentation	661
15.61.2.1 read_meteo_bin()	661
15.62mo_read_optional_data Module Reference	662
15.62.1 Detailed Description	663
15.62.2 Function/Subroutine Documentation	663
15.62.2.1 read_basin_avg_tws()	663
15.62.2.2 read_evapotranspiration()	664
15.62.2.3 read_neutrons()	665
15.62.2.4 read_soil_moisture()	666
15.63mo_read_spatial_data Module Reference	667
15.63.1 Detailed Description	668
15.63.2 Function/Subroutine Documentation	668
15.63.2.1 read_header_ascii()	668
15.63.2.2 read_spatial_data_ascii_dp()	669
15.63.2.3 read_spatial_data_ascii_i4()	670
15.64mo_read_timeseries Module Reference	670
15.64.1 Detailed Description	670
15.64.2 Function/Subroutine Documentation	671
15.64.2.1 read_timeseries()	671
15.65mo_read_wrapper Module Reference	672
15.65.1 Detailed Description	672
15.65.2 Function/Subroutine Documentation	673

15.65.2.1 check_consistency_lut_map()	673
15.65.2.2 read_data()	674
15.66mo_restart Module Reference	675
15.66.1 Detailed Description	675
15.66.2 Function/Subroutine Documentation	676
15.66.2.1 read_restart_config()	676
15.66.2.2 read_restart_states()	677
15.66.2.3 write_restart_files()	679
15.67mo_runoff Module Reference	681
15.67.1 Detailed Description	681
15.67.2 Function/Subroutine Documentation	681
15.67.2.1 l1_total_runoff()	681
15.67.2.2 runoff_sat_zone()	683
15.67.2.3 runoff_unsat_zone()	684
15.68mo_sce Module Reference	686
15.68.1 Detailed Description	687
15.68.2 Function/Subroutine Documentation	687
15.68.2.1 cce()	687
15.68.2.2 chkcst()	688
15.68.2.3 comp()	688
15.68.2.4 getpnt()	689
15.68.2.5 parstt()	690
15.68.2.6 sce()	690
15.68.2.7 sort_matrix()	694
15.69mo_set_netcdf_outputs Module Reference	695
15.69.1 Detailed Description	695
15.69.2 Function/Subroutine Documentation	695
15.69.2.1 set_netcdf()	695
15.70mo_snow_accum_melt Module Reference	696
15.70.1 Detailed Description	696
15.70.2 Function/Subroutine Documentation	696
15.70.2.1 snow_accum_melt()	696
15.71mo_soil_database Module Reference	698
15.71.1 Detailed Description	698
15.71.2 Function/Subroutine Documentation	699
15.71.2.1 generate_soil_database()	699
15.71.2.2 read_soil_lut()	700
15.72mo_soil_moisture Module Reference	701
15.72.1 Detailed Description	701
15.72.2 Function/Subroutine Documentation	702

15.72.2.1 feddes_et_reduction()	702
15.72.2.2 jarvis_et_reduction()	703
15.72.2.3 soil_moisture()	705
15.73mo_spatial_agg_disagg_forcing Module Reference	708
15.73.1 Detailed Description	709
15.73.2 Function/Subroutine Documentation	709
15.73.2.1 spatial_aggregation_3d()	709
15.73.2.2 spatial_aggregation_4d()	709
15.73.2.3 spatial_disaggregation_3d()	709
15.73.2.4 spatial_disaggregation_4d()	710
15.74mo_spatialsimilarity Module Reference	710
15.74.1 Detailed Description	710
15.74.2 Function/Subroutine Documentation	711
15.74.2.1 nndv_dp()	711
15.74.2.2 nndv_sp()	711
15.74.2.3 pd_dp()	711
15.74.2.4 pd_sp()	711
15.75mo_standard_score Module Reference	711
15.75.1 Detailed Description	712
15.75.2 Function/Subroutine Documentation	712
15.75.2.1 classified_standard_score_dp()	712
15.75.2.2 classified_standard_score_sp()	712
15.75.2.3 standard_score_dp()	712
15.75.2.4 standard_score_sp()	713
15.76mo_startup Module Reference	713
15.76.1 Detailed Description	713
15.76.2 Function/Subroutine Documentation	713
15.76.2.1 constants_init()	713
15.76.2.2 initialise()	714
15.76.2.3 l0_check_input()	716
15.76.2.4 l0_variable_init()	717
15.76.2.5 l1_variable_init()	718
15.76.2.6 l2_variable_init()	719
15.77mo_string_utils Module Reference	720
15.77.1 Detailed Description	721
15.77.2 Function/Subroutine Documentation	721
15.77.2.1 compress()	721
15.77.2.2 divide_string()	722
15.77.2.3 dp2str()	723
15.77.2.4 equalstrings()	723

15.77.2.5 i42str()	724
15.77.2.6 i4array2str()	724
15.77.2.7 i82str()	724
15.77.2.8 log2str()	724
15.77.2.9 nonull()	724
15.77.2.10sp2str()	725
15.77.2.11splitstring()	725
15.77.2.12startswith()	725
15.77.2.13str2num()	726
15.77.2.14tolower()	726
15.77.2.15toupper()	727
15.77.3 Variable Documentation	727
15.77.3.1 separator	728
15.78mo_template Module Reference	728
15.78.1 Detailed Description	728
15.78.2 Function/Subroutine Documentation	728
15.78.2.1 circum()	729
15.78.2.2 mean_dp()	729
15.78.2.3 mean_sp()	729
15.78.3 Variable Documentation	729
15.78.3.1 itest	729
15.78.3.2 pi_dp	730
15.78.3.3 pi_sp	730
15.79mo_temporal_aggregation Module Reference	730
15.79.1 Detailed Description	730
15.79.2 Function/Subroutine Documentation	730
15.79.2.1 day2mon_average_dp()	731
15.79.2.2 hour2day_average_dp()	731
15.80mo_temporal_disagg_forcing Module Reference	731
15.80.1 Detailed Description	732
15.80.2 Function/Subroutine Documentation	732
15.80.2.1 temporal_disagg_forcing()	732
15.81mo_timer Module Reference	734
15.81.1 Detailed Description	735
15.81.2 Function/Subroutine Documentation	735
15.81.2.1 timer_check()	735
15.81.2.2 timer_clear()	736
15.81.2.3 timer_get()	736
15.81.2.4 timer_print()	738
15.81.2.5 timer_start()	739

15.81.2.6 timer_stop()	740
15.81.2.7 timers_init()	742
15.81.3 Variable Documentation	742
15.81.3.1 clock_rate	742
15.81.3.2 cputime	742
15.81.3.3 cycles1	743
15.81.3.4 cycles2	743
15.81.3.5 cycles_max	743
15.81.3.6 max_timers	743
15.81.3.7 status	743
15.82mo_upscaling_operators Module Reference	743
15.82.1 Detailed Description	744
15.82.2 Function/Subroutine Documentation	744
15.82.2.1 l0_fractionalcover_in_lx()	744
15.82.2.2 majority_statistics()	746
15.82.2.3 upscale_arithmetic_mean()	746
15.82.2.4 upscale_geometric_mean()	749
15.82.2.5 upscale_harmonic_mean()	750
15.83mo_utils Module Reference	751
15.83.1 Detailed Description	753
15.83.2 Function/Subroutine Documentation	753
15.83.2.1 equal_dp()	753
15.83.2.2 equal_sp()	753
15.83.2.3 greaterqual_dp()	753
15.83.2.4 greaterqual_sp()	753
15.83.2.5 is_finite_dp()	753
15.83.2.6 is_finite_sp()	754
15.83.2.7 is_nan_dp()	754
15.83.2.8 is_nan_sp()	754
15.83.2.9 is_normal_dp()	754
15.83.2.10 is_normal_sp()	754
15.83.2.11 lesserequal_dp()	754
15.83.2.12 lesserequal_sp()	754
15.83.2.13 locate_0d_dp()	754
15.83.2.14 locate_0d_sp()	755
15.83.2.15 locate_1d_dp()	755
15.83.2.16 locate_1d_sp()	755
15.83.2.17 notequal_dp()	755
15.83.2.18 notequal_sp()	755
15.83.2.19 special_value_dp()	755

15.83.2.20	special_value_sp()	756
15.83.2.21	swap_vec_dp()	756
15.83.2.22	swap_vec_i4()	756
15.83.2.23	swap_vec_sp()	757
15.83.2.24	swap_xy_dp()	757
15.83.2.25	swap_xy_i4()	757
15.83.2.26	swap_xy_sp()	757
15.84	mo_write_ascii Module Reference	757
15.84.1	Detailed Description	757
15.84.2	Function/Subroutine Documentation	758
15.84.2.1	write_configfile()	758
15.84.2.2	write_optifile()	759
15.84.2.3	write_optinamelist()	760
15.85	mo_write_fluxes_states Module Reference	760
15.85.1	Detailed Description	761
15.85.2	Function/Subroutine Documentation	761
15.85.2.1	close()	762
15.85.2.2	createoutputfile()	762
15.85.2.3	fluxesunit()	764
15.85.2.4	geocoordinates()	765
15.85.2.5	mapcoordinates()	766
15.85.2.6	newoutputdataset()	767
15.85.2.7	newoutputvariable()	769
15.85.2.8	updatedataset()	770
15.85.2.9	updatevariable()	771
15.85.2.10	writetimestep()	772
15.85.2.11	writevariableattributes()	773
15.85.2.12	writevariabletimestep()	774
15.86	mo_xor4096 Module Reference	775
15.86.1	Function/Subroutine Documentation	775
15.86.1.1	get_timeseed_i4_0d()	775
15.86.1.2	get_timeseed_i4_1d()	776
15.86.1.3	get_timeseed_i8_0d()	776
15.86.1.4	get_timeseed_i8_1d()	776
15.86.1.5	xor4096d_0d()	776
15.86.1.6	xor4096d_1d()	776
15.86.1.7	xor4096f_0d()	776
15.86.1.8	xor4096f_1d()	777
15.86.1.9	xor4096gd_0d()	777
15.86.1.10	xor4096gd_1d()	777

15.86.1.11	xor4096gf_0d()	777
15.86.1.12	xor4096gf_1d()	777
15.86.1.13	xor4096l_0d()	778
15.86.1.14	xor4096l_1d()	778
15.86.1.15	xor4096s_0d()	778
15.86.1.16	xor4096s_1d()	778
15.86.2	Variable Documentation	778
15.86.2.1	n_save_state	778
16	Data Type Documentation	779
16.1	mo_moment::absdev Interface Reference	779
16.1.1	Member Function/Subroutine Documentation	779
16.1.1.1	absdev_dp()	779
16.1.1.2	absdev_sp()	779
16.2	mo_anneal::anneal Interface Reference	779
16.2.1	Detailed Description	780
16.2.2	Member Function/Subroutine Documentation	781
16.2.2.1	anneal_dp()	781
16.3	mo_append::append Interface Reference	782
16.3.1	Detailed Description	782
16.3.2	Member Function/Subroutine Documentation	783
16.3.2.1	append_char_3d()	783
16.3.2.2	append_char_m_m()	783
16.3.2.3	append_char_v_s()	783
16.3.2.4	append_char_v_v()	783
16.3.2.5	append_dp_3d()	784
16.3.2.6	append_dp_m_m()	784
16.3.2.7	append_dp_v_s()	784
16.3.2.8	append_dp_v_v()	784
16.3.2.9	append_i4_m_m()	784
16.3.2.10	append_i4_v_s()	784
16.3.2.11	append_i4_v_v()	784
16.3.2.12	append_i8_3d()	785
16.3.2.13	append_i8_m_m()	785
16.3.2.14	append_i8_v_s()	785
16.3.2.15	append_i8_v_v()	785
16.3.2.16	append_lgt_3d()	785
16.3.2.17	append_lgt_m_m()	785
16.3.2.18	append_lgt_v_s()	786
16.3.2.19	append_lgt_v_v()	786

16.3.2.20	append_sp_3d()	786
16.3.2.21	append_sp_m_m()	786
16.3.2.22	append_sp_v_s()	786
16.3.2.23	append_sp_v_v()	786
16.4	mo_corr::arth Interface Reference	786
16.4.1	Member Function/Subroutine Documentation	787
16.4.1.1	arth_dp()	787
16.4.1.2	arth_i4()	787
16.4.1.3	arth_sp()	787
16.5	mo_ncwrite::attribute Type Reference	788
16.5.1	Member Data Documentation	788
16.5.1.1	name	788
16.5.1.2	nvalues	788
16.5.1.3	values	788
16.5.1.4	xtype	788
16.6	mo_corr::autocoeffk Interface Reference	789
16.6.1	Member Function/Subroutine Documentation	789
16.6.1.1	autocoeffk_1d_dp()	789
16.6.1.2	autocoeffk_1d_sp()	789
16.6.1.3	autocoeffk_dp()	789
16.6.1.4	autocoeffk_sp()	789
16.7	mo_corr::autocorr Interface Reference	790
16.7.1	Member Function/Subroutine Documentation	790
16.7.1.1	autocorr_1d_dp()	790
16.7.1.2	autocorr_1d_sp()	790
16.7.1.3	autocorr_dp()	790
16.7.1.4	autocorr_sp()	790
16.8	mo_moment::average Interface Reference	790
16.8.1	Member Function/Subroutine Documentation	791
16.8.1.1	average_dp()	791
16.8.1.2	average_sp()	791
16.9	mo_global_variables::basininfo Type Reference	791
16.9.1	Member Data Documentation	792
16.9.1.1	l0_iend	792
16.9.1.2	l0_iendmask	792
16.9.1.3	l0_istart	792
16.9.1.4	l0_istartmask	792
16.9.1.5	l0_mask	792
16.9.1.6	l1_iend	793
16.9.1.7	l1_iendmask	793

16.9.1.8	<code>l1_istart</code>	793
16.9.1.9	<code>l1_istartmask</code>	793
16.9.1.10	<code>l1_mask</code>	793
16.9.1.11	<code>l2_iend</code>	793
16.9.1.12	<code>l2_iendmask</code>	793
16.9.1.13	<code>l2_istart</code>	793
16.9.1.14	<code>l2_istartmask</code>	793
16.9.1.15	<code>l2_mask</code>	794
16.10	<code>mo_mrm_global_variables::basininfo</code> Type Reference	794
16.10.1	Member Data Documentation	795
16.10.1.1	<code>gaugeidlist</code>	795
16.10.1.2	<code>gaugeindexlist</code>	795
16.10.1.3	<code>gaugenodelist</code>	795
16.10.1.4	<code>inflowgaugeheadwater</code>	795
16.10.1.5	<code>inflowgaugeidlist</code>	795
16.10.1.6	<code>inflowgaugeindexlist</code>	795
16.10.1.7	<code>inflowgaugenodelist</code>	796
16.10.1.8	<code>l0_coloutlet</code>	796
16.10.1.9	<code>l0_iend</code>	796
16.10.1.10	<code>l0_iendmask</code>	796
16.10.1.11	<code>l0_istart</code>	796
16.10.1.12	<code>l0_istartmask</code>	796
16.10.1.13	<code>l0_mask</code>	796
16.10.1.14	<code>l0_noutlet</code>	796
16.10.1.15	<code>l0_rowoutlet</code>	796
16.10.1.16	<code>l1_iend</code>	797
16.10.1.17	<code>l1_istart</code>	797
16.10.1.18	<code>l1_iend</code>	797
16.10.1.19	<code>l1_iendmask</code>	797
16.10.1.20	<code>l1_istart</code>	797
16.10.1.21	<code>l1_istartmask</code>	797
16.10.1.22	<code>l1_mask</code>	797
16.10.1.23	<code>l1_iend</code>	797
16.10.1.24	<code>l1_iendmask</code>	797
16.10.1.25	<code>l1_istart</code>	798
16.10.1.26	<code>l1_istartmask</code>	798
16.10.1.27	<code>l1_mask</code>	798
16.10.1.28	<code>ngauges</code>	798
16.10.1.29	<code>ninflowgauges</code>	798
16.11	<code>mo_errormeasures::bias</code> Interface Reference	798

16.11.1 Member Function/Subroutine Documentation	798
16.11.1.1 bias_dp_1d()	798
16.11.1.2 bias_dp_2d()	799
16.11.1.3 bias_dp_3d()	799
16.11.1.4 bias_sp_1d()	799
16.11.1.5 bias_sp_2d()	799
16.11.1.6 bias_sp_3d()	799
16.12mo_moment::central_moment Interface Reference	799
16.12.1 Member Function/Subroutine Documentation	800
16.12.1.1 central_moment_dp()	800
16.12.1.2 central_moment_sp()	800
16.13mo_moment::central_moment_var Interface Reference	800
16.13.1 Member Function/Subroutine Documentation	800
16.13.1.1 central_moment_var_dp()	800
16.13.1.2 central_moment_var_sp()	800
16.14mo_standard_score::classified_standard_score Interface Reference	801
16.14.1 Detailed Description	801
16.14.2 Member Function/Subroutine Documentation	801
16.14.2.1 classified_standard_score_dp()	802
16.14.2.2 classified_standard_score_sp()	802
16.15mo_corr::corr Interface Reference	802
16.15.1 Member Function/Subroutine Documentation	802
16.15.1.1 corr_dp()	802
16.15.1.2 corr_sp()	802
16.16mo_moment::correlation Interface Reference	803
16.16.1 Member Function/Subroutine Documentation	803
16.16.1.1 correlation_dp()	803
16.16.1.2 correlation_sp()	803
16.17mo_moment::covariance Interface Reference	803
16.17.1 Member Function/Subroutine Documentation	803
16.17.1.1 covariance_dp()	803
16.17.1.2 covariance_sp()	804
16.18mo_corr::crosscoeffk Interface Reference	804
16.18.1 Member Function/Subroutine Documentation	804
16.18.1.1 crosscoeffk_dp()	804
16.18.1.2 crosscoeffk_sp()	804
16.19mo_corr::crosscorr Interface Reference	804
16.19.1 Member Function/Subroutine Documentation	804
16.19.1.1 crosscorr_dp()	805
16.19.1.2 crosscorr_sp()	805

16.20mo_orderpack::ctrper Interface Reference	805
16.20.1 Member Function/Subroutine Documentation	805
16.20.1.1 d_ctrper()	805
16.20.1.2 i_ctrper()	805
16.20.1.3 r_ctrper()	805
16.21mo_temporal_aggregation::day2mon_average Interface Reference	806
16.21.1 Detailed Description	806
16.21.2 Member Function/Subroutine Documentation	806
16.21.2.1 day2mon_average_dp()	806
16.22mo_ncwrite::dims Type Reference	807
16.22.1 Member Data Documentation	807
16.22.1.1 dimid	807
16.22.1.2 len	807
16.22.1.3 name	808
16.23mo_ncwrite::dump_netcdf Interface Reference	808
16.23.1 Member Function/Subroutine Documentation	808
16.23.1.1 dump_netcdf_1d_dp()	808
16.23.1.2 dump_netcdf_1d_i4()	808
16.23.1.3 dump_netcdf_1d_sp()	809
16.23.1.4 dump_netcdf_2d_dp()	809
16.23.1.5 dump_netcdf_2d_i4()	809
16.23.1.6 dump_netcdf_2d_sp()	809
16.23.1.7 dump_netcdf_3d_dp()	809
16.23.1.8 dump_netcdf_3d_i4()	810
16.23.1.9 dump_netcdf_3d_sp()	810
16.23.1.10dump_netcdf_4d_dp()	810
16.23.1.11dump_netcdf_4d_i4()	810
16.23.1.12dump_netcdf_4d_sp()	810
16.23.1.13dump_netcdf_5d_dp()	811
16.23.1.14dump_netcdf_5d_i4()	811
16.23.1.15dump_netcdf_5d_sp()	811
16.24mo_utils::eq Interface Reference	811
16.24.1 Member Function/Subroutine Documentation	811
16.24.1.1 equal_dp()	812
16.24.1.2 equal_sp()	812
16.25mo_utils::equal Interface Reference	812
16.25.1 Detailed Description	812
16.25.2 Member Function/Subroutine Documentation	813
16.25.2.1 equal_dp()	813
16.25.2.2 equal_sp()	813

16.26mo_orderpack::fndnth Interface Reference	813
16.26.1 Member Function/Subroutine Documentation	813
16.26.1.1 d_fndnth()	813
16.26.1.2 i_fndnth()	813
16.26.1.3 r_fndnth()	814
16.27mo_corr::four1 Interface Reference	814
16.27.1 Member Function/Subroutine Documentation	814
16.27.1.1 four1_dp()	814
16.27.1.2 four1_sp()	814
16.28mo_corr::fourrow Interface Reference	814
16.28.1 Member Function/Subroutine Documentation	814
16.28.1.1 fourrow_dp()	815
16.28.1.2 fourrow_sp()	815
16.29mo_mrm_global_variables::gaugingstation Type Reference	815
16.29.1 Member Data Documentation	815
16.29.1.1 basinid	815
16.29.1.2 fname	816
16.29.1.3 gaugeid	816
16.29.1.4 q	816
16.30mo_utils::ge Interface Reference	816
16.30.1 Member Function/Subroutine Documentation	816
16.30.1.1 greaterequal_dp()	816
16.30.1.2 greaterequal_sp()	816
16.31mo_anneal::generate_neighborhood_weight Interface Reference	817
16.31.1 Member Function/Subroutine Documentation	817
16.31.1.1 generate_neighborhood_weight_dp()	817
16.32mo_ncread::get_ncvar Interface Reference	817
16.32.1 Member Function/Subroutine Documentation	818
16.32.1.1 get_ncvar_0d_dp()	818
16.32.1.2 get_ncvar_0d_i1()	818
16.32.1.3 get_ncvar_0d_i4()	818
16.32.1.4 get_ncvar_0d_sp()	818
16.32.1.5 get_ncvar_1d_dp()	818
16.32.1.6 get_ncvar_1d_i1()	819
16.32.1.7 get_ncvar_1d_i4()	819
16.32.1.8 get_ncvar_1d_sp()	819
16.32.1.9 get_ncvar_2d_dp()	819
16.32.1.10get_ncvar_2d_i1()	819
16.32.1.11get_ncvar_2d_i4()	820
16.32.1.12get_ncvar_2d_sp()	820

16.32.1.13	get_ncvar_3d_dp()	820
16.32.1.14	get_ncvar_3d_i1()	820
16.32.1.15	get_ncvar_3d_i4()	820
16.32.1.16	get_ncvar_3d_sp()	821
16.32.1.17	get_ncvar_4d_dp()	821
16.32.1.18	get_ncvar_4d_i1()	821
16.32.1.19	get_ncvar_4d_i4()	821
16.32.1.20	get_ncvar_4d_sp()	821
16.32.1.21	get_ncvar_5d_dp()	822
16.32.1.22	get_ncvar_5d_i1()	822
16.32.1.23	get_ncvar_5d_i4()	822
16.32.1.24	get_ncvar_5d_sp()	822
16.33	mo_xor4096::get_timeseed Interface Reference	823
16.33.1	Member Function/Subroutine Documentation	823
16.33.1.1	get_timeseed_i4_0d()	823
16.33.1.2	get_timeseed_i4_1d()	823
16.33.1.3	get_timeseed_i8_0d()	823
16.33.1.4	get_timeseed_i8_1d()	823
16.34	mo_anneal::gettemperature Interface Reference	823
16.34.1	Detailed Description	824
16.34.2	Member Function/Subroutine Documentation	825
16.34.2.1	gettemperature_dp()	825
16.35	mo_utils::greaterequal Interface Reference	825
16.35.1	Member Function/Subroutine Documentation	825
16.35.1.1	greaterequal_dp()	825
16.35.1.2	greaterequal_sp()	825
16.36	mo_global_variables::gridgeoref Type Reference	826
16.36.1	Member Data Documentation	826
16.36.1.1	cellsize	826
16.36.1.2	ncols	826
16.36.1.3	nodata_value	826
16.36.1.4	nrows	827
16.36.1.5	xllcorner	827
16.36.1.6	yllcorner	827
16.37	mo_mrm_global_variables::gridgeoref Type Reference	827
16.37.1	Member Data Documentation	828
16.37.1.1	cellsize	828
16.37.1.2	ncols	828
16.37.1.3	nodata_value	828
16.37.1.4	nrows	828

16.37.1.5 xllcorner	828
16.37.1.6 yllcorner	828
16.38mo_temporal_aggregation::hour2day_average Interface Reference	828
16.38.1 Detailed Description	829
16.38.2 Member Function/Subroutine Documentation	829
16.38.2.1 hour2day_average_dp()	829
16.39mo_orderpack::indmed Interface Reference	829
16.39.1 Member Function/Subroutine Documentation	830
16.39.1.1 d_indmed()	830
16.39.1.2 i_indmed()	830
16.39.1.3 r_indmed()	830
16.40mo_orderpack::indnth Interface Reference	830
16.40.1 Member Function/Subroutine Documentation	830
16.40.1.1 d_indnth()	830
16.40.1.2 i_indnth()	831
16.40.1.3 r_indnth()	831
16.41mo_orderpack::inspar Interface Reference	831
16.41.1 Member Function/Subroutine Documentation	831
16.41.1.1 d_inspar()	831
16.41.1.2 i_inspar()	831
16.41.1.3 r_inspar()	831
16.42mo_orderpack::inssor Interface Reference	832
16.42.1 Member Function/Subroutine Documentation	832
16.42.1.1 d_inssor()	832
16.42.1.2 i_inssor()	832
16.42.1.3 r_inssor()	832
16.43mo_utils::is_finite Interface Reference	832
16.43.1 Detailed Description	832
16.43.2 Member Function/Subroutine Documentation	833
16.43.2.1 is_finite_dp()	833
16.43.2.2 is_finite_sp()	833
16.44mo_utils::is_nan Interface Reference	833
16.44.1 Member Function/Subroutine Documentation	833
16.44.1.1 is_nan_dp()	833
16.44.1.2 is_nan_sp()	834
16.45mo_utils::is_normal Interface Reference	834
16.45.1 Member Function/Subroutine Documentation	834
16.45.1.1 is_normal_dp()	834
16.45.1.2 is_normal_sp()	834
16.46mo_errormeasures::kge Interface Reference	834

16.46.1 Detailed Description	835
16.46.2 Member Function/Subroutine Documentation	835
16.46.2.1 kge_dp_1d()	835
16.46.2.2 kge_dp_2d()	836
16.46.2.3 kge_dp_3d()	836
16.46.2.4 kge_sp_1d()	836
16.46.2.5 kge_sp_2d()	836
16.46.2.6 kge_sp_3d()	836
16.47mo_errormeasures::kgenocorr Interface Reference	836
16.47.1 Detailed Description	837
16.47.2 Member Function/Subroutine Documentation	837
16.47.2.1 kgenocorr_dp_1d()	837
16.47.2.2 kgenocorr_dp_2d()	838
16.47.2.3 kgenocorr_dp_3d()	838
16.47.2.4 kgenocorr_sp_1d()	838
16.47.2.5 kgenocorr_sp_2d()	838
16.47.2.6 kgenocorr_sp_3d()	838
16.48mo_moment::kurtosis Interface Reference	838
16.48.1 Member Function/Subroutine Documentation	839
16.48.1.1 kurtosis_dp()	839
16.48.1.2 kurtosis_sp()	839
16.49mo_utils::le Interface Reference	839
16.49.1 Member Function/Subroutine Documentation	839
16.49.1.1 lesserequal_dp()	839
16.49.1.2 lesserequal_sp()	839
16.50mo_utils::lesserequal Interface Reference	840
16.50.1 Member Function/Subroutine Documentation	840
16.50.1.1 lesserequal_dp()	840
16.50.1.2 lesserequal_sp()	840
16.51mo_linfit::linfit Interface Reference	840
16.51.1 Detailed Description	840
16.51.2 Member Function/Subroutine Documentation	841
16.51.2.1 linfit_dp()	841
16.51.2.2 linfit_sp()	841
16.52mo_errormeasures::lnnse Interface Reference	841
16.52.1 Member Function/Subroutine Documentation	842
16.52.1.1 lnnse_dp_1d()	842
16.52.1.2 lnnse_dp_2d()	842
16.52.1.3 lnnse_dp_3d()	842
16.52.1.4 lnnse_sp_1d()	842

16.52.1.5 Innse_sp_2d()	842
16.52.1.6 Innse_sp_3d()	842
16.53mo_utils::locate Interface Reference	843
16.53.1 Detailed Description	843
16.53.2 Member Function/Subroutine Documentation	843
16.53.2.1 locate_0d_dp()	843
16.53.2.2 locate_0d_sp()	844
16.53.2.3 locate_1d_dp()	844
16.53.2.4 locate_1d_sp()	844
16.54mo_errormeasures::mae Interface Reference	844
16.54.1 Member Function/Subroutine Documentation	844
16.54.1.1 mae_dp_1d()	844
16.54.1.2 mae_dp_2d()	844
16.54.1.3 mae_dp_3d()	845
16.54.1.4 mae_sp_1d()	845
16.54.1.5 mae_sp_2d()	845
16.54.1.6 mae_sp_3d()	845
16.55mo_mcmc::mcmc Interface Reference	845
16.55.1 Detailed Description	846
16.55.2 Member Function/Subroutine Documentation	849
16.55.2.1 mcmc_dp()	849
16.56mo_mcmc::mcmc_stddev Interface Reference	849
16.56.1 Detailed Description	850
16.56.2 Member Function/Subroutine Documentation	853
16.56.2.1 mcmc_stddev_dp()	853
16.57mo_moment::mean Interface Reference	853
16.57.1 Member Function/Subroutine Documentation	854
16.57.1.1 mean_dp()	854
16.57.1.2 mean_sp()	854
16.58mo_template::mean Interface Reference	854
16.58.1 Detailed Description	854
16.58.2 Member Function/Subroutine Documentation	855
16.58.2.1 mean_dp()	855
16.58.2.2 mean_sp()	855
16.59mo_percentile::median Interface Reference	855
16.59.1 Member Function/Subroutine Documentation	855
16.59.1.1 median_dp()	855
16.59.1.2 median_sp()	856
16.60mo_moment::mixed_central_moment Interface Reference	856
16.60.1 Member Function/Subroutine Documentation	856

16.60.1.1 mixed_central_moment_dp()	856
16.60.1.2 mixed_central_moment_sp()	856
16.61mo_moment::mixed_central_moment_var Interface Reference	856
16.61.1 Member Function/Subroutine Documentation	857
16.61.1.1 mixed_central_moment_var_dp()	857
16.61.1.2 mixed_central_moment_var_sp()	857
16.62mo_moment::moment Interface Reference	857
16.62.1 Member Function/Subroutine Documentation	857
16.62.1.1 moment_dp()	857
16.62.1.2 moment_sp()	858
16.63mo_orderpack::mrgref Interface Reference	858
16.63.1 Member Function/Subroutine Documentation	858
16.63.1.1 d_mrgref()	858
16.63.1.2 i_mrgref()	858
16.63.1.3 r_mrgref()	858
16.64mo_orderpack::mrgnrk Interface Reference	859
16.64.1 Member Function/Subroutine Documentation	859
16.64.1.1 d_mrgnrk()	859
16.64.1.2 i_mrgnrk()	859
16.64.1.3 r_mrgnrk()	859
16.65mo_errormeasures::mse Interface Reference	859
16.65.1 Member Function/Subroutine Documentation	859
16.65.1.1 mse_dp_1d()	860
16.65.1.2 mse_dp_2d()	860
16.65.1.3 mse_dp_3d()	860
16.65.1.4 mse_sp_1d()	860
16.65.1.5 mse_sp_2d()	860
16.65.1.6 mse_sp_3d()	860
16.66mo_orderpack::mulcnt Interface Reference	861
16.66.1 Member Function/Subroutine Documentation	861
16.66.1.1 d_mulcnt()	861
16.66.1.2 i_mulcnt()	861
16.66.1.3 r_mulcnt()	861
16.67mo_percentile::n_element Interface Reference	861
16.67.1 Member Function/Subroutine Documentation	861
16.67.1.1 n_element_dp()	862
16.67.1.2 n_element_sp()	862
16.68mo_netcdf::ncdataset Interface Reference	862
16.68.1 Detailed Description	864
16.68.2 Member Function/Subroutine Documentation	864

16.68.2.1	close()	864
16.68.2.2	getattribute()	865
16.68.2.3	getdimension()	865
16.68.2.4	getdimensionbyid()	865
16.68.2.5	getdimensionbyname()	866
16.68.2.6	getglobalattributechar()	866
16.68.2.7	getglobalattributef32()	866
16.68.2.8	getglobalattributef64()	866
16.68.2.9	getglobalattributei16()	866
16.68.2.10	getglobalattributei32()	866
16.68.2.11	getglobalattributei8()	866
16.68.2.12	getnovariables()	866
16.68.2.13	getunlimiteddimension()	866
16.68.2.14	getvariable()	867
16.68.2.15	getvariablebyname()	867
16.68.2.16	getvariableids()	867
16.68.2.17	getvariables()	868
16.68.2.18	hasdimension()	868
16.68.2.19	hasvariable()	868
16.68.2.20	initncdataset()	869
16.68.2.21	isunlimited()	869
16.68.2.22	setattribute()	869
16.68.2.23	setdimension()	869
16.68.2.24	setglobalattributechar()	870
16.68.2.25	setglobalattributef32()	870
16.68.2.26	setglobalattributef64()	870
16.68.2.27	setglobalattributei16()	870
16.68.2.28	setglobalattributei32()	870
16.68.2.29	setglobalattributei8()	870
16.68.2.30	setvariable()	871
16.68.2.31	setvariablewithids()	871
16.68.2.32	setvariablewithnames()	871
16.68.2.33	setvariablewithtypes()	871
16.68.3	Member Data Documentation	872
16.68.3.1	dataset	872
16.68.3.2	file	872
16.68.3.3	filename	872
16.68.3.4	fname	872
16.68.3.5	id	872
16.68.3.6	mode	872

16.68.3.7 netcdf	872
16.68.3.8 of	872
16.68.3.9 open	873
16.68.3.10opened	873
16.68.3.11the	873
16.69mo_netcdf::ncdimension Type Reference	873
16.69.1 Detailed Description	876
16.69.2 Member Function/Subroutine Documentation	876
16.69.2.1 getattribute()	876
16.69.2.2 getdata()	877
16.69.2.3 getdata1df32()	877
16.69.2.4 getdata1df64()	877
16.69.2.5 getdata1di16()	877
16.69.2.6 getdata1di32()	877
16.69.2.7 getdata1di8()	878
16.69.2.8 getdata2df32()	878
16.69.2.9 getdata2df64()	878
16.69.2.10getdata2di16()	878
16.69.2.11getdata2di32()	878
16.69.2.12getdata2di8()	878
16.69.2.13getdata3df32()	878
16.69.2.14getdata3df64()	878
16.69.2.15getdata3di16()	878
16.69.2.16getdata3di32()	879
16.69.2.17getdata3di8()	879
16.69.2.18getdata4df32()	879
16.69.2.19getdata4df64()	879
16.69.2.20getdata4di16()	879
16.69.2.21getdata4di32()	879
16.69.2.22getdata4di8()	879
16.69.2.23getdata5df32()	879
16.69.2.24getdata5df64()	879
16.69.2.25getdata5di16()	880
16.69.2.26getdata5di32()	880
16.69.2.27getdata5di8()	880
16.69.2.28getdatascalarf32()	880
16.69.2.29getdatascalarf64()	880
16.69.2.30getdatascalar16()	880
16.69.2.31getdatascalar32()	880
16.69.2.32getdatascalar8()	880

16.69.2.33	getdimensions()	880
16.69.2.34	getdtype()	881
16.69.2.35	getfillvalue()	881
16.69.2.36	getname()	881
16.69.2.37	getnodimensions()	881
16.69.2.38	getshape()	881
16.69.2.39	getvariableattributechar()	881
16.69.2.40	getvariableattributef32()	882
16.69.2.41	getvariableattributef64()	882
16.69.2.42	getvariableattributei16()	882
16.69.2.43	getvariableattributei32()	882
16.69.2.44	getvariableattributei8()	882
16.69.2.45	getvariablefillvaluef32()	882
16.69.2.46	getvariablefillvaluef64()	882
16.69.2.47	getvariablefillvaluei16()	882
16.69.2.48	getvariablefillvaluei32()	882
16.69.2.49	getvariablefillvaluei8()	883
16.69.2.50	hasattribute()	883
16.69.2.51	initncvariable()	883
16.69.2.52	unlimited()	883
16.69.2.53	setattribute()	883
16.69.2.54	setdata()	884
16.69.2.55	setdata1df32()	884
16.69.2.56	setdata1df64()	884
16.69.2.57	setdata1di16()	884
16.69.2.58	setdata1di32()	884
16.69.2.59	setdata1di8()	885
16.69.2.60	setdata2df32()	885
16.69.2.61	setdata2df64()	885
16.69.2.62	setdata2di16()	885
16.69.2.63	setdata2di32()	885
16.69.2.64	setdata2di8()	885
16.69.2.65	setdata3df32()	885
16.69.2.66	setdata3df64()	885
16.69.2.67	setdata3di16()	885
16.69.2.68	setdata3di32()	886
16.69.2.69	setdata3di8()	886
16.69.2.70	setdata4df32()	886
16.69.2.71	setdata4df64()	886
16.69.2.72	setdata4di16()	886

16.69.2.73	setdata4di32()	886
16.69.2.74	setdata4di8()	886
16.69.2.75	setdata5df32()	886
16.69.2.76	setdata5df64()	886
16.69.2.77	setdata5di16()	887
16.69.2.78	setdata5di32()	887
16.69.2.79	setdata5di8()	887
16.69.2.80	setdatascalarf32()	887
16.69.2.81	setdatascalarf64()	887
16.69.2.82	setdatascalarl16()	887
16.69.2.83	setdatascalarl32()	887
16.69.2.84	setdatascalarl8()	887
16.69.2.85	setfillvalue()	887
16.69.2.86	setvariableattributechar()	888
16.69.2.87	setvariableattributef32()	888
16.69.2.88	setvariableattributef64()	888
16.69.2.89	setvariableattributei16()	888
16.69.2.90	setvariableattributei32()	888
16.69.2.91	setvariableattributei8()	888
16.69.2.92	setvariablefillvaluef32()	889
16.69.2.93	setvariablefillvaluef64()	889
16.69.2.94	setvariablefillvaluei16()	889
16.69.2.95	setvariablefillvaluei32()	889
16.69.2.96	setvariablefillvaluei8()	889
16.69.3	Member Data Documentation	889
16.69.3.1	dimension [1/2]	889
16.69.3.2	dimension [2/2]	889
16.69.3.3	id	889
16.69.3.4	netcdf	889
16.69.3.5	parent	890
16.69.3.6	s	890
16.69.3.7	the [1/2]	890
16.69.3.8	the [2/2]	890
16.70	mo_netcdf::ncvariable Interface Reference	890
16.71	mo_utils::ne Interface Reference	890
16.71.1	Member Function/Subroutine Documentation	890
16.71.1.1	notequal_dp()	890
16.71.1.2	notequal_sp()	891
16.72	mo_orderpack::nearless Interface Reference	891
16.72.1	Member Function/Subroutine Documentation	891

16.72.1.1 d_nearless()	891
16.72.1.2 i_nearless()	891
16.72.1.3 r_nearless()	891
16.73mo_spatialsimilarity::nndv Interface Reference	891
16.73.1 Detailed Description	892
16.73.2 Member Function/Subroutine Documentation	893
16.73.2.1 nndv_dp()	893
16.73.2.2 nndv_sp()	893
16.74mo_utils::notequal Interface Reference	893
16.74.1 Member Function/Subroutine Documentation	893
16.74.1.1 notequal_dp()	894
16.74.1.2 notequal_sp()	894
16.75mo_errormeasures::nse Interface Reference	894
16.75.1 Member Function/Subroutine Documentation	894
16.75.1.1 nse_dp_1d()	894
16.75.1.2 nse_dp_2d()	894
16.75.1.3 nse_dp_3d()	895
16.75.1.4 nse_sp_1d()	895
16.75.1.5 nse_sp_2d()	895
16.75.1.6 nse_sp_3d()	895
16.76mo_string_utils::num2str Interface Reference	895
16.76.1 Detailed Description	895
16.76.2 Member Function/Subroutine Documentation	896
16.76.2.1 dp2str()	896
16.76.2.2 i42str()	896
16.76.2.3 i82str()	896
16.76.2.4 log2str()	897
16.76.2.5 sp2str()	897
16.77mo_string_utils::numarray2str Interface Reference	897
16.77.1 Detailed Description	897
16.77.2 Member Function/Subroutine Documentation	897
16.77.2.1 i4array2str()	898
16.78mo_orderpack::omedian Interface Reference	898
16.78.1 Member Function/Subroutine Documentation	898
16.78.1.1 d_median()	898
16.78.1.2 i_median()	898
16.78.1.3 r_median()	898
16.79mo_write_fluxes_states::outputdataset Interface Reference	899
16.79.1 Member Function/Subroutine Documentation	900
16.79.1.1 close()	900

16.79.1.2 updatedataset()	900
16.79.1.3 writetimestep()	900
16.79.2 Member Data Documentation	901
16.79.2.1 all	901
16.79.2.2 basin	901
16.79.2.3 count	901
16.79.2.4 counter	901
16.79.2.5 created	901
16.79.2.6 ibasin	901
16.79.2.7 id	901
16.79.2.8 nc	901
16.79.2.9 ncdataset	902
16.79.2.10steps	902
16.79.2.11store	902
16.79.2.12time	902
16.79.2.13to	902
16.79.2.14variables	902
16.79.2.15vars	902
16.79.2.16write	902
16.79.2.17written	902
16.80mo_mrm_write_fluxes_states::outputdataset Interface Reference	903
16.80.1 Member Function/Subroutine Documentation	904
16.80.1.1 close()	904
16.80.1.2 updatedataset()	904
16.80.1.3 writetimestep()	904
16.80.2 Member Data Documentation	904
16.80.2.1 all	904
16.80.2.2 basin	904
16.80.2.3 count	905
16.80.2.4 counter	905
16.80.2.5 created	905
16.80.2.6 ibasin	905
16.80.2.7 id	905
16.80.2.8 nc	905
16.80.2.9 ncdataset	905
16.80.2.10steps	905
16.80.2.11store	905
16.80.2.12time	906
16.80.2.13to	906
16.80.2.14variables	906

16.80.2.15vars	906
16.80.2.16write	906
16.80.2.17written	906
16.81mo_write_fluxes_states::outputvariable Interface Reference	907
16.81.1 Member Function/Subroutine Documentation	908
16.81.1.1 updatevariable()	908
16.81.1.2 writevariabletimestep()	908
16.81.2 Member Data Documentation	908
16.81.2.1 average	908
16.81.2.2 avg	908
16.81.2.3 before	909
16.81.2.4 between	909
16.81.2.5 calls	909
16.81.2.6 contains	909
16.81.2.7 count	909
16.81.2.8 counter	909
16.81.2.9 data [1/2]	909
16.81.2.10data [2/2]	909
16.81.2.11mask	909
16.81.2.12nc	910
16.81.2.13ncdataset	910
16.81.2.14number	910
16.81.2.15of	910
16.81.2.16reconstruct	910
16.81.2.17store	910
16.81.2.18the [1/3]	910
16.81.2.19the [2/3]	910
16.81.2.20the [3/3]	910
16.81.2.21to	911
16.81.2.22updatevariable	911
16.81.2.23variable	911
16.81.2.24which	911
16.81.2.25writes	911
16.81.2.26writing	911
16.82mo_mrm_write_fluxes_states::outputvariable Interface Reference	912
16.82.1 Member Function/Subroutine Documentation	913
16.82.1.1 updatevariable()	913
16.82.1.2 writevariabletimestep()	913
16.82.2 Member Data Documentation	913
16.82.2.1 average	913

16.82.2.2 avg	913
16.82.2.3 before	914
16.82.2.4 between	914
16.82.2.5 calls	914
16.82.2.6 contains	914
16.82.2.7 count	914
16.82.2.8 counter	914
16.82.2.9 data [1/2]	914
16.82.2.10data [2/2]	914
16.82.2.11mask	914
16.82.2.12nc	915
16.82.2.13ncdataset	915
16.82.2.14number	915
16.82.2.15of	915
16.82.2.16reconstruct	915
16.82.2.17store	915
16.82.2.18the [1/3]	915
16.82.2.19the [2/3]	915
16.82.2.20the [3/3]	915
16.82.2.21to	916
16.82.2.22updatevariable	916
16.82.2.23variable	916
16.82.2.24which	916
16.82.2.25writes	916
16.82.2.26writing	916
16.83mo_append::paste Interface Reference	916
16.83.1 Detailed Description	917
16.83.2 Member Function/Subroutine Documentation	917
16.83.2.1 paste_char_m_m()	917
16.83.2.2 paste_char_m_s()	917
16.83.2.3 paste_char_m_v()	918
16.83.2.4 paste_dp_m_m()	918
16.83.2.5 paste_dp_m_s()	918
16.83.2.6 paste_dp_m_v()	918
16.83.2.7 paste_i4_m_m()	918
16.83.2.8 paste_i4_m_s()	918
16.83.2.9 paste_i4_m_v()	919
16.83.2.10paste_i8_m_m()	919
16.83.2.11paste_i8_m_s()	919
16.83.2.12paste_i8_m_v()	919

16.83.2.13	paste_lgt_m_m()	919
16.83.2.14	paste_lgt_m_s()	919
16.83.2.15	paste_lgt_m_v()	919
16.83.2.16	paste_sp_m_m()	920
16.83.2.17	paste_sp_m_s()	920
16.83.2.18	paste_sp_m_v()	920
16.84	mo_spatialsimilarity::pd Interface Reference	920
16.84.1	Detailed Description	920
16.84.2	Member Function/Subroutine Documentation	921
16.84.2.1	pd_dp()	922
16.84.2.2	pd_sp()	922
16.85	mo_percentile::percentile Interface Reference	922
16.85.1	Member Function/Subroutine Documentation	922
16.85.1.1	percentile_0d_dp()	922
16.85.1.2	percentile_0d_sp()	922
16.85.1.3	percentile_1d_dp()	923
16.85.1.4	percentile_1d_sp()	923
16.86	mo_common_variables::period Type Reference	923
16.86.1	Member Data Documentation	924
16.86.1.1	dend	924
16.86.1.2	dstart	924
16.86.1.3	julend	924
16.86.1.4	julstart	924
16.86.1.5	mend	924
16.86.1.6	mstart	924
16.86.1.7	nobs	924
16.86.1.8	yend	925
16.86.1.9	ystart	925
16.87	mo_percentile::qmedian Interface Reference	925
16.87.1	Member Function/Subroutine Documentation	925
16.87.1.1	qmedian_dp()	925
16.87.1.2	qmedian_sp()	925
16.88	mo_orderpack::rapknr Interface Reference	925
16.88.1	Member Function/Subroutine Documentation	925
16.88.1.1	d_rapknr()	926
16.88.1.2	i_rapknr()	926
16.88.1.3	r_rapknr()	926
16.89	mo_read_spatial_data::read_spatial_data_ascii Interface Reference	926
16.89.1	Detailed Description	926
16.89.2	Member Function/Subroutine Documentation	927

16.89.2.1 read_spatial_data_ascii_dp()	927
16.89.2.2 read_spatial_data_ascii_i4()	927
16.90mo_corr::realft Interface Reference	927
16.90.1 Member Function/Subroutine Documentation	928
16.90.1.1 realft_dp()	928
16.90.1.2 realft_sp()	928
16.91mo_orderpack::refpar Interface Reference	928
16.91.1 Member Function/Subroutine Documentation	928
16.91.1.1 d_refpar()	928
16.91.1.2 i_refpar()	928
16.91.1.3 r_refpar()	929
16.92mo_orderpack::refsr Interface Reference	929
16.92.1 Member Function/Subroutine Documentation	929
16.92.1.1 d_refsr()	929
16.92.1.2 i_refsr()	929
16.92.1.3 r_refsr()	929
16.93mo_orderpack::rinpar Interface Reference	929
16.93.1 Member Function/Subroutine Documentation	930
16.93.1.1 d_rinpar()	930
16.93.1.2 i_rinpar()	930
16.93.1.3 r_rinpar()	930
16.94mo_errormeasures::rmse Interface Reference	930
16.94.1 Member Function/Subroutine Documentation	930
16.94.1.1 rmse_dp_1d()	931
16.94.1.2 rmse_dp_2d()	931
16.94.1.3 rmse_dp_3d()	931
16.94.1.4 rmse_sp_1d()	931
16.94.1.5 rmse_sp_2d()	931
16.94.1.6 rmse_sp_3d()	931
16.95mo_orderpack::rnkpar Interface Reference	932
16.95.1 Member Function/Subroutine Documentation	932
16.95.1.1 d_rnkpar()	932
16.95.1.2 i_rnkpar()	932
16.95.1.3 r_rnkpar()	932
16.96mo_errormeasures::sae Interface Reference	932
16.96.1 Member Function/Subroutine Documentation	933
16.96.1.1 sae_dp_1d()	933
16.96.1.2 sae_dp_2d()	933
16.96.1.3 sae_dp_3d()	933
16.96.1.4 sae_sp_1d()	933

16.96.1.5 sae_sp_2d()	933
16.96.1.6 sae_sp_3d()	933
16.97mo_julian::setcalendar Interface Reference	934
16.97.1 Member Function/Subroutine Documentation	934
16.97.1.1 setcalendarinteger()	934
16.97.1.2 setcalendarstring()	934
16.98mo_moment::skewness Interface Reference	935
16.98.1 Member Function/Subroutine Documentation	935
16.98.1.1 skewness_dp()	935
16.98.1.2 skewness_sp()	935
16.99mo_global_variables::soiltype Type Reference	936
16.99.1 Member Data Documentation	937
16.99.1.1 clay	937
16.99.1.2 db	937
16.99.1.3 dbm	937
16.99.1.4 depth	937
16.99.1.5 id	937
16.99.1.6 is_present	937
16.99.1.7 ks	937
16.99.1.8 ld	937
16.99.1.9 nhorizons	938
16.99.1.10ntillhorizons	938
16.99.1.11rzdepth	938
16.99.1.12sand	938
16.99.1.13thetafc	938
16.99.1.14thetafc_till	938
16.99.1.15hetapw	938
16.99.1.16hetapw_till	938
16.99.1.17hetas	938
16.99.1.18hetas_till	939
16.99.1.19ud	939
16.99.1.20wd	939
16.100mo_orderpack::sort Interface Reference	939
16.100.1 Detailed Description	939
16.100.2 Member Function/Subroutine Documentation	941
16.100.2.1d_refsor()	942
16.100.2.2_refsor()	942
16.100.2.3_refsor()	942
16.101mo_orderpack::sort_index Interface Reference	942
16.101.1 Member Function/Subroutine Documentation	942

16.101.1.1	sort_index_dp()	942
16.101.1.2	sort_index_i4()	942
16.101.1.3	sort_index_sp()	942
16.102	no_spatial_agg_disagg_forcing::spatial_aggregation Interface Reference	943
16.102.1	Detailed Description	943
16.102.2	Member Function/Subroutine Documentation	943
16.102.2.1	spatial_aggregation_3d()	943
16.102.2.2	spatial_aggregation_4d()	944
16.103	no_spatial_agg_disagg_forcing::spatial_disaggregation Interface Reference	944
16.103.1	Detailed Description	944
16.103.2	Member Function/Subroutine Documentation	944
16.103.2.1	spatial_disaggregation_3d()	945
16.103.2.2	spatial_disaggregation_4d()	945
16.104	no_utils::special_value Interface Reference	945
16.104.1	Detailed Description	945
16.104.2	Member Function/Subroutine Documentation	946
16.104.2.1	special_value_dp()	946
16.104.2.2	special_value_sp()	946
16.105	no_kind::sprs2_dp Type Reference	946
16.105.1	Detailed Description	947
16.105.2	Member Data Documentation	947
16.105.2.1	row	947
16.105.2.2	col	947
16.105.2.3	en	947
16.105.2.4	n	948
16.105.2.5	val	948
16.106	no_kind::sprs2_sp Type Reference	948
16.106.1	Detailed Description	948
16.106.2	Member Data Documentation	948
16.106.2.1	row	949
16.106.2.2	col	949
16.106.2.3	en	949
16.106.2.4	n	949
16.106.2.5	val	949
16.107	no_errormeasures::sse Interface Reference	949
16.107.1	Member Function/Subroutine Documentation	949
16.107.1.1	sse_dp_1d()	949
16.107.1.2	sse_dp_2d()	950
16.107.1.3	sse_dp_3d()	950
16.107.1.4	sse_sp_1d()	950

16.107.1.5sse_sp_2d()	950
16.107.1.6sse_sp_3d()	950
16.108.no_standard_score::standard_score Interface Reference	950
16.108.1.Detailed Description	951
16.108.2.Member Function/Subroutine Documentation	951
16.108.2.1standard_score_dp()	951
16.108.2.2standard_score_sp()	952
16.109.no_moment::stddev Interface Reference	952
16.109.1.Member Function/Subroutine Documentation	952
16.109.1.1stddev_dp()	952
16.109.1.2stddev_sp()	952
16.110.no_corr::swap Interface Reference	952
16.110.1.Member Function/Subroutine Documentation	952
16.110.1.1swap_1d_dpc()	953
16.110.1.2swap_1d_spc()	953
16.111.no_utils::swap Interface Reference	953
16.111.1.Detailed Description	953
16.111.2.Member Function/Subroutine Documentation	954
16.111.2.1swap_vec_dp()	954
16.111.2.2swap_vec_i4()	954
16.111.2.3swap_vec_sp()	954
16.111.2.4swap_xy_dp()	954
16.111.2.5swap_xy_i4()	954
16.111.2.6swap_xy_sp()	954
16.112.no_global_variables::twssstructure Type Reference	955
16.112.1.Member Data Documentation	955
16.112.1.1basinid	955
16.112.1.2name	955
16.112.1.3ws	955
16.113.no_orderpack::uniinv Interface Reference	956
16.113.1.Member Function/Subroutine Documentation	956
16.113.1.1d_uniinv()	956
16.113.1.2_uniinv()	956
16.113.1.3_uniinv()	956
16.114.no_orderpack::unipar Interface Reference	956
16.114.1.Member Function/Subroutine Documentation	956
16.114.1.1d_unipar()	957
16.114.1.2_unipar()	957
16.114.1.3_unipar()	957
16.115.no_orderpack::unirnk Interface Reference	957

16.115.1	Member Function/Subroutine Documentation	957
16.115.1.1	d_unirnk()	957
16.115.1.2	d_unirnk()	957
16.115.1.3	d_unirnk()	958
16.116	no_orderpack::unista Interface Reference	958
16.116.1	Member Function/Subroutine Documentation	958
16.116.1.1	d_unista()	958
16.116.1.2	d_unista()	958
16.116.1.3	d_unista()	958
16.117	no_orderpack::valmed Interface Reference	959
16.117.1	Member Function/Subroutine Documentation	959
16.117.1.1	d_valmed()	959
16.117.1.2	d_valmed()	959
16.117.1.3	d_valmed()	959
16.118	no_orderpack::valnth Interface Reference	959
16.118.1	Member Function/Subroutine Documentation	959
16.118.1.1	d_valnth()	959
16.118.1.2	d_valnth()	960
16.118.1.3	d_valnth()	960
16.119	no_ncwrite::var2nc Interface Reference	960
16.119.1	Detailed Description	961
16.119.2	Member Function/Subroutine Documentation	961
16.119.2.1	var2nc_1d_dp()	962
16.119.2.2	var2nc_1d_i4()	962
16.119.2.3	var2nc_1d_sp()	962
16.119.2.4	var2nc_2d_dp()	962
16.119.2.5	var2nc_2d_i4()	963
16.119.2.6	var2nc_2d_sp()	963
16.119.2.7	var2nc_3d_dp()	963
16.119.2.8	var2nc_3d_i4()	964
16.119.2.9	var2nc_3d_sp()	964
16.119.2.10	var2nc_4d_dp()	964
16.119.2.11	var2nc_4d_i4()	965
16.119.2.12	var2nc_4d_sp()	965
16.119.2.13	var2nc_5d_dp()	965
16.119.2.14	var2nc_5d_i4()	966
16.119.2.15	var2nc_5d_sp()	966
16.120	no_ncwrite::variable Type Reference	967
16.120.1	Member Data Documentation	968
16.120.1.1	latt	968

16.120.1.2count	968
16.120.1.3dimids	968
16.120.1.4dimtypes	969
16.120.1.5g0_b	969
16.120.1.6g0_d	969
16.120.1.7g0_f	969
16.120.1.8g0_i	969
16.120.1.9g1_b	969
16.120.1.10g1_d	969
16.120.1.11g1_f	969
16.120.1.12g1_i	969
16.120.1.13g2_b	970
16.120.1.14g2_d	970
16.120.1.15g2_f	970
16.120.1.16g2_i	970
16.120.1.17g3_b	970
16.120.1.18g3_d	970
16.120.1.19g3_f	970
16.120.1.20g3_i	970
16.120.1.21g4_b	970
16.120.1.22g4_d	971
16.120.1.23g4_f	971
16.120.1.24g4_i	971
16.120.1.25ame	971
16.120.1.26att	971
16.120.1.27dims	971
16.120.1.28vls	971
16.120.1.29subs	971
16.120.1.30art	971
16.120.1.31nlimited	972
16.120.1.32arid	972
16.120.1.33flag	972
16.120.1.34type	972
16.121.no_moment::variance Interface Reference	972
16.121.1Member Function/Subroutine Documentation	972
16.121.1.1variance_dp()	972
16.121.1.2variance_sp()	972
16.122.no_xor4096::xor4096 Interface Reference	973
16.122.1Member Function/Subroutine Documentation	973
16.122.1.1xor4096d_0d()	973

17.24mo_julian.f90 File Reference	995
17.25mo_kind.f90 File Reference	996
17.26mo_linfir.f90 File Reference	997
17.27mo_mcmc.f90 File Reference	997
17.28mo_message.f90 File Reference	998
17.29mo_meteo_forcings.f90 File Reference	998
17.30mo_mhm.f90 File Reference	999
17.31mo_mhm_constants.f90 File Reference	999
17.32mo_mhm_eval.f90 File Reference	1001
17.33mo_moment.f90 File Reference	1001
17.34mo_mpr_pet.f90 File Reference	1002
17.35mo_mpr_runoff.f90 File Reference	1003
17.36mo_mpr_smhorizons.f90 File Reference	1003
17.37mo_mpr_soilmoist.f90 File Reference	1003
17.38mo_mrm_constants.f90 File Reference	1004
17.38.1 Detailed Description	1004
17.39mo_mrm_eval.f90 File Reference	1005
17.40mo_mrm_file.f90 File Reference	1005
17.41mo_mrm_global_variables.f90 File Reference	1006
17.42mo_mrm_init.f90 File Reference	1009
17.43mo_mrm_mpr.f90 File Reference	1009
17.44mo_mrm_net_startup.f90 File Reference	1009
17.45mo_mrm_objective_function_runoff.f90 File Reference	1010
17.46mo_mrm_read_config.f90 File Reference	1011
17.47mo_mrm_read_data.f90 File Reference	1012
17.48mo_mrm_read_latlon.f90 File Reference	1012
17.49mo_mrm_restart.f90 File Reference	1012
17.50mo_mrm_routing.f90 File Reference	1013
17.51mo_mrm_signatures.f90 File Reference	1013
17.52mo_mrm_tools.f90 File Reference	1014
17.53mo_mrm_write.f90 File Reference	1014
17.54mo_mrm_write_fluxes_states.f90 File Reference	1015
17.55mo_multi_param_reg.f90 File Reference	1016
17.56mo_ncread.f90 File Reference	1016
17.57mo_ncwrite.f90 File Reference	1017
17.58mo_netcdf.f90 File Reference	1019
17.59mo_neutrons.f90 File Reference	1021
17.59.1 Detailed Description	1022
17.60mo_nml.f90 File Reference	1022
17.61mo_objective_function.f90 File Reference	1023

17.62mo_optimization.f90 File Reference	1023
17.63mo_orderpack.f90 File Reference	1024
17.64mo_percentile.f90 File Reference	1026
17.65mo_pet.f90 File Reference	1026
17.66mo_prepare_gridded_lai.f90 File Reference	1027
17.67mo_read_config.f90 File Reference	1027
17.68mo_read_forcing_nc.f90 File Reference	1027
17.69mo_read_latlon.f90 File Reference	1028
17.70mo_read_lut.f90 File Reference	1028
17.71mo_read_meteo.f90 File Reference	1028
17.72mo_read_optional_data.f90 File Reference	1029
17.73mo_read_spatial_data.f90 File Reference	1029
17.74mo_read_timeseries.f90 File Reference	1029
17.75mo_read_wrapper.f90 File Reference	1030
17.76mo_restart.f90 File Reference	1030
17.77mo_runoff.f90 File Reference	1030
17.78mo_sce.f90 File Reference	1031
17.78.1 Function/Subroutine Documentation	1031
17.78.1.1 set_optional()	1031
17.78.1.2 write_best_final()	1032
17.78.1.3 write_best_intermediate()	1032
17.78.1.4 write_population()	1033
17.78.1.5 write_termination_case()	1033
17.79mo_set_netcdf_outputs.f90 File Reference	1033
17.80mo_snow_accum_melt.f90 File Reference	1033
17.81mo_soil_database.f90 File Reference	1034
17.82mo_soil_moisture.f90 File Reference	1034
17.83mo_spatial_agg_disagg_forcing.f90 File Reference	1035
17.84mo_spatialsimilarity.f90 File Reference	1035
17.85mo_standard_score.f90 File Reference	1036
17.86mo_startup.f90 File Reference	1036
17.87mo_string_utils.f90 File Reference	1037
17.88mo_template.f90 File Reference	1037
17.89mo_temporal_aggregation.f90 File Reference	1038
17.90mo_temporal_disagg_forcing.f90 File Reference	1038
17.91mo_timer.f90 File Reference	1039
17.92mo_upscaling_operators.f90 File Reference	1039
17.93mo_utils.f90 File Reference	1040
17.94mo_write_ascii.f90 File Reference	1041
17.95mo_write_fluxes_states.f90 File Reference	1041

17.96mo_xor4096.f90 File Reference	1042
17.97mrm_driver.f90 File Reference	1043
17.97.1 Function/Subroutine Documentation	1043
17.97.1.1 mrm_driver()	1043
17.98RELEASES.md File Reference	1044
Bibliography	1045
Index	1047

Chapter 1

Introduction to mHM

This chapter is divided in the following sections:

- [Short Description](#)
- [The Grid-based mHM Model](#)
- [Model Formulation](#)
- [The Multiscale Parameter Regionalization Technique](#)
- [The Parameter Estimation Problem](#)
- [Model Calibration](#)
- [Test basin](#)
- [Protocols](#)

1.1 Short Description

This document describes the source code for the mesoscale Hydrologic Model mHM. mHM is based on accepted hydrological conceptualizations and is able to reproduce as accurately as possible not only observed discharge hydrographs at any point within a basin but also the distribution of soil moisture among other state variables and fluxes. To achieve these goals and to ensure a reliable performance in ungauged basins, this model employs a multiscale parameter regionalization technique to obtain effective at the scale of interest.

This model is driven by daily or hourly precipitation, temperature fields that are acquired either from satellite products or from observation networks. In the latter case, the driving meteorological data has to be prepared in advance. A module for external drift Kriging is available upon request.

1.2 The Grid-based mHM Model

The mHM hydrologic model is based on numerical approximations of dominant hydrological processes that have been tested in various models: HBV [2], [10], [5] and VIC [11]. This model includes also a number of new features that will be described in the next section. In general, this model simulates the following processes: canopy interception, snow accumulation and melting, soil moisture dynamics, infiltration and surface runoff, evapotranspiration,

subsurface storage and discharge generation, deep percolation and baseflow, and discharge attenuation and flood routing (mHM). More information about the model can be found in [14].

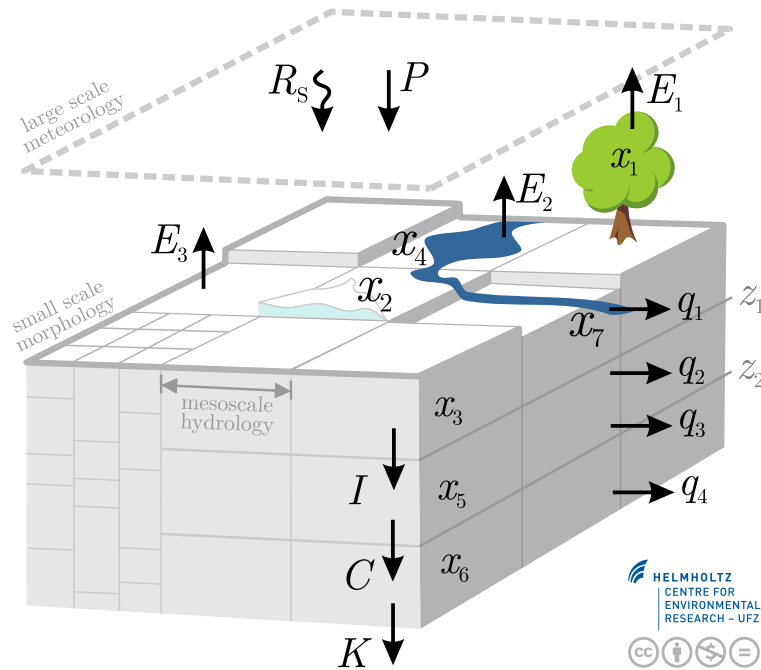


Figure 1.1: Typical mHM cell

Dominant processes of the hydrological cycle at mesoscale span over several orders of magnitude [6]. In this model, three levels (mHM levels) will be differentiated to better represent the spatial variability of state and inputs variables:

- *Level-0*: Spatial discretization suitable to describe the main features of the terrain, the main soil characteristics (pedotop), and the land cover. The cell size at this level is denoted by ℓ_0 .
- *Level-1*: Spatial discretization used to describe dominant hydrological processes [4] at the mesoscale as well as the main geological formations of the basin. The cell size at this level is denoted by ℓ_1 .
- *Level-2*: Spatial discretization suitable to describe the variability of the meteorological forcings at the mesoscale, for example the formation of convective precipitation. The cell size at this level is denoted by ℓ_2 .

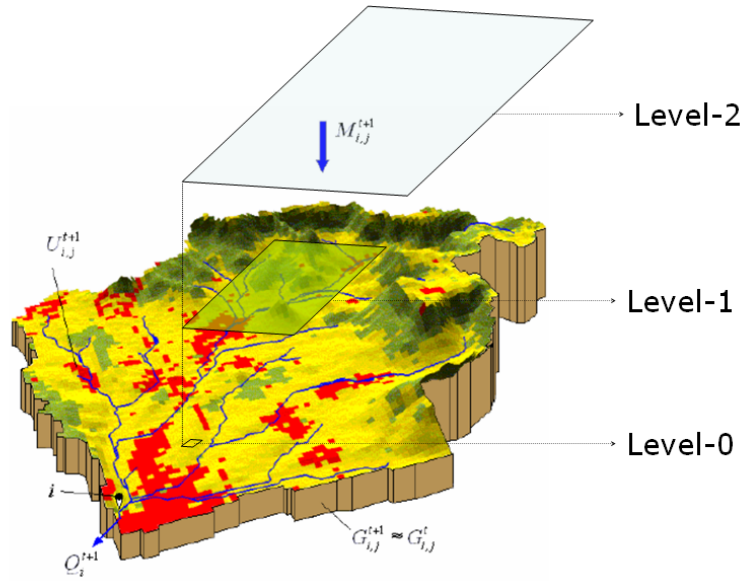


Figure 1.2: Hierarchy of data and modeling levels in mHM

1.3 Model Formulation

A mesoscale basin is an open natural system, composed of very heterogeneous materials and having fuzzy boundary conditions. The continuity assumption of the the input variables is quite difficult to justify considering that the spatial heterogeneity of a basin is mostly described by discrete attributes such soil texture types, land cover classes, and geological formations. Due to these reasons, a system of ordinary differential equations ODEs was adopted to describe the evolution of the state variables at a given location i within the domain Ω . This system of ODE is

$$\begin{aligned}
 \dot{x}_{1i} &= P_i(t) - F_i(t) - E_{1i}(t) \\
 \dot{x}_{2i} &= S_i(t) - M_i(t) \\
 \dot{x}_{3i}^l &= (1 - \rho^l) I_i^{l-1}(t) - E_{3i}^l(t) - I_i^l(t) \\
 \dot{x}_{4i} &= \rho^1 (R_i(t) + M_i(t)) - E_{2i}(t) - q_{1i}(t) \\
 \dot{x}_{5i} &= I_i^L(t) - q_{2i}(t) - q_{3i}(t) - C_i(t) \\
 \dot{x}_{6i} &= C_i(t) - q_{4i}(t) \\
 \dot{x}_{7i} &= \hat{Q}_i^0(t) - \hat{Q}_i^1(t)
 \end{aligned}$$

$$\forall i \in \Omega.$$

where

Inputs	Description
P	Daily precipitation depth, mm d ⁻¹
E_p	Daily potential evapotranspiration (PET), mm d ⁻¹
T	Daily mean air temperature, °C

Fluxes	Description
S	Snow precipitation depth, mm d ⁻¹
R	Rain precipitation depth, mm d ⁻¹
M	Melting snow depth, mm d ⁻¹

Fluxes	Description
E_p	Potential evapotranspiration, mm d ⁻¹
F	Throughfall, mm d ⁻¹
E_1	Actual evaporation intensity from the canopy, mm d ⁻¹
E_2	Actual evapotranspiration intensity, mm d ⁻¹
E_3	Actual evaporation from free-water bodies, mm d ⁻¹
I	Recharge, infiltration intensity or effective precipitation, mm d ⁻¹
C	Percolation, mm d ⁻¹
q_1	Surface runoff from impervious areas, mm d ⁻¹
q_2	Fast interflow, mm d ⁻¹
q_3	Slow interflow, mm d ⁻¹
q_4	Baseflow, mm d ⁻¹

Outputs	Description
Q_i^0	Simulated discharge entering the river stretch at cell i , m ³ s ⁻¹
Q_i^1	Simulated discharge leaving the river stretch at cell i , m ³ s ⁻¹

States	Description
x_1	Depth of the canopy storage, mm
x_2	Depth of the snowpack, mm
x_3	Depth of soil moisture content in the root zone, mm
x_4	Depth of impounded water in reservoirs, water bodies, or sealed areas, mm
x_5	Depth of the water storage in the subsurface reservoir, mm
x_6	Depth of the water storage in the groundwater reservoir, mm
x_7	Depth of the water storage in the channel reservoir, mm

Indices	Description
l	Index denoting a root zone horizon, $l = 1, \dots, L$ (say $L = 3$), in the first layer, $0 \leq z \leq z_1$
t	Time index for each Δt interval
ρ^l	Overall influx fraction accounting for the impervious cover within a cell

1.4 The Multiscale Parameter Regionalization Technique

mHM requires at most 28 parameters (depending of the configuration) per cell to account for the spatial variability of the dominant hydrological processes at a mesoscale river basin. These *effective parameters* have to be estimated through calibration. Calibrating this model with a significant number of free parameters for every grid cell would lead to over-parameterization in a mesoscale catchment. This, in turn, would tend to increase the predictive uncertainty of the model due to the *equifinality* [3] of feasible solutions. Moreover, the high dimensionality of this optimization problem is also a daunting task for the state-of-the-art optimization algorithms [12]. To overcome this problem a

multiscale parameter regionalization (MPR) was employed in the mHM model [14].

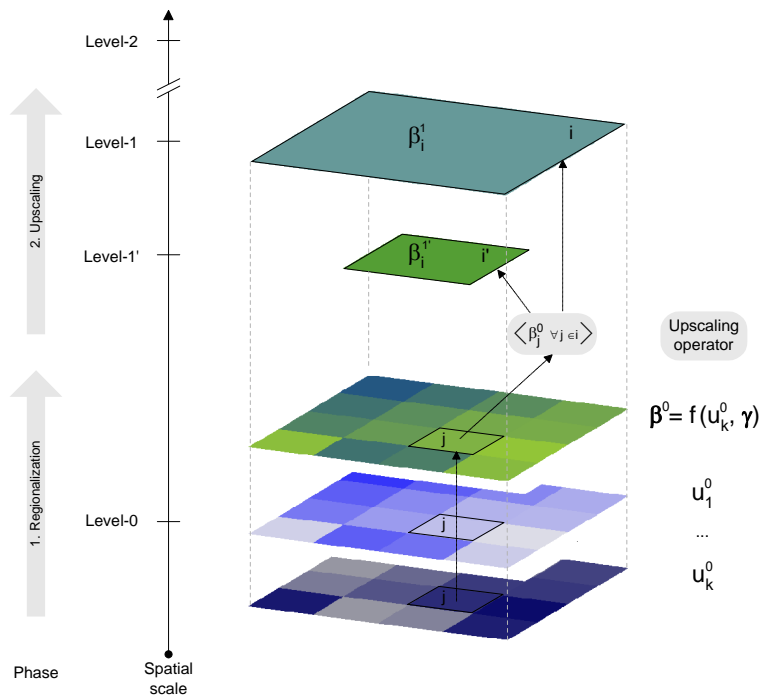


Figure 1.3: MPR

Based on this regionalization method, model parameters at a coarser grid (Level-1) are linked with their corresponding ones at a finer resolution (Level-0) (MPR). The linkage is done with upscaling operators. Model parameters at the finer scale are, in turn, regionalized with nonlinear transfer functions that couple catchment descriptors with the global parameters. Process understanding and empirical evidence are used to define these a priori relationships. The general form of an upscaling operator O is:

$$\beta_{ki}(t) = O_k \langle \beta_{kj}(t) \quad \forall j \in i \rangle_i$$

where

$$\beta_{kj}(t) = f_k(u_j(t), \gamma).$$

Here $k = 1, \dots, K$ with K denoting the number of distributed model parameters. u_j denotes a v -dimensional predictor vector for cell j at level-0 which is contained by cell i at level-1 (e.g. land cover, elevation, soil texture).

γ is a s -dimensional vector of transfer parameters (super parameters), with s denoting the number of free parameters to be calibrated or total degrees of freedom. $O_k(\bullet)_i$ denotes the kind of operator applied for the regionalization of the parameter k . Several types of operators were employed, e.g. majority \mathcal{M} , arithmetic mean \mathcal{A} , maximum difference \mathcal{D} , geometric mean \mathcal{G} , and harmonic mean \mathcal{H} . This table also shows the type of relationship employed for the transfer function and the predictors. By establishing such a relationship, the calibration algorithm finds good solutions for the transfer functions parameters ($s = 45$) instead of the model parameters for every grid cell. This, in turn, implies a great reduction of complexity since $K \times n \gg s$, where n denotes the total number of cells of a given basin at level-1.

1.5 The Parameter Estimation Problem

Let $\mathbf{M}\{\mathbf{f}, \mathbf{g}\}$ be a dynamic, spatially distributed, parameter efficient, integrated model that relates a number of state variables \mathbf{x} with some *observables* called: inputs \mathbf{u} and outputs \mathbf{y} . In general, the system is described by the following system of equations

$$\begin{aligned}\dot{\mathbf{x}}(i, t) &= \mathbf{f}(\mathbf{x}, \mathbf{u}, \boldsymbol{\beta}, \boldsymbol{\gamma})(i, t) + \boldsymbol{\eta}(i, t) \\ \mathbf{y}(i, t) &= \mathbf{g}(\mathbf{x}, \mathbf{u}, \boldsymbol{\beta}, \boldsymbol{\gamma})_{\Omega} + \boldsymbol{\varepsilon}(i, t)\end{aligned}$$

where

- \mathbf{f} is a system of functional relationships in mHM (continuous or discrete) that denote the evolution of the system over time.
- \mathbf{g} is a vector of functional relationships used to quantify the expected output (e.g. runoff) of the model denoted by $\hat{\mathbf{y}}$.
- $\boldsymbol{\varepsilon}$ denotes the uncertainty of the system originated by defects on measurements of both the inputs and outputs.
- $\boldsymbol{\eta}$ denotes the uncertainty originated by the simplification included during the formulation of \mathbf{M} or due to the lack of knowledge of the relevant processes (i.e. any kind of model structure deficiency). This term is ignored in the present case.
- $\boldsymbol{\beta}$ denotes the fields of effective mHM parameters at level-1 estimated as described in section (MPR).
- $\boldsymbol{\gamma}$ is a vector of global parameters characterized by a probability density function $\Phi_{\boldsymbol{\gamma}}$.
- i, t represent a point in space and time respectively.

In general, $\boldsymbol{\gamma}$ can be estimated, for example, by

$$\min_{\hat{\boldsymbol{\gamma}}} = \|\mathbf{y} - \hat{\mathbf{y}}\|$$

where $\|\cdot\|$ denotes a robust estimator. Many procedures to estimate global parameters are provided in mHM (e.g. simulated annealing [1], dynamically dimensioned search [16]). Other techniques can be found in the CHS Fortran Library.

1.6 Model Calibration

Good parameter sets for $\boldsymbol{\gamma}$ were identified with a split-sampling technique using an adaptive constrained optimization algorithm based on simulated annealing (SA) [1]. The overall model efficiency was estimated as a weighted combination of four estimators based on the Nash-Sutcliffe efficiency (NSE) between observed and calculated streamflows using three different time scales (daily, monthly and annual) as well as the logarithms of the streamflow to downplay the effects of the peak flows over the low flows [9]. These objective functions are denoted by ϕ_k , $k = 1, 4$. Every objective function should be normalized in the interval $[0, 1]$, with 1 representing the best possible solution. The overall objective function to be minimized is then

$$\Phi = \left(\sum_i w_i^p (1 - \phi_i)^p \right)^{\frac{1}{p}}$$

where $p > 1$, and $\sum_{i=1}^4 w_i = 1$. Here p is an exponent according to the compromise programming technique [8] and w_i denote the degree of importance of each objective. High values of p , say $p = 6$, should be chosen to avoid substitution of objective function values at low levels. In general, the estimators related to daily streamflows were twice as important as the long-term ones, thus $\{w_i\} = \{\frac{2}{4}, \frac{1}{4}, \frac{1}{4}, \frac{2}{4}\}$. The NSE for a given time interval t' is given by

$$\phi_k = 1 - \frac{\sum_{t'} (y_k(t') - \hat{y}_k(t'))^2}{\sum_{t'} (y_k(t') - \bar{y}_k(t'))^2}$$

where $\bar{y}_k(t')$ is the mean value of the observations time series over the calibration period. The index k denotes here the daily, monthly, yearly, and the transformed $\ln y(t)$ streamflow discharges. y and \hat{y} denote the observed and simulated streamflows at a given time scale. Further details about mHM's calibration options can be found in the section [Calibration Options](#).

1.7 Helpful links

Coding and Documentation Style (see [Coding and Documentation Style](#))

Setup netCDF on your MacOS system (see [Install NETCDF](#))

Data Preparation for mHM (see [Data Preparation for mHM](#))

1.8 Test basin

mHM comes with a test basin. For details see [The details about the test basin](#).

1.9 Protocols

To set up a new input data for mHM see [Protocols for setting up a new mHM basin](#).

Chapter 2

Getting Started

This chapter will introduce the structure of mHM technically.

2.1 Receive mHM

The current release of mHM is available through a code repository provided by the version control system [SVN](http://subversion.apache.org/) (<http://subversion.apache.org/>). Comprehensive instructions on accessing the repository can be found at <https://svn.ufz.de/mhm>.

2.2 Install NETCDF

The mHM input and output file format is NETCDF, so the according library is required on your system. Please go to www.unidata.ucar.edu/software/netcdf in order to download the current version.

- **on Linux:** Install as described in the online documentation.
- **on MacOS:** Use the short guide below.

2.2.1 Short Guide Install to NETCDF on Linux (example Ubuntu 16.04 with gfortran v5.4.0)

The official netcdf documentation can be found under:

http://www.unidata.ucar.edu/software/netcdf/docs/getting_and_building_netcdf.html

If not yet available you need to **install curl** on your computer. Download and unpack curl, go to the related folders and use the following commands:

```
cd /Downloads/curl-7.57
CURLDIR=/usr/local
./configure --prefix=${CURLDIR}
sudo make check
sudo make install
```

Download the five dependencies zlib, hdf5, szip, netcdf and netcdf fortran and unzip them to some folder (not the place where you will install them, e.g., /Downloads/).

Download and unpack zlib, go to the related folder and use the following commands:

```
cd /Downloads/zlib-1.2.11
ZLIBDIR=/usr/local
./configure --prefix=${ZLIBDIR}
make check
sudo make install
```

Download and unpack szip, go to the related folder and use the following commands:

```
cd /Downloads/szip-2.1.1
SDIR=/usr/local
./configure --prefix=${SDIR} sudo make check
sudo make install
```

Download and unpack hdf5, go to the related folder and use the following commands:

```
cd /Downloads/hdf5-1.8.19
H5DIR=/usr/local
./configure --with-zlib=${ZDIR} --prefix=${H5DIR}
make check
sudo make install
```

Download and unpack netCDF for C , which is needed for Fortran. Go to the related folder and use with the following commands:

```
cd /Downloads/netcdf-4.4.4
NCDIR=/usr/local
LD_LIBRARY_PATH=/usr/local/lib
CPPFLAGS=-I${H5DIR}/include LDFLAGS=-L${H5DIR}/lib ./configure --prefix=${NCDIR}
make check
sudo make install
```

Download and unpack netCDF for Fortran , go to the related folder and use the following commands:

```
cd /Downloads/netcdf-fortran-4.4.4
FC=gfortran CPPFLAGS=-I${H5DIR}/include LDFLAGS=-L${H5DIR}/lib ./configure --prefix=/usr/local/netcdf_4.4
_gfortran54
make check
sudo make install
```

2.2.2 Short Guide Install to NETCDF on MacOS

Download and unpack zlib v. 1.2.11, go to the related folder and use the following commands:

```
cd /Downloads/zlib-1.2.11/
./configure --prefix=/usr/local
make
make check
make test
sudo make install
```

Download and unpack szib v. 2.1.1, go to the related folder and use the following commands:

```
cd /Downloads/szip-2.1.1/
sudo ./configure --prefix=/usr/local
F77=/usr/local/bin/gfortran ./configure --prefix=/usr/local
make
sudo make check
sudo make install
```

Download and unpack hdf5 v. 1.8.19, go to the related folder and use the following commands:

```
cd /Downloads/hdf5-1.8.19/
FC=gfortran ./configure --prefix=/usr/local --with-zlib=/usr/local --with-szlib=/usr/local --enable-
production --enable-hl --with-pthread --with-pic
make
make check
sudo make install
```

Download and unpack netCDF for C v. 4.4.4, which is needed for Fortran. Go to the related folder and use with the following commands:

```
cd /Downloads/netcdf-4.4.4/
CPPFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib ./configure --prefix=/usr/local/
make
make check
sudo make install
```

Download and unpack netCDF for Fortran v. 4.4.4, go to the related folder and use the following commands:

For NAG compiler only:

```
FC=nagfor LDFLAGS='-L/usr/local/lib' CPPFLAGS='-I/usr/local/include' FCFLAGS="-fpp -mismatch_all -kind=byte"
" FFLAGS="-fpp -mismatch_all -kind=byte" ./configure --prefix=/usr/local/netcdf_4.4_nag53
```

For gfortran compiler only:

```
FC=gfortran LDFLAGS='-L/usr/local/lib' CPPFLAGS='-I/usr/local/include' ./configure --prefix=/usr/local/
netcdf_4.4_gfortran71
```

Compile. Note that it is possible that some tests fail during the make check according to the settings of your compiler.

```
make
make check
sudo make install
```

2.3 Run mHM under CYGWIN on Windows

A NETCDF installation under Windows7 has only been tested using CYGWIN (<https://www.cygwin.com/>). When executing the CYGWIN setup, the following packages have to be installed:

First, the following netcdf packages have to be installed ([fig_cygwin_netcdf](#)):

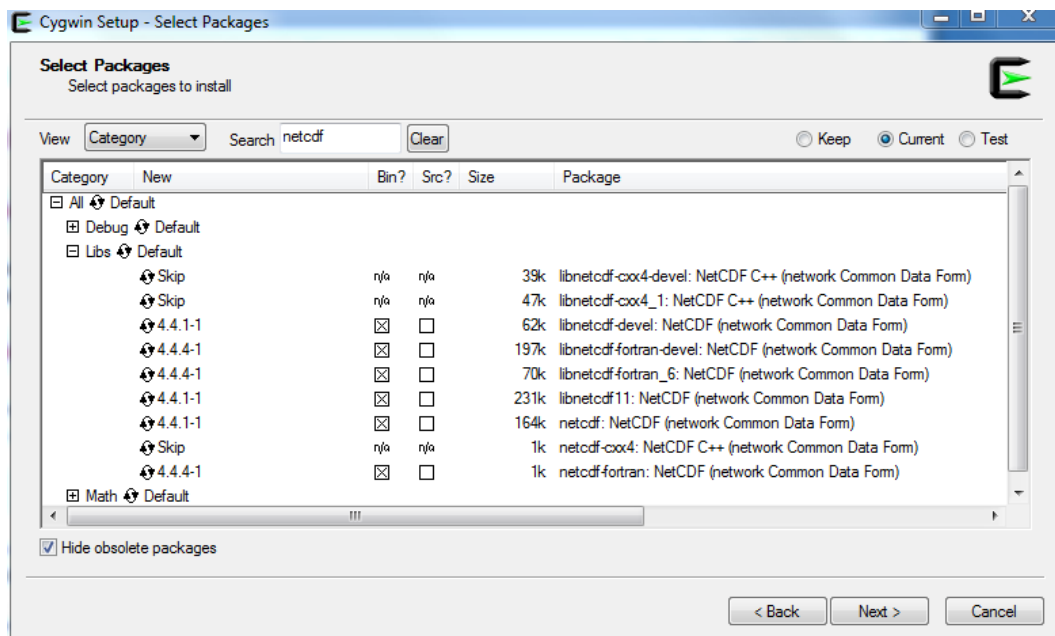


Figure 2.1: netcdf packages for CYGWIN

Second, the following hdf5 version has to be installed for netcdf-4 support. Please note the version number, which has to be 1.8.12 NOT 1.8.13 (Just click the 1.8.13 to get to 1.8.12).

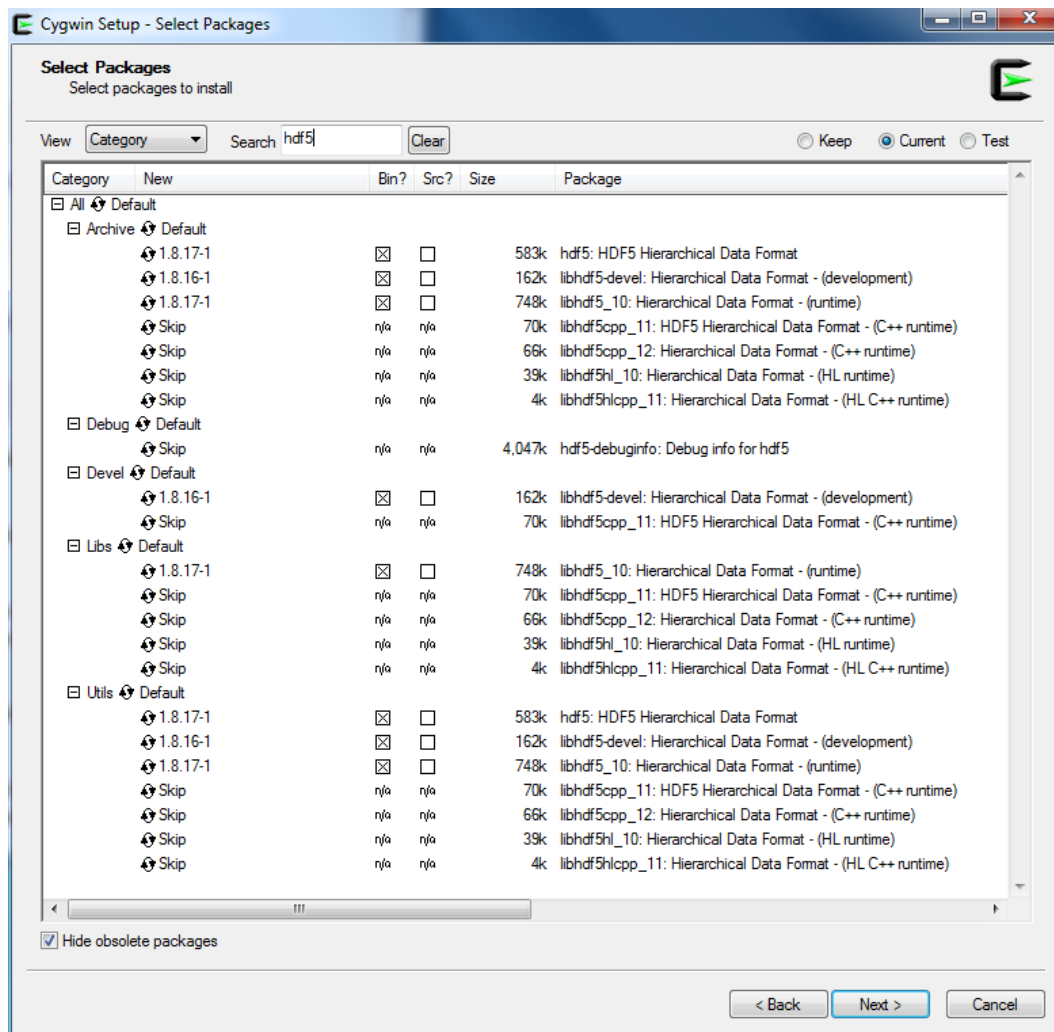


Figure 2.2: hdf5 packages for CYGWIN

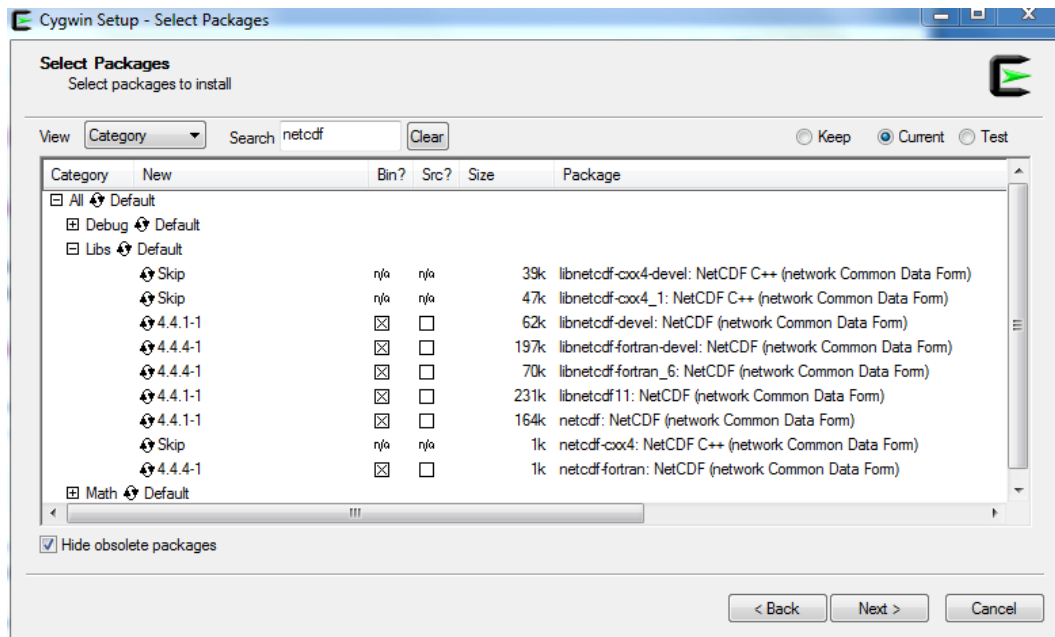


Figure 2.3: hdf5 packages for CYGWIN

Third, the gfortran compiler ([fig_cygwin_gfortran](#)) and ([fig_cygwin_liggfortran](#)) have to be installed

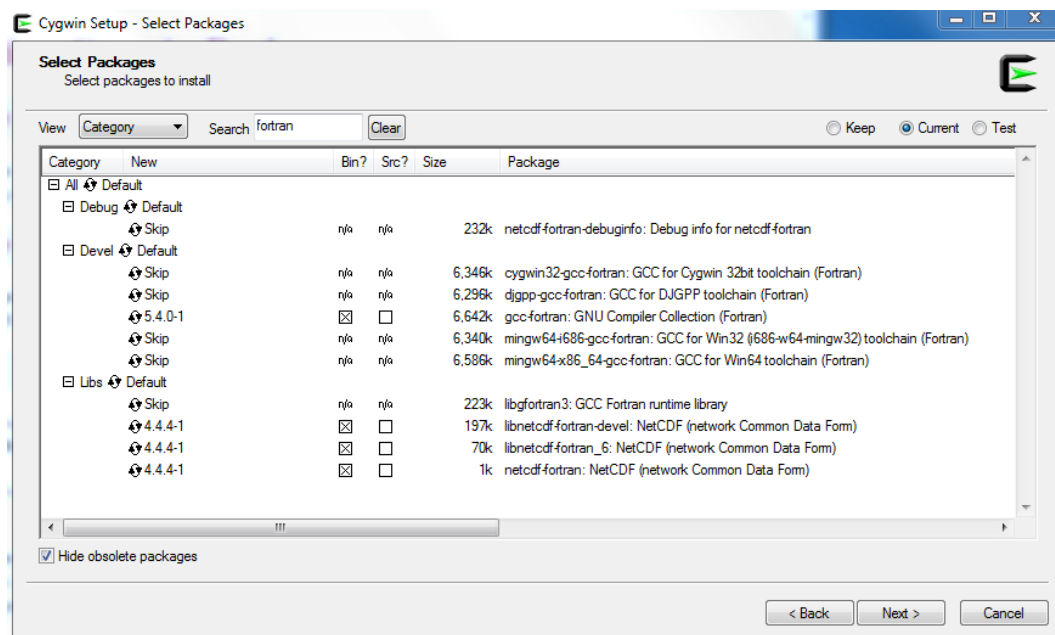


Figure 2.4: gfortran packages for CYGWIN

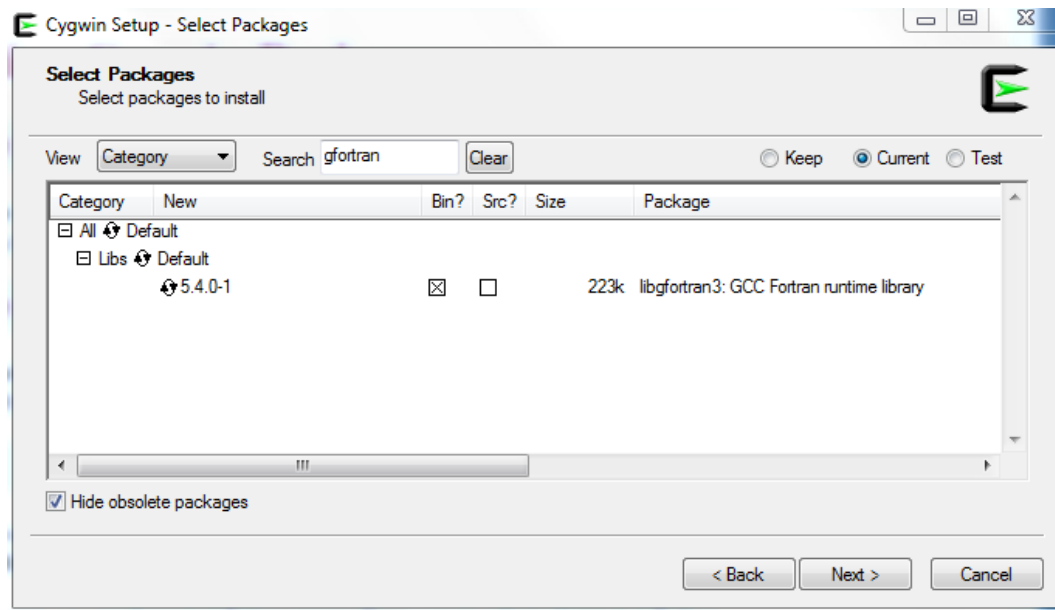


Figure 2.5: libgfortran packages for CYGWIN

Fourth, GNU make has to be made available by selecting the following:

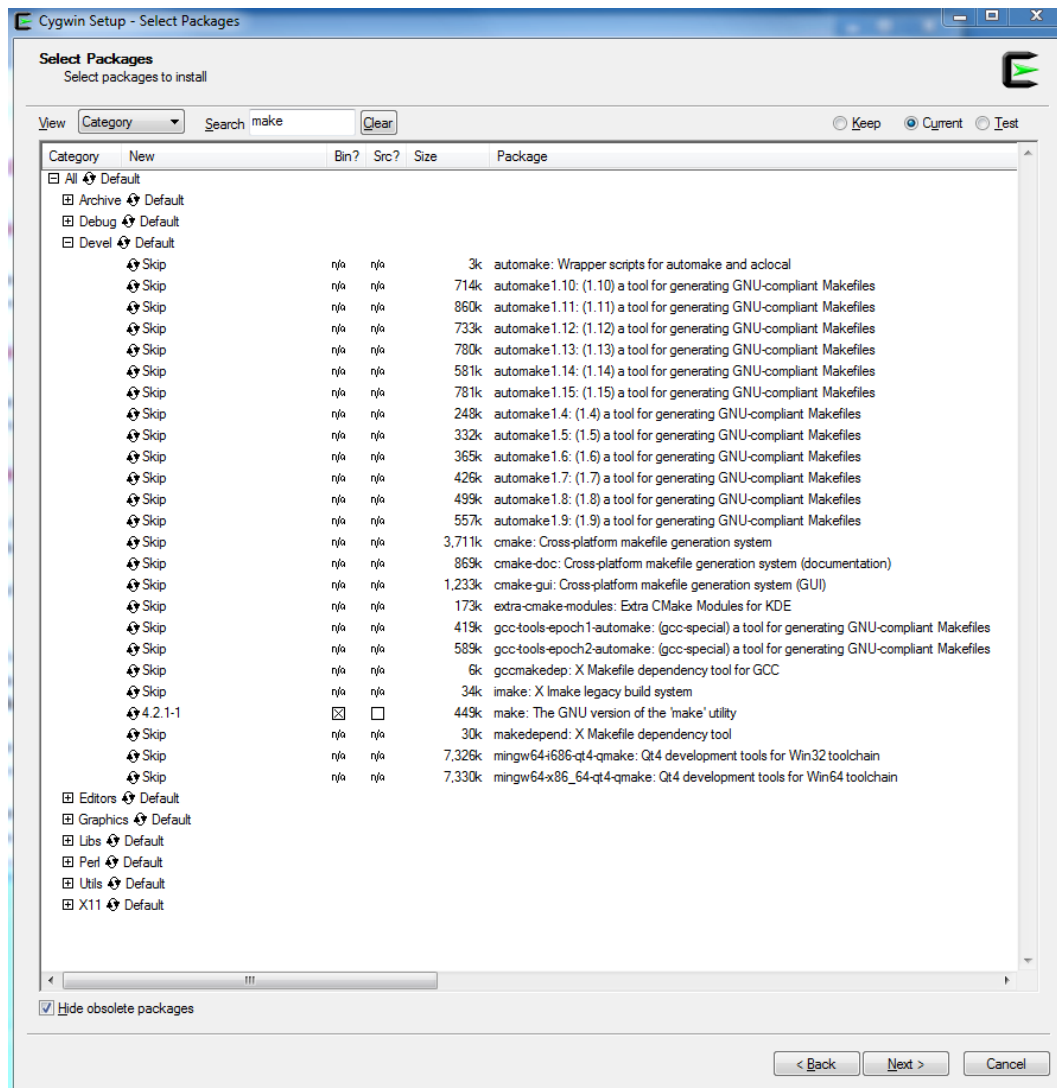


Figure 2.6: make packages for CYGWIN

Once the installation of CYGWIN including the packages outlined above is completed, you have to change the system in the Makefile to 'cygwin'. Then the mhm has proven to compile and run successfully under Windows.

Note that Cygwin users are advised to define following flag in the Makefile, otherwise compilation won't be successful.

```
EXTRA_LDFLAGS += -Wl,--stack,12485760
EXTRA_CFLAGS += -Wl,--stack,12485760
```

2.4 Compile mHM

Compilations of the code need to know paths and locations of its dependencies that are specific to the individual operating system. Before compiling, make sure that your individual requirements hold.

Open `Makefile` and adjust at least the following settings:

- `system := [SYSNAME]`
Choose an arbitrary name of your system and add your settings in `make.config/[SYSNAME].alias`. For assistance you can adapt the existing profiles in `make.config/`.

- `compiler := [COMPILER]`
Choose a compiler that was defined in `make.config/[SYSNAME].[COMPILER]`.
- `release := debug|release`
Debug mode checks all dependencies and code fractions and will be slower during compilation.
- `netcdf := netcdf4`
Choose the netcdf version your system is using.

Additional assistance for editing Makefiles are available throughout the internet. Finally, mHM can be compiled with the simple command

```
make system=cygwin compiler=gnu release=debug
```

In between two compilations, it might be useful to apply the command `make cleanclean` in order to start the next `make` from scratch (without using old temporary files).

2.5 Test mHM on Example Basin

The mHM distribution usually comes with an example test basin located in

```
test_basin/
```

The directory provides input data for the test basin and output examples. Detailed information about this test basin can be found in the chapter [The details about the test basin](#). All parameters and paths are already set by default to the test case, so you can just start the simulation with the command

```
./mhm
```

This will run mHM on two basins simultaneously and create output files for discharge and interception in `test/output_b*`. The chapter [Visualizing Model Output](#) provides further information on visualising mHM results.

2.6 Run your own Simulation

Pretty much the first step mHM takes during runtime is reading the three configuration files:

- `mhm.nml`
- `mhm_output.nml`
- `mhm_parameters.nml`

When editing these files, we recommend to use syntax highlighting for Fortran, which is featured in emacs, for example.

2.6.1 Main Configuration: Paths, Periods, Switches

The file `mhm.nml` contains the main configuration for running mHM in your catchments. Since the comments should explain each single setting, this section will only roughly describe the structure of the file. By the way, paths can be relative, absolute and even symbolic links.

- **Common Settings:** Defines output path, input look-up tables and input data format (all "nc" or all "bin") for all basins in your simulation.

- **Basin wise paths:** Set paths for input and output. Create a block for each basin. Remove needless blocks.
- **Resolution:** Hourly or Daily time step. Hydrologic resolution should be factorisable with the input resolutions (e.g. meteo: 4000, hydro: 2000, morph: 100). The routing resolutions determines the velocity of water from cell to cell (keep it greater than the hydro resolution). If you change the routing, remember to recalibrate the model (see [Calibration and Optimization](#)).
- **Restart:** mHM does provide you the option to save the whole model configuration (incl. states, fluxes, routing network, and model parameters) at the end of the simulation period. mHM is then able to restart a new simulation with this model configuration. This reduces the amount of computational time in the newly started simulation because mHM does not have to re-setup the model configuration (e.g., parameter fields and routing network).
- **Periods:** Your actual period of interest should be subsequent of a warming period, where model dynamics can set in properly. Remember to expand your input data by that period, too.
- **Soil Layers:** Soil parameters (not properties) are upscaled vertically in mHM. In this section you specify the number of horizons and depths for the hydrological processing. (This for example you do not find in any other model)
- **Land Cover:** You can provide different land cover files for different years.
- **LAI data:** Switch for choosing LAI option. The user can either select to run mHM with monthly look-up-table LAI values defined for 10 land classes or choose to run mHM using gridded LAI input data (e.g., MODIS). Gridded LAI data must be provided on a daily time-step at the Level-0 resolution. /*!
- **Process Switches:**
 - Process case 5 - potential evapotranspiration (PET):
 1. PET is input (processCase(5)=0)
 2. PET after Hargreaves-Samani (processCase(5)=1)
 3. PET after Priestley-Taylor (processCase(5)=2)
 4. PET after Penman-Monteith (processCase(5)=3)
 - Process case 8 - routing can be activated (=1) or deactivated (=0).
- **Annual Cycles:** Values for pan evaporation hold in impervious regions only. The meteorological forcing table disaggregates daily input data to hourly values.

2.6.2 Output Configuration: Time Steps, States, Fluxes

The file `mhm_output.nml` regulates how often (e.g. `timeStep_model_outputs = 24`) and which variables (fluxes and states) should be written to the final netcdf file `[OUTPUT_DIRECTORY]/mHM_Fluxes_States.nc`. We recommend to only switch on variables that are of actual interest, because the file size will greatly increase with the number of containing variables. During calibration (see [Calibration and Optimization](#)) no output file will be written.

2.6.3 Regionalised Parameters: Initial Values and Ranges

The file `mhm_parameters.nml` contains all global parameters and their initial values. They have been determined by calibration in German basins and seem to be transferable to other catchments. If you come up with a very different catchment or routing resolution, these parameters should be recalibrated (see section [Calibration and Optimization](#)).

2.7 Calibration and Optimization

By default, mHM runs without caring about observed discharge data. It will use default values of the global regionalised parameters, defined in `mhm_parameters.nml`. In order to fit discharge to observed data, mHM

has to be recalibrated. This will optimise the parameters such that mHM arrives at a best fit for discharge. The optimization procedure runs mHM many many times, sampling parameters in their given ranges for each iteration, until the objective function converges to a confident best fit.

2.7.1 The Optimization Routines

mHM comes with four available optimization methods:

- **MCMC:** The Monte Carlo Markov Chain sampling of parameter sets is recommended for estimation of parameter uncertainties. Intermediate results are written to `mcmc_tmp_parasets.nc`.
- **DDS:** The Dynamically Dimensioned Search is an optimization routine known improve the objective within a small number of iterations. However, the result of DDS is not necessarily close to your global optimum. Intermediate results are written to `dds_results.out`.
- **Simulated Annealing:** Simulated Annealing is a global optimization algorithm. SA is known to require a large number of iterations before convergence (e.g. 100 times more than DDS), but finds parameter sets closer to the global minimum like DDS. Intermediate results are written to `anneal_results.out`.
- **SCE:** The Shuffled Complex Evolution is a global optimization algorithm which is based on the shuffling of parameter complexes. It needs more iterations compared to the DDS (e.g. 20 times more), but less compared to Simulated Annealing. The increasing computational effort (i.e. iterations) leads to more reliable estimation of the global optimum compared to DDS. Intermediate results are written to `sce_results.out`.

Objective functions currently implemented in mHM are:

- **NSE:** Nash-Sutcliffe Efficiency, assuming constant + linearly relative errors. Recommended for fitting high flows.
- **lnNSE:** logarithmic Nash-Sutcliffe Efficiency, recommended for fitting low flows.
- **0.5*(NSE+lnNSE):** weights both NSE and lnNSE by 50%, roughly fits high and low flows.
- **Likelihood:** The confidence bands are probability density functions that capture variable errors, recommended for hydrological discharge.
- **KGE:** Kling Gupta model efficiency: combined measure for variability, bias and correlation
- **PD:** Pattern dissimilarity (PD) of spatially distributed soil moisture
- **ETC:** Combination of other multi-objective functions (streamflow, TWS anomaly, evapotranspiration, soil moisture), see `mhm.nml` for details

2.7.2 Calibration Settings for mHM

The following settings in `mhm.nml` are required for calibrating mHM:

- `optimize = .true.`
- `opti_method = 1`
Choose methods: 0 (MCMC), 1 (DDS), 2 (SA), 3 (SCE)
- `opti_function = 1`
Choose objective functions: 1 (NSE), 2 (lnNSE), 3 (50% NSE+lnNSE), 4 (Likelihood)
- `nIterations = 40000`
Maximum number of iterations (mHM runs), optimisers will exit earlier if convergence criteria are reached.
- `seed = -9`
The default value -9 will take a seed number from the system clock. Be warned that simulations might not be random if you define a positive seed here.

More specific settings are offered at the very end of the file `mhm.nml`.

In `mhm_parameters.nml` you find the initial values and ranges from which the optimiser will sample parameters. Most ranges are very sensitive and have been determined by detailed sensitivity analysis, so we do not recommend to change them. With the FLAG column you may choose which parameters are allowed to be optimised. The parameter `gain_loss_GWreservoir_karstic` is not meant to be optimised.

Please mind that optimization runs will take long and may demand a huge amount of hardware resources. We recommend to submit those jobs to a cluster computing system.

2.7.3 Final Calibration Results

During calibration, mHM does not write out fluxes, states and discharge, so you need to perform a final run with the calibrated parameters. When the simulations are finished, the optimal parameter set is written to

```
[OUTPUT_DIRECTORY]/FinalParams.out
[OUTPUT_DIRECTORY]/FinalParams.nml
```

You can run mHM directly with the generated namelist (by renaming it) or incorporate the results in `FinalParams.out` as new initial values into `mhm_parameters.nml`. There are two scripts that help you with that:

- `pre-proc/create_multiple_mhm_parameter_nml.sh`
Useful for many optimisation runs (many lines in `FinalParams.out`)
- `post-proc/opt2nml-params.pl`
Useful to analyze where the optimisation arrived. Using two arguments, `pathto/FinalParams.out` and `pathto/mhm_parameters.nml`, it will (1) create a new `mhm_parameters.nml.opt` filled with the new values and (2) directly show the new parameters within their ranges and warn if some have been close to their end of range by 1%.

As soon as the new parameters are set, deactivate the `optimize` switch in `mhm.nml` and **rerun** mHM once in order to obtain the final optimised output.

You should also have a look at the parameter evolution (e.g. `sce_results.out`) or final results. If any of the parameters stick to the very end of their allowed range, the result is not optimal and you will be in serious trouble. Possible reasons might be bad parameter ranges (even though they have been optimised mathematically, but in a basin or resolution not comparable to yours) or bad input data.

Chapter 3

Data Preparation for mHM

This chapter is divided in the following sections:

- [Getting Started](#)
- [Preparation of the Forcings](#)
- [Preparation of the Morphological Data](#)
- [A possible GIS workflow](#)
- [Table Data](#)
- [Land Cover Data](#)

3.1 Getting Started

To run mHM the user requires a number of datasets. The following subsection gives a short overview and references to freely available datasets.

3.1.1 Meteorological variables

Meteorological data is usually available from the weather services of the countries of modelling interest. Freely available alternatives are the EOBS (<http://www.ecad.eu/download/ensembles/download.php>) and the WATCH datasets (http://www.eu-watch.org/gfx_content/documents/README-WFD-EI.pdf).

You may have difficulties finding measurement data for Potential Evapotranspiration (PET). One possible solution, would be to calculate PET from the much easier available variables mean, maximum and minimum air temperature, using the Hargreaves-Samani method.

The two meteorological variables which are needed:

Name	Unit	Temporal resolution
Precipitation	mm	hourly to daily
Average air temperature	°C	hourly to daily

Dependent of the specification for the potential evapotranspiration (processCase(5)) additional meteorological variables may be needed:

- processCase(5) = 0 - PET is read from input.

- processCase(5) = 1 - Hargreaves-Samani equation
- processCase(5) = 2 - Priestley-Taylor equation
- processCase(5) = 3 - Penam-Monteith equation

Name	Unit	Temporal resolution
0 - Potential evapotranspiration	mm	hourly to daily
1 - Minimum air temperature	°C	daily
1 - Maximum air temperature	°C	daily
2 - Net radiation	$W m^{-2}$	daily
3 - Net radiation	$W m^{-2}$	daily
3 - Absolut vapur pressure of air	Pa	daily
3 - Windspeed	$m s^{-1}$	daily

3.1.2 Morphological variables

In addition to the Digital Elevation Models (DEM) usually provided by federal authorities, a number of free alternatives exists. SRTM data is available from 60° South to 60° North in a 3" (~ 90 m) resolution (<http://srtm.csi.cgiar.org/>), the ASTER-GDEM covers the entire globe with a resolution of 1" (<http://gdem.ersdac.jspacesystems.or.jp/>). Hydrographically corrected SRTM data and a number of derived products in different resolutions are available from the HydroSHEDS project (<http://hydrosheds.cr.usgs.gov/index.php>).

Soil and hydrogeological data is usually provided by the geological surveys. On the soil side the Harmonized World Soil Database would be a free alternative (<http://webarchive.iiasa.ac.at/Research/LUC/External-World-soil-database/HTML/>).

Name	Unit	Temporal resolution
Digital elevation model	m	-
Soil maps with textural properties (% sand and clay contents, bulk density per horizon, and root depth zone)	-	-
Geological maps with aquifer properties (specific yield, permeability, aquifer thickness)	-	-
Streamflow location	(m,m) (lat,lon)	-

3.1.3 Land Cover

Free land cover data is available from different sources. The Corine programm provides land cover scenes for Europe with resolutions of 100m and 250m (<http://www.eea.europa.eu/publications/COR0-landcover>), the Global Land Cover Map 2000 a dataset covers the entire world in a resolution of 1km.

Name	Unit	Temporal resolution
Land cover scenes	-	monthly to annual
Leaf area index	-	weekly to monthly

3.1.4 Gauging Station Information

Streamflow measurments should also be available from federal authorities. In addition, the Global Runoff Data Center provides timeseries of varying length for several thousand gauging stations all over the world (<http://www.bafg.de/GRDC>)

Name	Unit	Temporal resolution
Streamflow measurements	m^3s^{-1}	hourly to daily

3.1.5 Optional Data

Name	Unit	Temporal resolution
Snow cover	-	daily to weekly
Land surface temperature	K	daily to weekly
Groundwater station location	(m, m) (lat, lon)	-
Groundwater head measurements	m	weekly to monthly
Eddy covariance station location	(m, m) (lat, lon)	-
Eddy covariance measurements	m	hourly

3.2 Preparation of the Forcings

Forcing data should be available from federal databases like DWD for Germany. Be sure to bring data in the netcdf format, like

```
precipitation-1950-2013.nc
```

These files can be edited with the CDO module package in order to select or change data. Please read the according CDO and NCKS reference sheets for details and make sure to load the CDO modules before starting.

3.2.1 Extract Data to the Size of a Catchment

Let FULLMAP.nc be the the forcing data of a region much greater than your catchment. Let X1, X2, Y1, Y2 be the coordinates (lon and lat) of a rectangular overlapping the catchment. The forcing data MAP.nc for your catchment can be extracted by the CDO command:

```
cdo sellonlatbox,X1,X2,Y1,Y2 FULLMAP.nc MAP.nc
```

Optionally, certain pixels can be extracted, too:

```
ncks -d x,1,1 -d y,2,2 MAP.nc PIXEL.nc
```

3.2.2 Extact Data to the Time Period of Interest

Let MAP.nc be the forcing data including your period of interest. Let DATE1 and DATE2 be the starting and ending dates of your period, respectively. Their date format is YYYY-MM-DD. The data PERIOD.nc can be extracted by the CDO command:

```
cdo seldate,DATE1,DATE2 MAP.nc PERIOD.nc
```

Please note that a reference date according to DATE1 should be provided in PERIOD.nc:

```
cdo setreftime,DATE1,00:00:00,days PERIOD.nc FINAL.nc
```

3.2.3 Data Types

Any data provided for mHM should be of the data type DOUBLE. You should convert all data related to a variable VAR (e.g. tavg) with the following CDO command:

```
cdo -b F64 selname,VAR FINAL.nc FINAL64.nc
```

The only exception here is the time, its data type has to be INTEGER. This can be changed with NCAP2:

```
ncap2 -s 'time=int(time)' STILLNOTFINAL.nc FINAL.nc
```

3.2.4 Variable Names

In mHM the variable names of the forcing are hard-coded. They need to be

- "tavg" for average temperature
- "pre" for precipitation
- "pet" for evapotranspiration

For example, you can change the variable name TEMPERATURE in your NETCDF file with the following CDO command:

```
cdo chname,TEMPERATURE,tavg TEMPDATA.nc TEMPDATA-FINAL.nc
```

3.2.5 The Header File

Every meteorological data file should come with an additional file "header.txt" in the same directory. In the following example, we have only one pixel of data (ncols=nrows=1) and a cell size of 4km. The easting and northing coordinates of the lower left corner should be similar to the morphological data of your catchment.

```
ncols      1
nrows      1
xllcorner  639357.3
yllcorner  5723706.2
cellsize   4000
NODATA_value -9999
```

3.2.6 NODATA values

In order to be consistent in your data, you should specify the same NODATA value everywhere. For example, specify "-9999" in your header file as NODATA value. Usually, this should be changed also in NC files by the `ncatted` command. The following example overwrites or adds (o) the NODATA attribute "_FillValue" with value "-9999" of type double (d) to the variable "pre":

```
ncatted -O -a _FillValue,pre,o,d,-9999. INOUT.nc
```

3.3 Preparation of the LatLon Grid

As input mHM additionally needs a `latlon.nc` file specifying the geographical location of every grid cell in WGS84 coordinates. This file has to be adjusted for every resolution of the level-1 hydrological simulations.

For creating a latlon file mHM comes with the python script `create_latlon.py`, which can be found in `pre-proc/`. Detailed information for the usage of this python script can be found in the online help by typing:

```
python [MHM_DIRECTORY]/pre-proc/create_latlon.py -h
```

`create_latlon.py` needs several specifications via command line switches. First, the coordinate system of the morphological and meteorological data have to be specified according to www.spatialreference.org by using the switch: `-c`. Second, you need to specify three header files containing the corresponding information for

the different spatial resolutions connected to mHM. The three resolutions are the resolution of 1) the morphological input (switch: `-f`), 2) the hydrological simulation (switch: `-g`), and 3) the routing (switch: `-e`).

These header files can be produced by adopting the header file of the meteorological data. Therefore, copy one of these files that you generated for meteorological data (see [Preparation of the Forcings](#)):

```
cp [INPUT_DIRECTORY]/input/meteo/pre/header.txt [INPUT_DIRECTORY]/input/latlon/
```

Edit the new file `header.txt` such that `cellsize` equals your hydrologic resolution. You have to adapt `ncols` and `nrows` to that resolution such that it covers the whole region. For example, reducing the cell size from 2000 to 100, the number of columns and rows should be increased by a factor of 20.

Third, you need to set the path and filename for the resulting output file by using the switch `-o`.

An example for creating a lat-lon-file looks like

```
python [MHM_DIRECTORY]/pre-proc/create_latlon.py -c epsg:31463 -f header_100m.txt -g header_1000m.txt -e
header_2000m.txt -o ../latlon/latlon.nc
```

Keep in mind that you need one latlon file for each hydrologic resolution in mHM. Since the filename `latlon.nc` is hard-coded in mHM, we recommend to create different directories for each resolution you need, containing the related header and latlon file.

3.4 Preparation of the Morphological Data

MHM needs several morphological input datasets. All these have to be provided as raster maps in the ArcGIS ascii-format, which stores the above header and the actual data as plain text. Some of the raster files need to be complemented by look-up tables providing additional information. The tables and their structure are described in more detail in subsection [Table Data](#). Take care of the following limitations to mHM input data during data processing:

- All gridded input, i.e. your morphological and your meteorological data, needs to cover the same spatial domain. That means, that the values `xllcorner`, `yllcorner`, `xllcorner+ncols*cellsize` and `yllcorner+nrows*cellsize` have to be identical for all files!
- MHM allows you to provide your meteorological forcing in a different horizontal resolution than the morphological data. The larger cellsize however needs to be a multiple of the smaller.

The required datasets and their corresponding filenames:

Description	Raster file name	Table file name
Sink filled Digital Elevation Model (DEM)	dem.asc	-
Slope map	slope.asc	-
Aspect Map	aspect.asc	-
Flow Direction map	fdir.asc	-
Flow Accumulation map	facc.asc	-
Gauge(s) position map	idgauges.asc	[gauge-id].txt
Soil map	soil_class.asc	soil_classdefinition.txt
Hydrogeological map	geology_class.asc	geology_classdefinition.txt
Leaf Area Index (LAI) map	LAI_class.asc	LAI_classdefinition.txt
Land use map	your choice	-

3.5 A possible GIS workflow

In the following paragraphs a possible GIS workflow is outlined using the software ArcMAP 10 with the Spatial Analyst Extension and optionally the Arc Hydro Tools (http://downloads.esri.com/blogs/hydro/↔AH2/ArcHydroTools_2_0.zip)

3.5.1 General considerations

- As the spatial discretizations (i.e. resolutions, origin) of your datasets will most likely differ, it is recommended to set the following environmental settings on every processing step outlined in the following paragraphs.

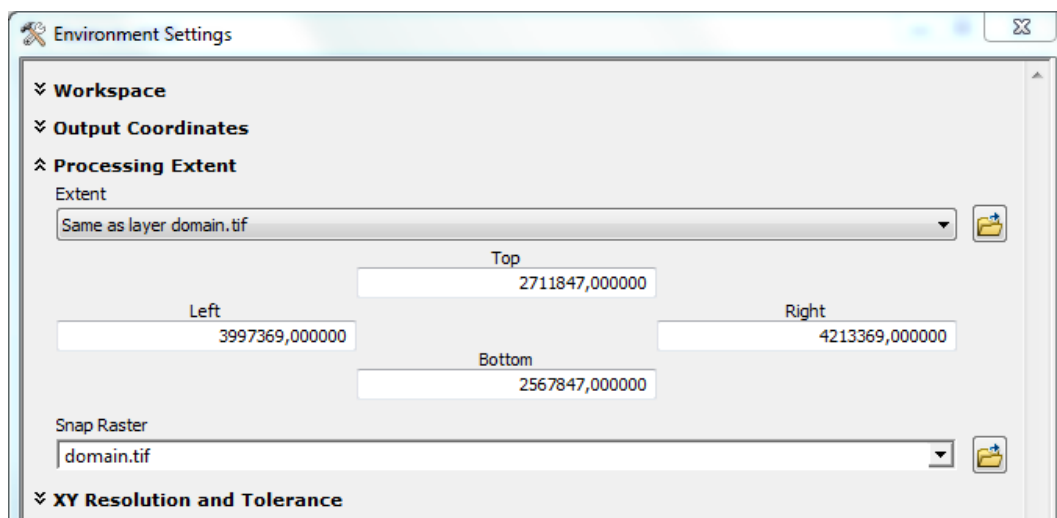


Figure 3.1: Button 'Environments...' in all Toolbox windows

Point to the largest of your input data grids in 'Extent' and 'Snap Raster', which is usually the dataset with the coarsest horizontal resolution. In the likely case, that this is your meteorological input, create a grid from any of your netcdf-files first.

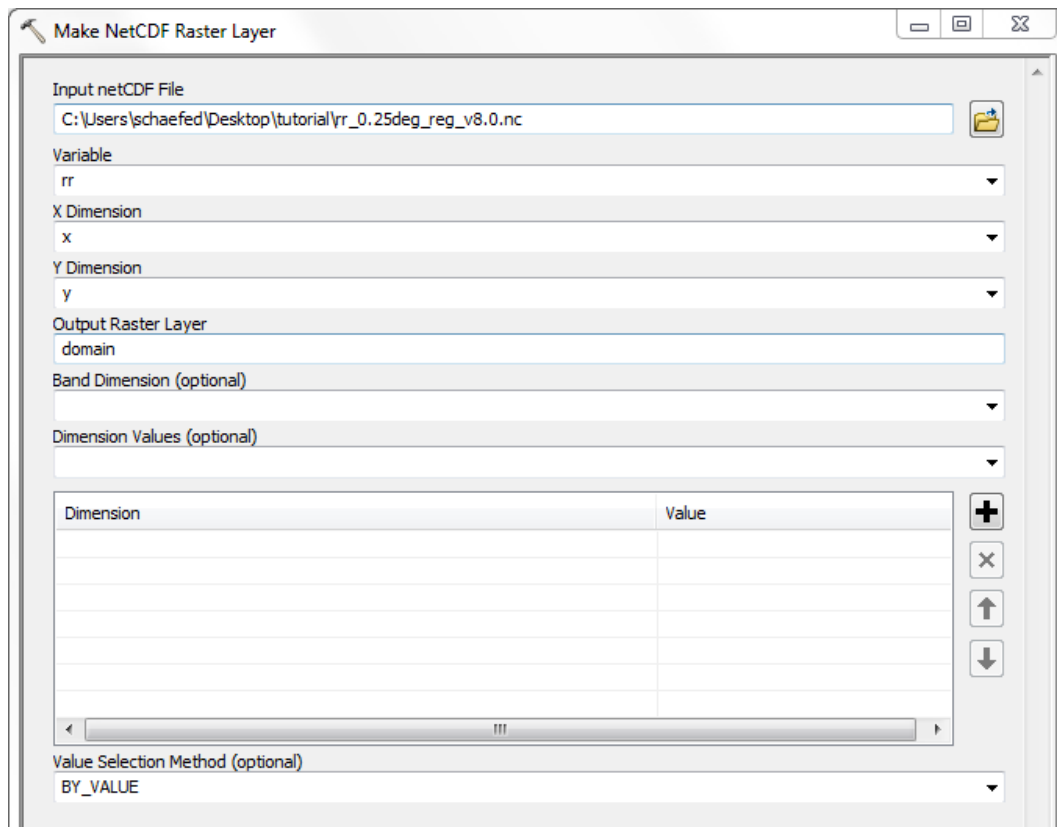


Figure 3.2: System Toolboxes -> Multidimension Tools -> Make NetCDF Raster Layer

Save the output grid (right click the raster in the 'Table of Contents' -> 'Data' -> 'Export Data...').

- If the map projections of your datasets differ, a harmonization of these becomes necessary. Repeat the depicted step to convert all your input files. Which projection to choose is highly dependent on your simulation domain and size. For the current model version an equal-area projection is strongly recommended.

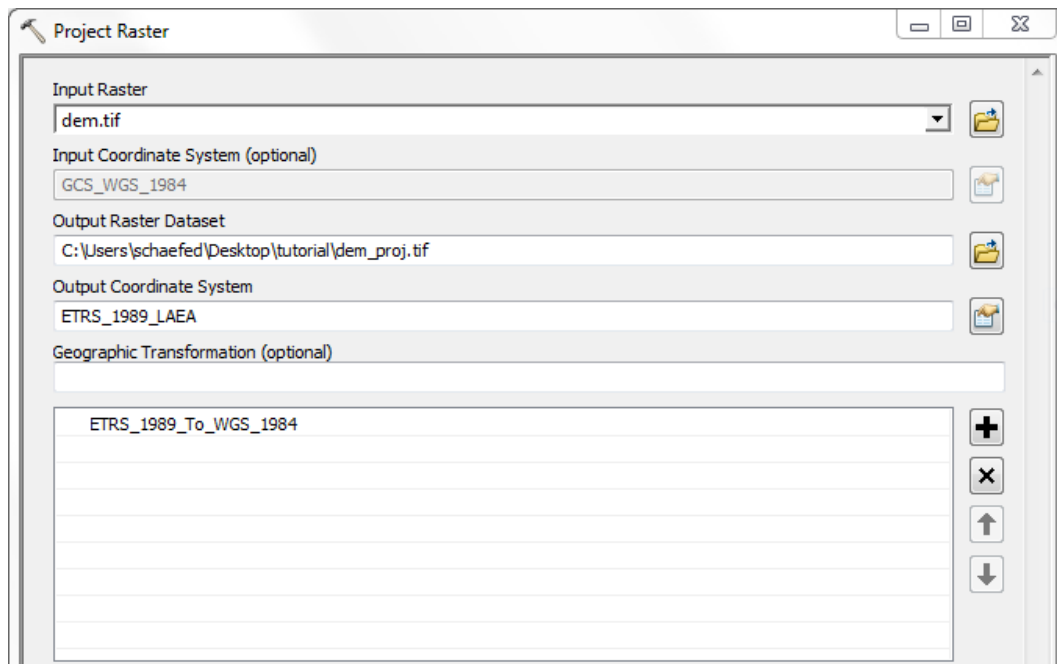


Figure 3.3: System Toolboxes -> Data Management Tools -> Projections and Transformations -> Raster -> Project Raster

- If your input datasets are not already in the desired level-0 resolution, resample the DEM, the hydrogeological, LAI, soil and land use maps. Choosing an appropriate resolution depends on data quality and needed level of simulation detail, but keep in mind that:
 1. Your different input resolution levels must be multiples of each other. E.g. you should choose a level-0 resolution of 100m (instead of 90m in case you are using SRTM data) if your meteorological input resolution is 4km.
 2. Model runtime directly depends on the number of grid cells.

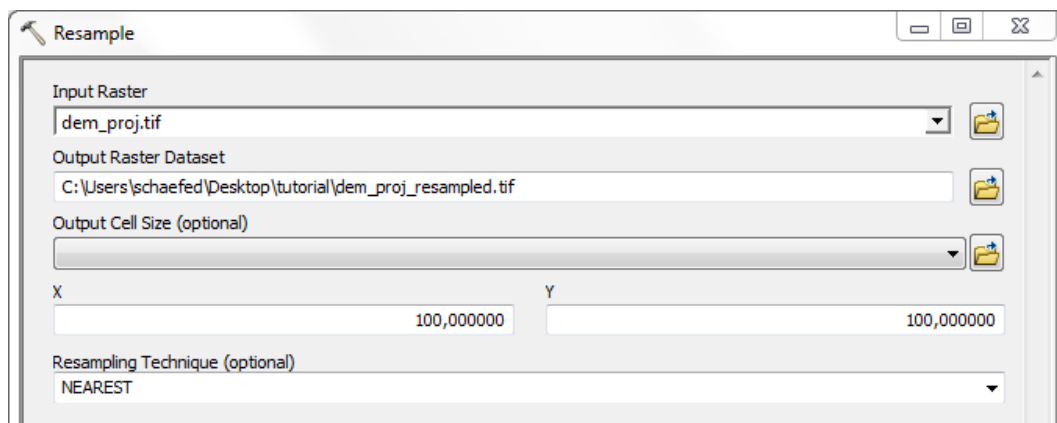


Figure 3.4: System Toolboxes -> Data Management Tools -> Raster -> Raster Processing -> Resample

- It is important that all your morphological input files exactly cover the same spatial domain. That also means that if a cell contains valid data in any one of the datasets, the very same cell must also be defined in all the others. One possibility to solve this typical problem would be to set such 'doubtful' cells to the corresponding NODATA_value. Therefore create a mask as depicted below, which only contains cells, that are defined everywhere.



Figure 3.5: System Toolboxes -> Spatial Analyst Tools -> Map Algebra -> Raster Calculator

- Mask all the mentioned datasets with the output of the 'Raster Calculator' following the procedure described in [Mask the datasets](#) of this tutorial. In case the described processing step is necessary, accomplish it **before** you reach subsection [Flow direction and flow accumulation](#) ! Masking these maps would most likely disturb the hydrological properties of your catchment data and result in unexpected model behaviour.

3.5.2 Slope map

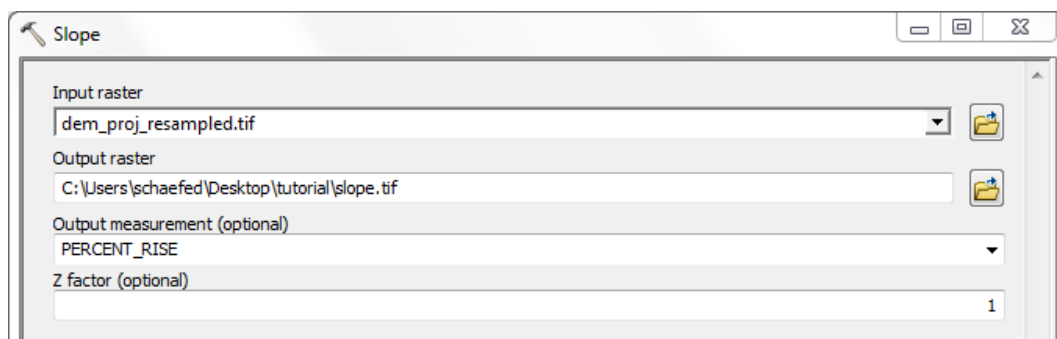


Figure 3.6: System Toolboxes -> Spatial Analyst Tools -> Surface -> Slope

3.5.3 Aspect map

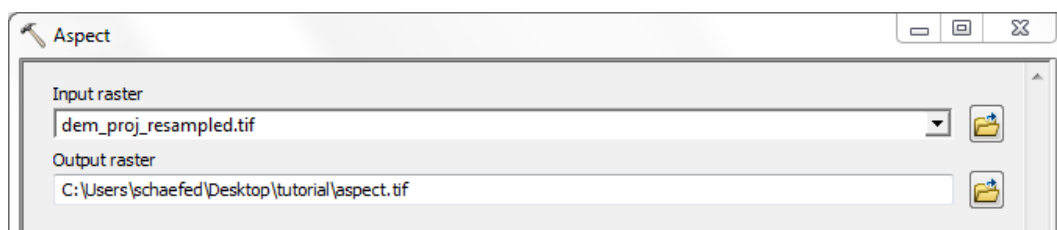


Figure 3.7: System Toolboxes -> Spatial Analyst Tools -> Surface -> Aspect

3.5.4 Fill DEM sinks

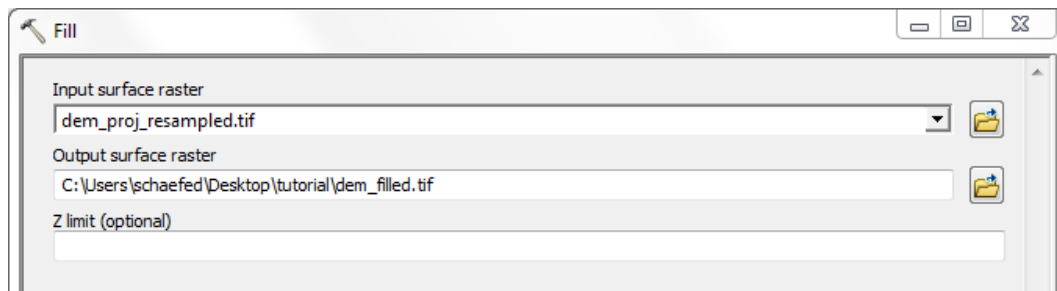


Figure 3.8: System Toolboxes -> Spatial Analyst Tools -> Hydrology -> Fill

3.5.5 Flow direction and flow accumulation

Depending on quality and resolution of the DEM map, these steps can be done with the respective tools from the Spatial Analyst Extension or by using the Arc Hydro Tools.

3.5.5.1 Spatial Analyst

If a high quality DEM, with a resolution fine enough to represent small scale river morphology is available, you may calculate flow direction and flow accumulation directly.

- Flow Direction

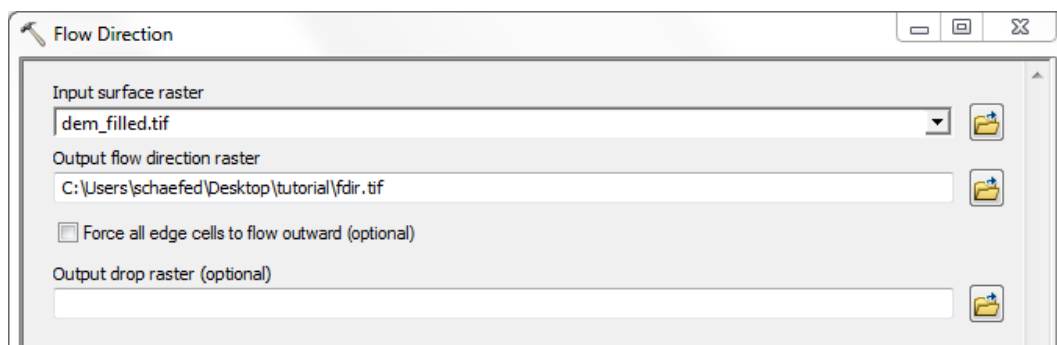


Figure 3.9: System Toolboxes -> Spatial Analyst Tools -> Hydrology -> Flow Direction

- Flow Accumulation

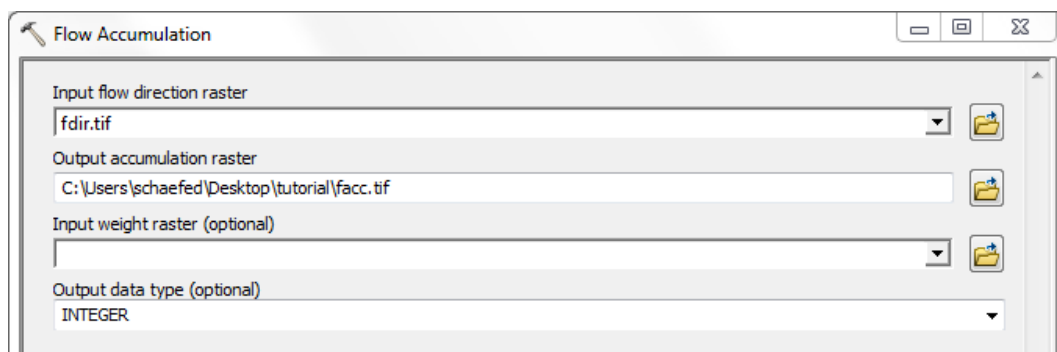


Figure 3.10: System Toolboxes -> Spatial Analyst Tools -> Hydrology -> Flow Accumulation

3.5.5.2 Arc Hydro Tools

Using the Arc Hydro Tools is recommended in case of doubtful DEM quality and/or coarse map resolutions.

- In a first step the original DEM must be reconditioned, i.e. the given altitudes will be reassigned in dependence of a stream network. The latter must be given as a Line Shapefile. In case the necessary stream network file is not available, you can get one from the USGS HydroSHEDS download portal <http://hydrosheds.cr.usgs.gov/dataavail.php>.

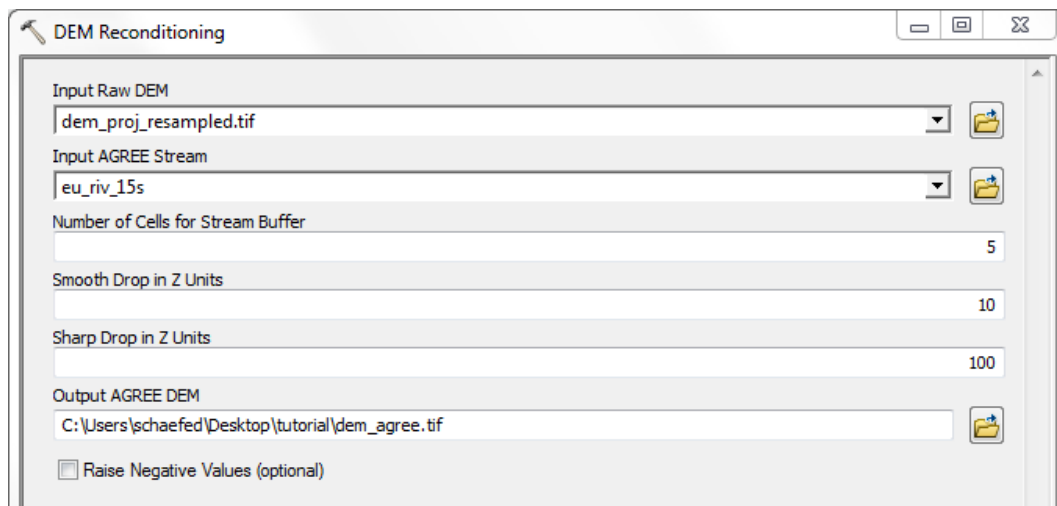


Figure 3.11: System Toolboxes -> Arc Hydro Tools -> Terrain Preprocessing -> DEM Reconditioning

- Fill the sinks in the resulting reconditioned DEM

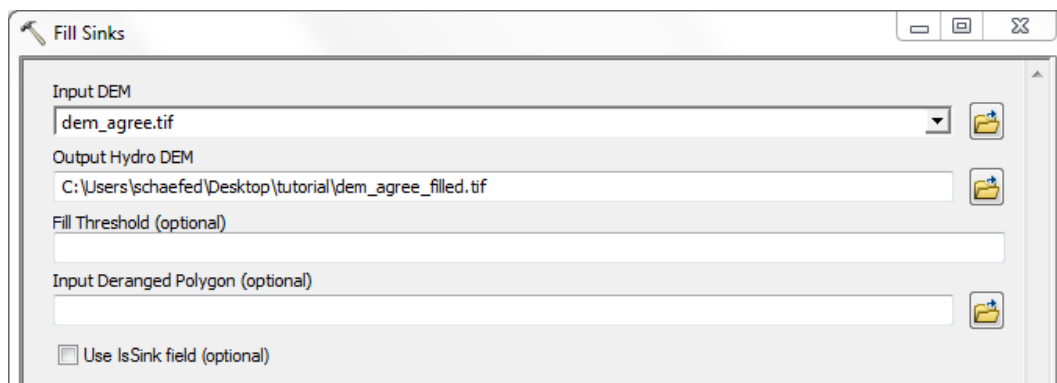


Figure 3.12: System Toolboxes -> Arc Hydro Tools -> Terrain Preprocessing -> Fill Sinks

- Calculate Flow Direction

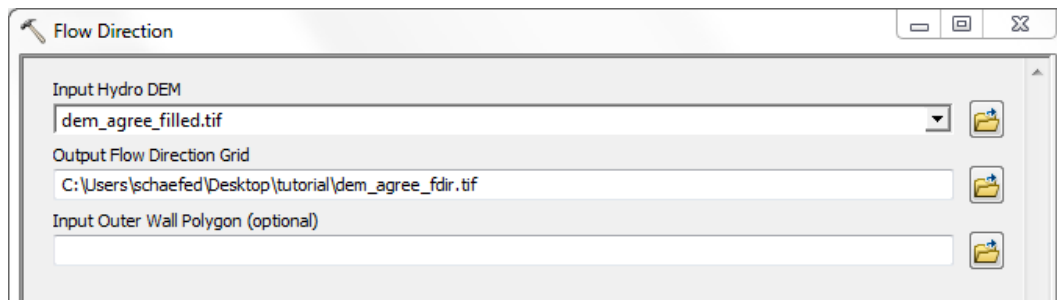


Figure 3.13: System Toolboxes -> Arc Hydro Tools -> Terrain Preprocessing -> Flow Direction

- Create a Flow Accumulation map

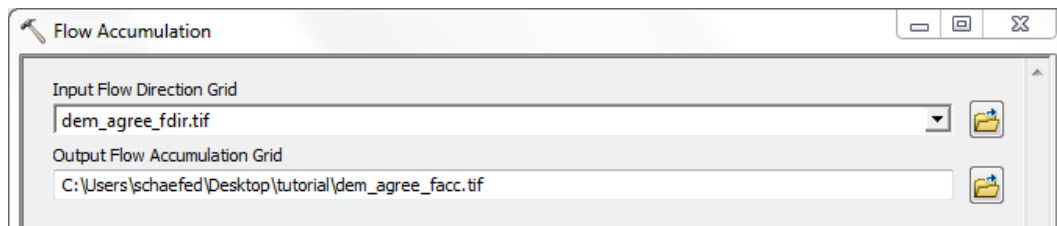


Figure 3.14: System Toolboxes -> Arc Hydro Tools -> Terrain Preprocessing -> Flow Accumulation

3.5.6 Gauges map

- Assuming that you have the positions of your gauges in a table, which has at least the columns x, y, id (in any order and/or amongst other columns), you are able to convert your data into a Point Shapefile. Choose the appropriate fields and do not forget to set the coordinate system information.

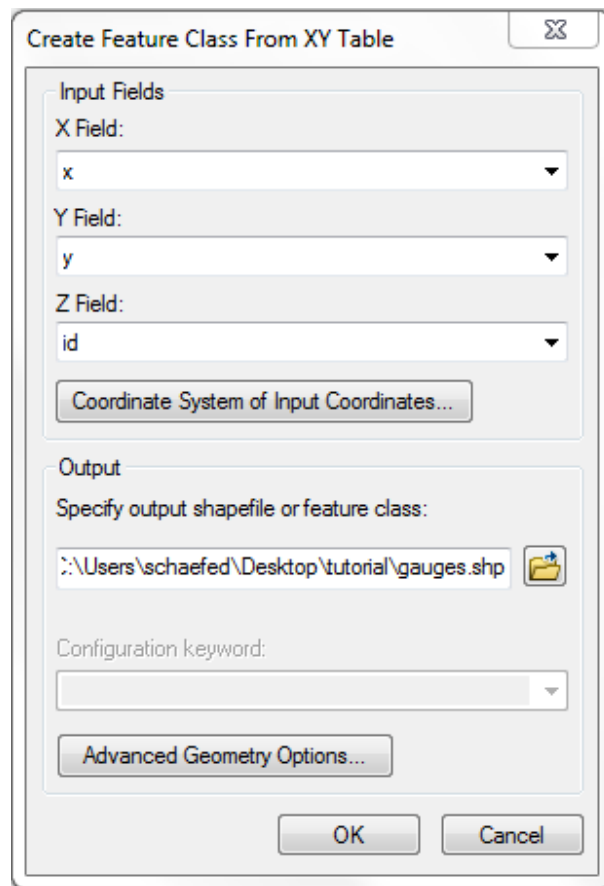


Figure 3.15: Right click on the table in Arc Catalog - > Create Feature Class -> From XY Table

- Check the positions of your gauges against the previously created Flow Accumulation map. If the points do not exactly match the stream-like features on the Flow Accumulation grid, edit the Shapefile (right click on the Point Feature in the Table of Contents -> Edit Features -> Start Editing) and move the gauges to do so. Changing the depiction of the Flow Accumulation map might facilitate this step (i.e. right click the Flow Accumulation map in the Table of Contents -> Properties -> Tab 'Symbolology' -> Choose 'Stretched' in the 'Show' box on the left hand side and Type 'Standard Deviations' in the center box. It might also be necessary to invert the color ramp by (un-)checking the 'Invert' check box). Remember to stop the editing process and save your edits (Editor Toolbar -> Editor -> Stop Editing).

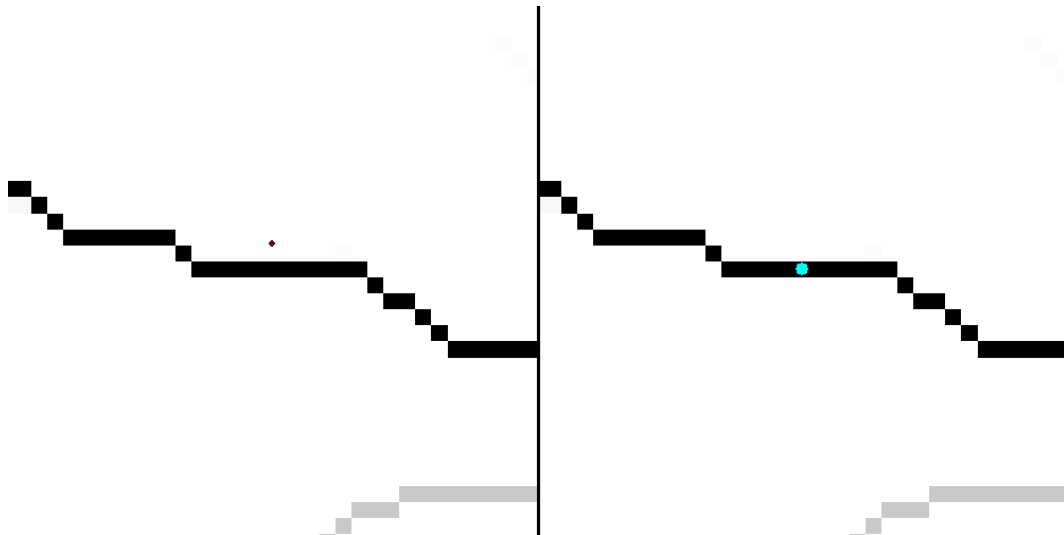


Figure 3.16: Mismatching gauges (left) should be corrected (right)

- Convert the resulting shapefile into an raster map with level-0 resolution.

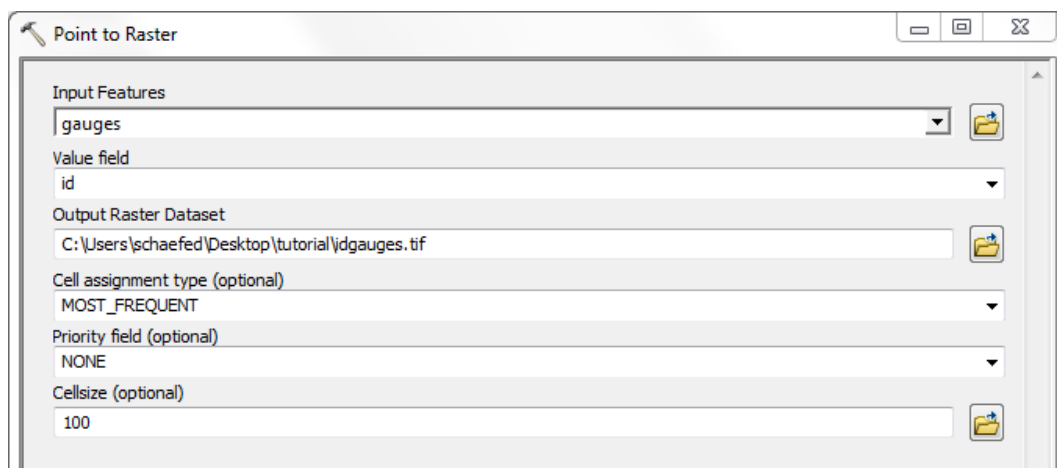


Figure 3.17: System Toolboxes -> Conversion Tools -> To Raster -> Point to Raster

3.5.7 Watershed delineation

- Edit the (possibly corrected) gauges shapefile created during the last processing step again and remove all but the outlet gauge. If you need the unedited file, create a copy of the original Shapefile before editing it.
- Delineate the basin

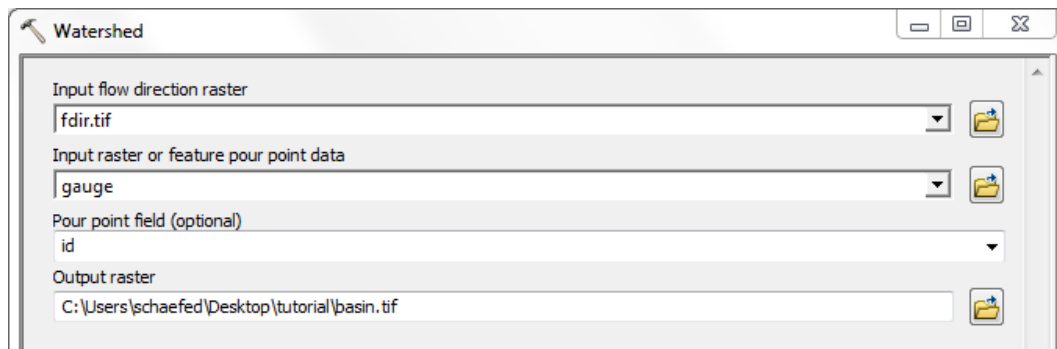


Figure 3.18: System Toolboxes -> Spatial Analyst -> Hydrology -> Watershed

3.5.8 Mask the datasets

Mask all mHM input raster files with the created watershed mask

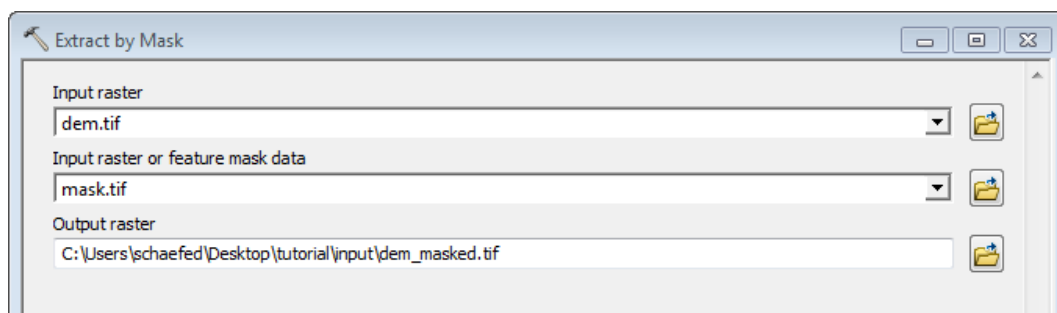


Figure 3.19: System Toolboxes -> Spatial Analysis Tools > Extraction > Extract by Mask

3.5.9 Write the ascii grids

Convert the processed and masked raster maps into ASCII files

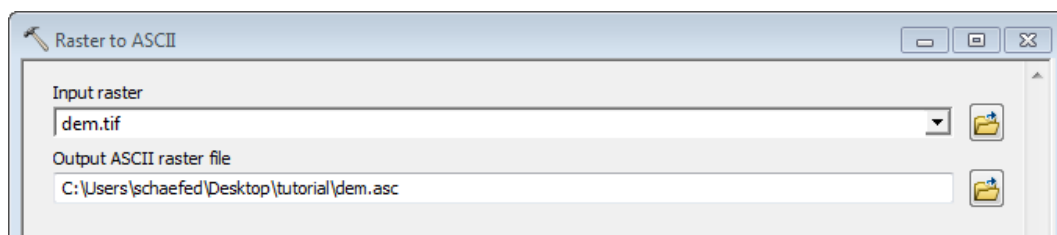


Figure 3.20: System Toolboxes -> Conversion Tools -> From Raster -> Raster to ASCII

3.6 Land Cover Data

Unlike the other classified datasets (soils, hydrogeology, LAI), where as much information as available can be added, the land use data is restricted to three different classes, which are listed below:

Class	Description
1	Forest

Class	Description
2	Impervious
3	Pervious

3.7 Table Data

As previously stated, the input datasets 'soil_class.asc', 'geology_class.asc', 'idgauges.asc' and 'LAI_class.asc' need to be complemented by look-up-tables. All these look-up tables specify the total number of classes/units to read in the very first line, following the keywords 'nSoil_Types', 'nGeo_Formations' and 'NoLAIclasses' respectively. The second line acts as a header describing the contents of the following data. In the subsection [The soil look-up table](#) hydrogeo_table and [The LAI look-up table](#) screenshots of shorted, but sufficient table files are presented. All fields are further listed and described in the respective tables.

3.7.1 The soil look-up table

```
nSoil_Types 2
MU_GLOBAL  HORIZON UD[mm] LD[mm] CLAY[%] SAND[%] BD[gcm-3]
1           1         0      300    23.0    38.0    1.2
1           2        300    1000    31.0    25.0    1.4
2           1         0      50     19.85   53.45   1.41
2           2         50     300    31.33   45.22   1.5
2           3        300    1450   35.42   39.8    1.67
```

Figure 3.21: A possible 'soil_classdefinon.txt' file

Field	Description
MU_GLOBAL	Soil id as given in the corresponding 'soil_class.asc' map. All raster map ids must be given here
HORIZON	Horizon number starting at the top of the soil column. Your are free to provide any number of horizons for each soil individually
UD[mm]	Upper depth of the horizon in millimeter. Should be 0 for the first, and the lower depth of the superimposing horizon for all others
LD[mm]	Lower depth of the horizon in millimeter
CLAY[%]	Clay content in percent
SAND[%]	Sand content in percent
BD[gcm-3]	Mineral bulk density in grams per cubic centimeter

3.7.2 The hydrogeolgy look-up table

```
nGeo_Formations 2
GeoParam(i)  ClassUnit  Karstic  Description
1            1          0      GeoUnit-1
2            2          0      GeoUnit-2
```

Figure 3.22: A possible 'geology_classdefinon.txt' file

Field	Description
GeoParam(i)	Parameter number of the formation, the link to 'mhm_parameter.nml'

Field	Description
ClassUnit	Class id as present in the 'hydrogeology_class.asc' raster map
Karstic	Presence of karstic formation (0: False, 1: True)
Description	Text description of the unit

3.7.3 The LAI look-up table

NoLAIclasses		10											
ID	LAND-USE	Jan.	Feb.	Mar.	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.
1	Coniferous-forest	11	11	11	11	11	11	11	11	11	11	11	11
2	Deciduous-forest	0.5	0.5	1.5	4.0	7.0	11	12	12	11	8.0	1.5	0.5
3	Mixed-forest	3.0	3.0	4.0	6.0	8.0	11	11.5	11.5	11	9.0	4.0	3.0
4	Sparse-forest	2.0	2.0	3.0	5.5	6.5	7.5	7.5	7.5	6.5	4.0	2.5	2.0
5	Sealed-Water-bodies	0.01	0.01	0.02	0.04	0.06	0.09	0.09	0.07	0.06	0.04	0.02	0.02
6	Viniculture	1.0	1.0	1.0	1.5	2.0	3.5	4.0	4.0	4.0	1.5	1.0	1.0
7	Intensive-orchards	2.0	2.0	2.0	2.0	3.0	3.5	4.0	4.0	4.0	2.5	2.0	2.0
8	Pasture	2.0	2.0	3.0	4.0	5.0	5.0	5.0	5.0	5.0	3.0	2.5	2.0
9	Fields	0.4	0.4	0.3	0.7	3.0	5.2	4.6	3.1	1.3	0.2	0.0	0.0
10	Wetlands	2.0	2.0	3.0	4.0	5.0	5.0	5.0	5.0	5.0	3.0	2.5	2.0

Figure 3.23: A possible 'LAI_classdefinon.txt' file

Field	Description
ID	LAI class id as given in the file LAI_class.asc
LAND-USE	Description of the LAI class
Jan. to Dec.	Monthly LAI values

3.7.4 The gauge files

```

00139:BRUGG/AARE (Abfluss)      DAILY
nodata -9999
n      1      measurements per day [1, 1440]
start 1996 01 01 00 00 (YYYY MM DD HH MM)
end    1996 01 06 00 00 (YYYY MM DD HH MM)
1996 01 01 00 00      368.500
1996 01 02 00 00      408.100
1996 01 03 00 00      493.600
1996 01 04 00 00      502.500
1996 01 05 00 00      453.100
1996 01 06 00 00      439.600

```

Figure 3.24: A possible gauge file

The structure of the gauge files is different from the look-up tables listed so far. The following table describes its content.

Line	Description
1	Gives basic information about the gauging station (Station-Id:Station-Name/River-Name)
2	Specifies the nodata value
3	The number of measurements per day
4	The start date of the time series in the format given in parentheses
5	The end date of the time series in the format given in parentheses

Line	Description
6 to end	The measurement data in cubic meter per second preceded by the actual date in the same format as the dates of lines four and five

For every gauge id given in 'idgauges.asc' one gauge file has to be created as outlined above. The file name has to **exactly** reflect the gauge id and carry a '.txt' extension. The table data file for a gauge with an id of 0343 in 'idgauges.asc' should therefore be named '0343.txt'.

3.8 Post-GIS preparation

Make sure that all decimals are indicated with dots, not commas:

```
perl -i.bak -pe 's/\,/./g' *.asc
```

Make sure that accuracy of your llcorners is reasonable. For example, the following code removes all but one digits after the dot:

```
perl -i.bak -pe 's/^(^([xy].+))\.(^(\d)\d+/$1\.$2/g' *.asc
```

Make sure that your files cover the same spatial domain. In case your grid origins match, but you are missing rows and columns, the perl script `pre-proc/enlargegrid.pl` will pad your data at the top and the right end of the grid.

```
/basin/input/morph/$ ~/mhm/pre-proc/enlargegrid.pl aspect.pl
31 rows with 47 cols.
/basin/input/morph/$ ~/mhm/pre-proc/enlargegrid.pl 50 40
aspect.asc
dem.asc
facc.asc
fdir.asc
/basin/input/morph/$ ~/mhm/pre-proc/enlargegrid.pl aspect.pl
40 rows with 50 cols.
```

If your data differs more substantially the program 'mHM/pre-proc/resize_grid.py' will harmonize your grids.

```
python match_grids.py source_grid target_grid out_grid
```

The script takes three Input arguments:

1. The source grid to be enlarged
2. The target grid which could also be an header file as described in subsection [The Header File](#)
3. An output file

The program will fail, if your target grid is smaller than the source grid or if the cellsizes of both grids are not divisable. If necessary the source grid will be shifted in order to make both origins match. This 'shift' is only accomplished by changing the values of the corner coordinates, no real interpolation will be done.

Chapter 4

Visualizing Model Output

mHM usually writes two files into the output directory specified in mhm.nml:

```
ConfigFile.log
daily_discharge.out
discharge.nc
mHM_Fluxes_States.nc
mRM_Fluxes_States.nc
```

In addition to these, three restart files are saved into the restart directory also specified in mhm.nml:

```
xxx_config.out
xxx_L11_config.nc
001_states.nc
```

The first file ConfigFile.log is ASCII type and summarizes the main configuration of mHM. The second file daily_discharge.out is ASCII type and can be read by any standard editor or ASCII interpreting scripts. It contains the simulated and observed discharge. The same timeseries are stored for all gauges in discharge.nc (the third file), but in NETCDF format. In mHM_Fluxes_States.nc all the spatial and temporal output is written, with the exception of routed discharge which can be found in mRM_Fluxes_States.nc (also a NETCDF file). These binary NETCDF files can be visualised with NCVIEW (part of the NETCDF library), VISIT (LLNL Software) or analysis scripts based on Python (PyNGL). Please note that daily_discharge.out, discharge.nc and mRM_Fluxes_States will only be written if the routing is switched on (i.e., process_case 8 equals one).

The restarting files are mainly intended for improving model performance (i.e., to avoid spin-up time) rather than for visualization purposes. The xxx prefix indicate the basin number. Since these files are also written into a binary NETCDF file, some remarks will be made below for the right interpretation of their content.

For a quick analysis we recommend NCVIEW since it quickly visualises those files via command-line and X-forwarding. Make sure to load the NC module before starting.

```
ncview FILE.nc
```

In order to hide unnecessary output messages, you may pipe them to a log file:

```
ncview FILE.nc 1> ~/log/ncview.out 2> ~/log/ncview.err
```

If you want to transfer large nc files through servers or visualise them locally, it might be useful to compress the data before. The featured nc file deflation with the ncks module will decrease the file size usually by 30-60%. Choose the deflation level from minimum (0) to maximum (9). The -4 switch in the following command will convert netcdf3 files to version 4 simultaneously.

```
ncks -4 -L 9 IN.nc OUT.nc
```

Using X-Forwarding under Windows

On Windows we recommend CYGWIN including MINTTY, OPENSSH and XINIT, XMING as the X server. An alternative is PUTTY using X-forwarding. The workflow in MINTTY is as follows:

```
/bin/XWin.exe -multiwindow  
ssh -Y SERVERNAME  
module load ncview  
ncview FILE.nc
```

In some cases it has proven to be necessary to add the following line to `/etc/profile`

```
export DISPLAY=:0
```

Exploring the contents of the Restarting files

Restarting files are simple binary dumps of arrays and vectors of all constants, parameters, state variables (1D, 2D) and fluxes at a given point in time of a simulation that are needed for executing the subsequent mHM time step in a new instance of this model without performing spin-out simulations and additional run time up to the previous time point. The stored information is divided into three categories: 1) Configuration variables at L0 and L1 levels (`xxx_config.nc`), 2) configuration variables at L11 level (i.e., routing) (`xxx_L11_config.nc`), and 3) and effective parameters, state variables and fluxes at L1 and L11 levels (`xxx_states.nc`).

WARNINGS

1) Results may appear inverted in NCVIEW with respect to the geographical coordinates used to describe the basin domain. This is due to the lack of geographical coordinates in this NetCDF visualizer and the fact that it simply dumps the content of a 2D array in C convention (row first, DIM2). In addition to that, mHM 2D arrays are stored in (lon, lat) or (X,Y) ordering, which is the transpose of the ESRI convention (lat, lon) or (Y,X).

2) Caution should be taken to interpret flow direction variables (i.e., variable `L11_fDir` in `xxx_L11_config.nc`) if visualized with NCVIEW because of its implicit 90 ° counter-clockwise rotation. Hence the values depicted NCVIEW using 8-flow direction convention (1,2,4,8,16,32,64,128) (i.e., 1 pointing to the east and then moving clockwise every 45 °) have to be rotated accordingly. In other words, direction 1 should be interpreted as 64, 2 as 128, and so on. All 2D variables, however, included the previous one, will produce consistent maps as that shown in ([Moselle Basin](#)) if plotted with appropriate routines (e.g., Python or NCL) that take into account the geographic coordinates stored within the NetCDF file.

Chapter 5

Calibration Options

5.1 Calibration of Parameters on Observations

In order to use the calibration feature of mHM, turn the `optimize` switch in `mhm.nml` to `.true..`

5.1.1 Parameters

MHM calibrates all parameters in `mhm_parameters.nml` which are flagged for optimization (column "FLAG"). Parameters can be forced to stay constant during calibration by setting the flag to 0. The upper and lower bounds define the ranges in within which the optimization methods vary the parameters. Although permitted, we recommend not to change the bounds, since they are the result of intensive sensitivity studies covering a wide range of basins.

5.1.2 Methods

Different optimization methods are available to find the best configuration of parameters for the selected objective function. You may chose between Simulated Annealing (SA), Dynamically Dimensioned Search (DDS) and Shuffled Complex Evolution algorithm (SCE). Details about these methods can be found in the module description part of this manual. At the very end of `mhm.nml` additional settings are offered for optimization.

5.1.3 Functions

The measure to define how well a quantity fits to the observations depends strongly on the specific type of application. mHM offers a variety of options for different quantities like discharge, soil moisture, or total water storage. For example, an NSE type of function will not be able to catch low values as well as $\ln(\text{NSE})$ would. In `mhm.nml`, individual objective functions are defined for specific quantities.

5.1.4 Data

Discharge and total water storage data should be provided in ASCII file format for each gauge/basin. Soil moisture and neutron data need to be provided as spatio-temporal netcdf files. Example files can be found in the folder `optional_data` in the `test_basin`. Furthermore, a `optional_data` namelist in `mhm.nml` provides several options regarding the handling of such data. For example, for soil moisture data, the temporal resolution needs to be set in `mhm.nml`. However, neutron data is restricted to be daily in the current release. The spatial resolution of soil moisture and neutrons data needs to match the hydrological resolution of mHM.

For correct preparation of input data, please take a look at the input data in the `test_basin`, which will serve as a good example. Furthermore, it could be helpful to generate optional data files from precedent mHM output, because then the temporal resolution and the spatial grid is guaranteed to match the needs of mHM. The files then can be modified with tools like `cdo` or `python`.

5.1.5 Output

Calibration runs result in a parameter file called `FinalParams.nml` which contains the optimized parameter set. Replace `mhm_parameters.nml` with `FinalParams.nml`, then run mHM in foreward mode in order to obtain hydrologic predictions.

Chapter 6

Coding and Documentation Style

The coding and documentation style guide has the following sections:

[General](#)

[In and Out of Files](#)

[Modules, Subroutines and Functions](#)

[Variables](#)

[Documentation](#)

General

- Follow the coding and documentation style of template [mo_template.f90](#) very closely.
- Enforce SI units in all code, with the only exception of mms^{-1} for $\text{kg m}^{-2}\text{s}^{-1}$.
- mHM is computed in double precision (dp).

```
real(dp) :: wind_speed ! wind speed in [m s-1]
```

- New routines should be tested with at least two different compilers (of different vendors).

In and Out of Files

- Filenames are all lower case.
- Module names and filenames are the same and start with mo_
- Filenames should be descriptive of the content and use as little abbreviations as possible.
- Fortran filenames end with .f90

```
mo_string_utils.f90
```

- Never use tabs.
- Break lines at column 130, at most. This includes comments

```
sum_of_all = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + &  
            t + u + v + w + x + y + z  
zero       = 0.0_dp ! This literal can be used whenever something has to be initialised, so that  
                  ! it will be initialised with the right number precision
```

- Indent all code blocks.

```

if (abs(a-b)>epsilon(1.0_dp)) then
  if (a >= b) then
    a = b
  else
    b = a
  endif
endif
endif

```

Modules, Subroutines and Functions

- All modules and programs have IMPLICIT NONE.
- All USE statements have the only clause.

```

module mo_calc

  use mo_kind, only: i4, dp

  implicit none

  ...

```

- Modules are PRIVATE by default.
- Module routines are made available explicitly, i.e. PUBLIC.
- Give 1-line descriptions after the public definition.

```

module mo_calc

  use mo_kind, only: i4, dp

  implicit none

  private

  public :: calc ! Calculates the thing

contains

  function calc(x)

  ...

```

- Try to write elemental pure subroutines whenever possible (see below).
- Extremely short subroutines should be avoided.
- Avoid very long subroutines (>500 lines).

Variables

- Use expressive and descriptive variable names for all variables in the namelist and for all variables that have a larger scope, i.e. that are used in more than a screen full.
This means that counter variables can still be i, ii, j, ... and local variables can also be called short names such as tmp or itmp.
But variables, which are important to read the formula, should have descriptive names such as ido_that or iDoThat.
Variable names should not be too long either. The agreed maximum numbers are

- filenames < 30 characters
- variable names < 20 characters

- All variables and literals use explicit type declarations from `mo_kind`.

```

elemental pure function circum(radius)

  use mo_kind,      only: dp
  use mo_consts,   only: pi_dp

  implicit none

  real(dp), intent(in) :: radius
  real(dp)           :: circum

  circum = 2.0_dp * pi_dp * radius

end function circum

```

- Array dimensions are in the following order: lon (east-west), lat (north-south), z (vertical), t (time level). Additional dimensions can be used and are placed in front of t.
- Always pass arrays as sub-program arguments. This means: use as little global variables as possible.
- All input/output arguments have an intent.

```

subroutine calc(x,y)

  use mo_kind, only: i4, dp

  implicit none

  real(dp),      dimension(:,:),      intent(in)  :: x
  integer(i4),   dimension(size(x,1),size(x,2)), intent(out) :: y

  ...

```

- Use intuitive names for optional arguments.

```

function calc(x, inverse)

  use mo_kind, only: i4, dp

  implicit none

  real(dp),      dimension(:,:),      intent(in) :: x
  logical,       optional,            intent(in) :: inverse
  integer(i4),   dimension(size(x,1),size(x,2)) :: calc

  logical :: iinverse

  iinverse = .false.
  if (present(inverse)) then
    iinverse = .false.
    if (inverse) iinverse = .true.
  endif

  ...

```

- Global variables are initialised at the setup stage.
- Never use hardcoded number literals such as 5., 3.14, etc. Exceptions might be -1, 0, 1, and 2. But all literals are appended by `_dp`, `_i4`, etc.

```

elemental pure function circum(radius)

  use mo_kind,      only: dp
  use mo_consts,   only: pi_dp

  implicit none

  real(dp), intent(in) :: radius
  real(dp)           :: circum

  circum = 2.0_dp * pi_dp * radius

end function circum

```

- Avoid pointers whenever possible.

- Index notation is encouraged whenever whole arrays are treated such as $a(:) = b(:)*c(:)$. It is still very good for contiguous subarrays such as $a(1:n-1) = b(1:n-1)*c(1:n-1)$. It is discouraged, though, for mixed indexing such as $a(1:n-1) = b(2:n)*c(1:n-1)$. This is (1) prone to coding error, (2) not easy to read by other programmers, and (3) hard to parallelise with OpenMP or MPI. Use `forall` or `do` instead.

Documentation

Follow the documentation structure before the interface mean in [mo_template.f90](#).

Documentation uses the automatic documentation system doxygen (<http://www.doxygen.org/>). See the manual for the complete list of documentation tags: <http://www.doxygen.nl/manual.html>

- The documentation of a routine is in front of the routine definition.
- Documentation for a module interface must be in front of the interface definition.
The individual routines do not need extra documentation.
- Only public elements will be considered by the automatic documentation system doxygen.
- Make your routines public at the beginning of the file. Append a one-line descriptions to the public statement.

Chapter 7

The details about the test basin

The test data coming with the mHM source code are for the Moselle River basin upstream of Perl, a place, where the Moselle River leaves France and enters Luxembourg and Germany ([Moselle Basin](#)). The catchment area is approximately 11,500 km², altitude ranges between 150 and 1300 m. a.m.s.l. The Moselle River originates from the Vosges Mountains and is a tributary of the Rhine River. The origin of data used in the test example is listed below.

Meteorological forcings (temperature and precipitation):

We acknowledge the E-OBS dataset from the EU-FP6 project ENSEMBLES (<http://ensembles-eu.metoffice.com>) and the data providers in the ECA&D project (<http://www.ecad.eu>).

"Haylock, M.R., N. Hofstra, A.M.G. Klein Tank, E.J. Klok, P.D. Jones, M. New. 2008: A European daily high-resolution gridded dataset of surface temperature and precipitation. J. Geophys. Res (Atmospheres), 113, D20119, doi:10.1029/2008JD10201".

(<http://www.ecad.eu/download/ensembles/ensembles.php> 02.04.2014)

Hydrological observations:

We acknowledge the Global Runoff Data Centre (GRDC), a repository for the world's river discharge data and associated metadata for providing the discharge data.

(http://www.bafg.de/GRDC/EN/01_GRDC/12_plcy/policy_guidelines.pdf;jsessionid=AA96F6F200B3880575879F15534B6669.live2052?__blob=publicationFile 02.04.2014)

SRTM (Shuttle Radar Topography Mission):

We acknowledge the U.S. Geological Survey's Earth Resources Observation and Science (EROS) Center or NASA's Land Processes Distributed Active Archive Center (LP DAAC) for providing the digital elevation model.

(<http://www2.jpl.nasa.gov/srtm/cbanddataproducts.html> 15.05.2014)

Harmonized world soil database:

We acknowledge the use of Harmonized World Soil Database dataset of the Food and Agriculture Organization of the United Nations (FAO) the International Institute for Applied Systems Analysis (IIASA), International Soil Reference and Information Centre (ISRIC), Institute of Soil Science – Chinese Academy of Sciences (ISSCAS) and Joint Research Centre of the European Commission (JRC).

(<http://webarchive.iiasa.ac.at/Research/LUC/External-World-soil-database/HTML/> 02.04.2014)

European soil database:

We acknowledge the European Commission for providing the European soil database.

(http://ec.europa.eu/geninfo/legal_notices_en.htm 02.04.2013)

Land cover:

We acknowledge the European Environment Agency for providing the CORINE land cover dataset.

(<http://www.eea.europa.eu/publications/COR0-landcover> – 15.04.2014)

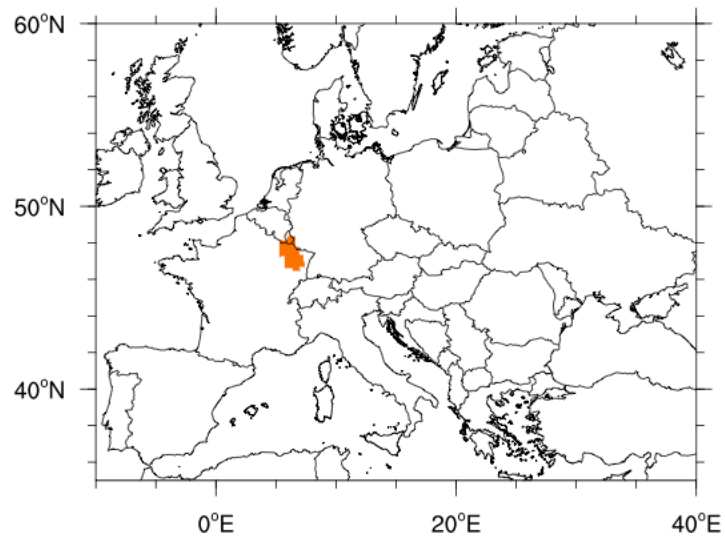


Figure 7.1: Location of the Moselle River basin upstream of Perl within the European domain

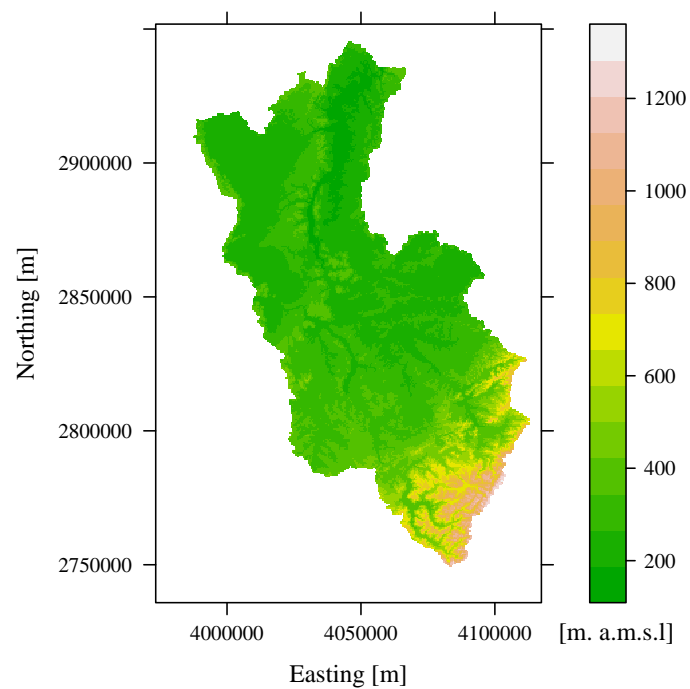


Figure 7.2: Digital elevation model for the Moselle River basin upstream of Perl

Chapter 8

Protocols for setting up a new mHM basin

The following checks and protocols describe some standard procedures for the setup of a new basin in mHM. They are based on the personal experience of the authors and do not cover all possible subjects. They are to be seen as a first guideline when working with a new basin in mHM but do not represent a complete instruction.

This chapter is divided into the following sections:

- check for input data errors
- protocol for generating a restart file
- protocol for determining the warm-up period for a new basin

8.1 Check for possible input data errors using mHM outputs

Besides mHM check for the possible input data errors (in the [mo_startup](#) routine), there could still be room for the data errors. Before you make the exhaustive model runs for calibration, make sure that you have processed your input data correctly. Below are some tips for detecting data problems using the mHM outputs.

1. Make a default mHM simulation for a relatively longer time-period, as supported by the input data sets (say 20-30 years).
2. Print out distributed fluxes/states, for example, at a monthly time scale. This feature is supported in the current mHM version using the name list file "mhm_outputs.nml". You just need to tick ".TRUE." to the fluxes/ states you want to save as a netcdf file.

Check at-least the following model outputs:

- (a) height of snow pack (L1_snowpack) [mm] – case 2 outputFlxState(2)=.TRUE.
 - This variable should be check in those basins which have snow covers/glaciers.
 - Snowpacks in the mountainous/glacier regions should be higher than those of other areas.
 - Snowpack during winter should be higher than that during summer.
- (b) mean volumetric soil moisture averaged over all soil layers [mm/mm] – case 5 (= L1_soilMoist / L1_soilMoistSat) outputFlxState(5)=.TRUE.
 - Soil moist. during winter should be higher than that of during summer.
 - In general, soil moist. in hilly areas is higher compared to that in flatter areas
- (c) actual evapotranspiration aET [mm/T] – case 9 outputFlxState(9)=.TRUE.
 - Evapotrans. during winter should be lower than that of during summer.
 - In general, evapotrans. in hilly areas is lower compared to that in flatter areas
- (d) total discharge generated per cell (L1_total_runoff) [mm/T] – case 10 outputFlxState(10)=.TRUE.

- In general, runoff in hilly areas is higher compared to that in flatter areas
3. Check the temporal dynamics of modeled fluxes/states across several locations (i.e., at several grid cells).
 - They should, in general, exhibit distinct behavior in the temporal dynamics.
 4. You could calculate a long-term mean annual dynamics of variables like
 - (a) actual evapotranspiration
 - (b) total runoff
 - (c) precipitation (use values provided in the meteo-input file)
 - This could fairly give you an idea about the respective water balance regime of the modeled basin. For example, hilly areas should exhibit a higher runoff, as a result of a relatively higher precipitation and lower ET values, compared to those of the flat areas.
 - Compute the values of runoff coefficient (long-term mean annual runoff / long-term mean annual precipitation) These values should be lesser than 1.0 everywhere.
 - Compute the values of long-term mean annual evapotranspiration / long-term mean annual pot. evapotranspiration These values should be also lesser than 1.0 everywhere.
 5. Compare the respective spatial patterns of each variable with some known literature results (google them for your respective basins)

Above check list are just few (based on the personal experience of the authors) among many possible ones, and the list will be updated as soon as we find new ways to deal with data problems.

8.2 Protocol for generating a restart file

Not specified yet.

8.3 Protocol for determining the warm-up period for a new basin

Not specified yet.

Chapter 9

mHM Dependencies

NetCDF4

Reading and writing of input and output files requires the NetCDF4 library to be present. For example, version 4.1.2 can be [downloaded from here](#). See also the [NetCDF 4.1.2 release notes](#).

Chapter 10

Publications using mHM

- Peichl, M., Thober, S., Meyer, V., and Samaniego, L.: The Effect of Soil Moisture Anomalies on Maize Yield in Germany, *Nat. Hazards Earth Syst. Sci. Discuss.*, doi:10.5194/nhess-2017-144, in review, 2017, <http://www.nat-hazards-earth-syst-sci-discuss.net/nhess-2017-144/>
- Zink, M., Kumar, R., Cuntz, M., & Samaniego, L. (2017). A high-resolution dataset of water fluxes and states for Germany accounting for parametric uncertainty. *Hydrology and Earth System Sciences*, 21(3), 1769–1790. doi:10.5194/hess-21-1769-2017, <http://www.hydrol-earth-syst-sci.net/21/1769/2017/hess-21-1769-2017.html>
- Heße, F., Zink, M., Kumar, R., Samaniego, L., & Attinger, S. (2017). Spatially distributed characterization of soil-moisture dynamics using travel-time distributions. *Hydrology and Earth System Sciences*, 21(1), 549–570. doi:10.5194/hess-21-549-2017, <http://www.hydrol-earth-syst-sci.net/21/549/2017/>
- Baroni, G., Zink, M., Kumar, R., Samaniego, L., & Attinger, S. (2017). Effects of uncertainty in soil properties on simulated hydrological states and fluxes at different spatio-temporal scales. *Hydrology and Earth System Sciences*, 21(5), 2301–2320. doi:10.5194/hess-21-2301-2017, <http://www.hydrol-earth-syst-sci.net/21/2301/2017/hess-21-2301-2017.html>
- Wollschlaeger, U., Attinger, S., Borchardt, D., Brauns, M., Cuntz, M., Dietrich, P., ... Zacharias, S. (2017). The Bode hydrological observatory: a platform for integrated, interdisciplinary hydro-ecological research within the TERENO Harz/Central German Lowland Observatory. *Environmental Earth Sciences*, 76(1), 29. doi:10.1007/s12665-016-6327-5, <https://link.springer.com/article/10.1007/s12665-016-6327-5>
- Mueller, C., Zink, M., Samaniego, L., Krieg, R., Merz, R., Rode, M., & Knoeller, K. (2016). Discharge Driven Nitrogen Dynamics in a Mesoscale River Basin As Constrained by Stable Isotope Patterns. *Environmental Science & Technology*, 50(17), 9187–9196. doi:10.1021/acs.est.6b01057, <http://pubs.acs.org/doi/abs/10.1021/acs.est.6b01057>
- Zink, M., Samaniego, L., Kumar, R., Thober, S., Mai, J., Schaefer, D., & Marx, A. (2016). The German drought monitor. *Environmental Research Letters*, 11(7), 74002. doi:10.1088/1748-9326/11/7/074002, <http://iopscience.iop.org/article/10.1088/1748-9326/11/7/074002/meta>
- Marx, A., Samaniego, L., Kumar, R., Thober, S., Mai, J., & Zink, M. (2016). Der Duerremonitor - Aktuelle Information zur Bodenfeuchte in Deutschland. In *Wasserressourcen - Wissen in Flussgebieten vernetzen* (pp. 131–142). Forum fuer Hydrologie und Wasserbewirtschaftung, <http://www.ufz.de/index.php?en=20939&ufzPublicationIdentifier=17277>
- Rakovec, O., Kumar, R., Mai, J., Cuntz, M., Thober, S., Zink, M., Attinger, S., Schaefer, D., Schroen, M., Samaniego, L. (2016). Multiscale and Multivariate Evaluation of Water Fluxes and States over European River Basins. *Journal of Hydrometeorology*, 17(1), 287–307. doi:10.1175/JHM-D-15-0054.1, <http://journals.ametsoc.org/doi/abs/10.1175/JHM-D-15-0054.1>

- Rakovec, O., Kumar, R., Attinger, S. and Samaniego, L. (2016). Improving the realism of hydrologic model functioning through multivariate parameter estimation. *Water Resources Research*, 52. doi:10.1002/2016WR019430, <http://onlinelibrary.wiley.com/doi/10.1002/2016WR019430/full>
- Nijzink, R. C., Samaniego, L., Mai, J., Kumar, R., Thober, S., Zink, M., Schaefer, D., Savenije, H. H. G., and Hrachowitz, M.: The importance of topography-controlled sub-grid process heterogeneity and semi-quantitative prior constraints in distributed hydrological models, *Hydrol. Earth Syst. Sci.*, 20, 1151-1176, doi:10.5194/hess-20-1151-2016, 2016., <http://www.hydrol-earth-syst-sci.net/20/1151/2016/>
- Thober, S., Kumar, R., Sheffield, J., Mai, J., Schaefer, D., & Samaniego, L. (2015). Seasonal Soil Moisture Drought Prediction over Europe Using the North American Multi-Model Ensemble (NMME). *Journal of Hydrometeorology*, 16(6), 2329–2344. doi:10.1175/JHM-D-15-0053.1, <http://journals.ametsoc.org/doi/abs/10.1175/JHM-D-15-0053.1>
- Cuntz, M., Mai, J., Zink, M., Thober, S., Kumar, R., Schaefer, D., ... Samaniego, L. (2015). Computationally inexpensive identification of noninformative model parameters by sequential screening. *Water Resources Research*, 51(8), 6417–6441. doi:10.1002/2015WR016907, <http://onlinelibrary.wiley.com/doi/10.1002/2015WR016907/full>
- Livneh, B., Kumar, R., & Samaniego, L. (2015). Influence of soil textural properties on hydrologic fluxes in the Mississippi river basin. *Hydrological Processes*, 29(21), 4638-4655, <http://onlinelibrary.wiley.com/doi/10.1002/hyp.10601/full>
- Schoeniger, A., Woehling, T., Samaniego, L., & Nowak, W. (2014). Model selection on solid ground: Rigorous comparison of nine ways to evaluate Bayesian model evidence. *Water resources research*, 50(12), 9484-9513, <http://onlinelibrary.wiley.com/doi/10.1002/2014WR016062/full>
- Woehling, T., Samaniego, L., & Kumar, R. (2013). Evaluating multiple performance criteria to calibrate the distributed hydrological model of the upper Neckar catchment. *Environmental Earth Sciences*, 69(2), 453–468. doi:10.1007/s12665-013-2306-2, <https://link.springer.com/article/10.1007/s12665-013-2306-2>
- Samaniego, L., Kumar, R., & Zink, M. (2013). Implications of Parameter Uncertainty on Soil Moisture Drought Analysis in Germany. *Journal of Hydrometeorology*, 14(1), 47–68. doi:10.1175/JHM-D-12-075.1, <http://journals.ametsoc.org/doi/abs/10.1175/JHM-D-12-075.1>
- Kumar, R., Livneh, B., & Samaniego, L. (2013). Toward computationally efficient large-scale hydrologic predictions with a multiscale regionalization scheme. *Water Resources Research*, 49(9), 5700–5714. doi:10.1002/wrcr.20431, <http://onlinelibrary.wiley.com/doi/10.1002/wrcr.20431/full>
- Kumar, R., Samaniego, L., & Attinger, S. (2013). Implications of distributed hydrologic model parameterization on water fluxes at multiple scales and locations. *Water Resources Research*, 49(1), 360–379. doi:10.1029/2012WR012195, <http://onlinelibrary.wiley.com/doi/10.1029/2012WR012195/abstract>
- Zacharias, S., Bogen, H., Samaniego, L., Mauder, M., Fuß, R., Puetz, T., ... & Bens, O. (2011). A network of terrestrial environmental observatories in Germany. *Vadose Zone Journal*, 10(3), 955-973, <https://dl.sciencesocieties.org/publications/vzj/abstracts/10/3/955>
- Kalbacher, T., Delfs, J. O., Shao, H., Wang, W., Walther, M., Samaniego, L., ... & Sun, F. (2012). The IWAS-ToolBox: software coupling for an integrated water resources management. *Environmental Earth Sciences*, 65(5), 1367-1380, <https://link.springer.com/article/10.1007/s12665-011-1270-y>
- Samaniego, L., Kumar, R., & Jackisch, C. (2011). Predictions in a data-sparse region using a regionalized grid-based hydrologic model driven by remotely sensed data. *Hydrology Research*, 42(5), 338–355. doi:10.2166/nh.2011.156, <http://hr.iwaponline.com/content/42/5/338.article-info>
- Kumar, R., Samaniego, L., & Attinger, S. (2010). The effects of spatial discretization and model parameterization on the prediction of extreme runoff characteristics. *Journal of Hydrology*, 392(1-2), 54–69. doi:10.1016/j.jhydrol.2010.07.047, <http://www.sciencedirect.com/science/article/pii/S0022169410004865>

- Samaniego, L., Kumar, R., & Attinger, S. (2010). Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale. *Water Resources Research*, 46(5), W05523. doi:10.1029/2008WR007327, <http://onlinelibrary.wiley.com/doi/10.1029/2008WR007327/full>

Chapter 11

mHM RELEASE NOTES

mHM v5.8 (Dec 2017)

New Features:

- Implementation of a new process for PET correction based on LAI at PET `process(5)=-1` (Cuneyd Demirel + *GEUS* colleagues);
- Pre-processor code for SOILGRIDS data as used for the EDgE project (Rohini Kumar);
- Reduced computational time of the neutron forward model COSMIC by factors of 30–100 (Maren Kaluza);
- Compression of the netCDF output files (David Schaefer);
- Optional project description added into the `mhm.nml`

Bugs resolved from release 5.7:

- `processCase(3)=3` did not work when compiled with openMP.
- openMP declarations missing in `mo_mpr_smhorizons.f90` for the case of `iFlag_soil=1`

Known bugs:

None.

Restrictions:

- For `gfortran` compilers mHM supports only v4.8 and higher.
- If you wish to use a special process description of evapotranspiration (`processCase(4)`) please contact [Matthias Zink](#).
- If you wish to use features connected to ground albedo neutrons (`processCase(9)`), please contact [Martin Schrön](#).
- If you wish to use the multi-scale Routing Model as stand-alone version, please contact [Stephan Thober](#).

mHM v5.7 (Jun 2017)

New Features:

- New process descriptions for the soil evapotranspiration module:

- Field capacity dependency to root fraction coefficient (`processCase(3)=2`) – implemented by G↔EUS.
- Jarvis (1989, J. Hydrol.) evapotranspiration reduction (`processCase(3)=3`) – implemented by G↔EUS.
- Use local, monthly LAI climatology instead of look-up table, i.e., `LAI_classdefinition.txt` (`timeStep_LAI_input=1`).
- New objective functions for model calibration
 - Calibration of mHM using catchment average evapotranspiration (`opti_function=27`).
 - Calibration of mHM using soil moisture and streamflow simultaneously (`opti_function=28`).

Bugs resolved:

- Calibration using `processCase(8)=2` (routing with adaptive timestep) does properly work now.
- Streamflow output is now properly written to the NetCDF.

Known bugs:

None

Restrictions:

- For `gfortran` compilers mHM supports only v4.8 and higher.
- If you wish to use a special process description of evapotranspiration (`processCase(4)`) please contact [Matthias Zink](#).
- If you wish to use features connected to ground albedo neutrons (`processCase(9)`), please contact [Martin Schrön](#).
- If you wish to use the multi-scale Routing Model as stand-alone version, please contact [Stephan Thober](#).

mHM v5.6 (Dec 2016)

New Features:

- **Routing extended:** Implementation of a new parametrization for the routing model (`processCase(8)=2`). This routing option is based on an adaptive time step to improve the scalability and transferability of the model as well as a significant reduction in run time. The adaptive time step is calculated as ratio of routing resolution and celerity, the latter can be given as parameter in `mhm_parameter.nml`.

Bugs resolved:

- Any model time step from 1 h to 24 h can be chosen (in releases v5.4 and v5.5 only 1 h worked properly).
- Estimation of the Hargreaves-Samani PET for high altitudes works properly now (there have been numerical issues for high latitude values).
- Reading catchment outlets from the `restart` file works now (bug appeared in v5.5).

Known bugs:

- Calibration using `processCase(8)=2` (adaptive timestep) does not work, please use `processCase(8)=1`.

Restrictions:

- For `gfortran` compilers mHM supports only v4.8 and higher.
- If you wish to use a special process description of evapotranspiration (`processCase(4)`) please contact [Matthias Zink](#).
- If you wish to use features connected to ground albedo neutrons (`processCase(9)`), please contact [Martin Schrön](#).
- If you wish to use the multi-scale Routing Model as stand-alone version, please contact [Stephan Thober](#).

mHM v5.5 (Jun 2016)**New Features:**

- Routing works on domains with multiple outlets (e.g., continental level).
- New option for providing soil data. They can be provided as predefined layers (one map per layer).
- Speed up of mHM for big domains, due to reformulations in the model start up.
- Pre-processing: new tools for i) cutting out a catchment from a existing dataset, ii) estimation of Hargreaves-Samani evapotranspiration, and iii) enlarging the grids of the input files.

Bugs resolved:

- Assigning routing parameters is done properly now.

Known bugs:

- Specifying a model time step of 24h (in `mhm.nml`) does not work, please stick with the default time step (1h)

Restrictions:

- For `gfortran` compilers mHM supports only v4.8 and higher.
- If you wish to use a special process description of evapotranspiration (`processCase(4)`) please contact [Matthias Zink](#).
- If you wish to use features connected to ground albedo neutrons (`processCase(9)`), please contact [Martin Schrön](#).
- If you wish to use the multi-scale Routing Model as stand-alone version, please contact [Stephan Thober](#).

mHM v5.4 (Dec 2015)**New Features:**

- The routing of mHM can be used as a stand alone version/independent software called *multiscale Routing Model mRM*, e.g. for coupling to other environmental models.
- A new output, i.e. fields of routed discharge, is now available. They are stored in `mRM_fluxes_and_states.nc` and are controlled by a new namelist contained in the `mrm_outputs.nml` file, which is an optional file.
- New calibration objectives have been incorporated. It is now possible, additionally to the former objectives, to calibrate mHM against additional input data:

- total water storage (e.g. GRACE) and discharge simultaneously (`opti_function=15`), and/or
- cosmic ray neutron counts (`opti_function=17`).
- New post-processing: a mHM python class for reading all inputs and outputs of a model run can be found in [post-proc/](#).
- Reorganization of the NetCDF writing in mHM to simplify future implementations of additional outputs.

Bugs resolved:

- Calibration with catchment average soil moisture (`opti_function=10`) works properly now.
- Discharge output for multi basin runs with different time periods for each basin works properly now.
- Hargreaves-Samani PET calculation (`processCase(5)=1`) is valid on southern hemisphere too now.

Known bugs:

- None.

Restrictions:

- For `gfortran` compilers mHM supports only v4.8 and higher.
- If you wish to use a special process description of evapotranspiration (`processCase(4)`) please contact [Matthias Zink](#).
- If you wish to use features connected to ground albedo neutrons (`processCase(9)`), please contact [Martin Schrön](#).
- If you wish to use the multi-scale Routing Model as stand-alone version, please contact [Stephan Thober](#).

mHM v5.3 (Jun 2015)

New Features:

- Simulation period and warming days can be now given per basin (see `time_periods` in [mhm.nml](#))
- Enabling use of MPI (set `mpi=true` in `Makefile` and use `#ifdef MPI` for MPI specific code)
- Optional input data can be loaded for example to calibrate against soil moisture (see `optional_data` in [mhm.nml](#))
- Generation of ground albedo cosmic-ray neutrons (see `processCase(10)` in [mhm.nml](#)); these calculations are based on the [COSMIC](#) code, which was originally written by Rafael Rosolem. Please contact [Martin Schrön](#) if you like to use this new feature.
- Several new objective functions, e.g. calibrating the Kling-Gupta efficiency of catchment's average soil moisture (`opti_function=10`) or calibrating multiple basins regarding Kling-Gupta efficiency of discharge (`opti_function=14`) among others; calibration against soil moisture is still purpose to research (`opti_function=10-14`). The interested user may contact [Matthias Zink](#) for further details.

Bugs resolved:

- Calibration using potential evapotranspiration from input file (i.e. `processCase(4)=0`) is now working properly

Known bugs:

- Compiling mHM with the recent Cygwin version under Windows is leading to an error message indicating circular dependencies. The reason for this is Unicode characters in some source code files. Please contact mhm-admin@ufz.de, if you get this error message. We will provide you the files with cleaned characters..

Restrictions:

- If you wish to use a special process description of evapotranspiration (i.e. Hargreaves-samani, Priestley-Taylor, or Penman-Monteith) please contact [Matthias Zink](#). The special cases are set in `mhm.nml` (see `processCase(4)`).
- If you wish to use the new feature of calculating neutron counts please contact [Martin Schrön](#). The feature can be enabled in `mhm.nml` (see `processCase(8)`).

mHM 5.2 (Dec 2014)**New Features:**

- Chunk-wise reading of input data (see `timestep_model_inputs`)
- Complete revision of writing netCDF files
- Possibility to discard multi-scale parameter regionalization (MPR) calculations (see `perform_mpr`)
- Several process descriptions of evapotranspiration implemented (see `processCase(4)`): Read PET, Hargreaves-Samani, Priestley-Taylor, Penman-Monteith. Please contact [Matthias Zink](#) if you use one of the last three options, since the code is not under GNU Public license up to now.
- Adding routines for signature calculations of time series (see `mo_signatures`)
- New objective function for calibrating discharge with Kling-Gupta efficiency measure (KGE, see `opti_function`)
- New output variables (see `mhm_outputs.nml`)
- Sorting algorithm changed to public available library orderpack (see `mo_orderpack`)

Bugs resolved:

- Some variables in restart file where not assigned correctly
- Variables not initialized correctly

Known bugs:

- Calibration using PET values read from the input file (`processCase(5)=0`) is running, but yields wrong results due to a wrong initialization of variables. **The bug is resolved and will be released with version 5.3.**

mHM 5.1 (Jun 2014)**New Features:**

- OpenMP handling of routines such as the multi-scale parameter regionalization (MPR)
- Multi-scale implementation, i.e. running mHM simultaneously in several basins with different resolutions
- Automatic check case framework, i.e. testing new implementations on their validity and back-compatibility

- Implementation of inflow gauges, i.e. feeding discharge time series from upstream areas at catchment boundaries
- File `gaugeinfo.txt` specifying gauging stations is now part of namelist `mhm.nml`
- Code is now free of Numerical Recipes proprietary code
- Can now run on a single cell (no routing performed) Hydrological modelling resolution (L1) equal to morphological input data resolution (L0) possible
- Windows compatible (with Cygwin)
- Support of regular geographic coordinate systems (e.g lat-lon) in addition to equal-area coordinate systems (UTM)

Bugs resolved:

- Initialization of states was not correct when running mHM in calibration mode.
- Calculated parameter values (`mhm_parameters.nml`) not necessarily in bound (check added).
- Aggregation/Disaggregation of meteorological data corrected.
- Forecast with mHM did not work because modelling period was restricted to discharge data period.
- Wrong mapping of evaluation discharge gauges for runs involving multiple gauges.

Known bugs:

- Print out of River network in Config File is wrong for Multi-Basin setup, i.e., the River network is always properly written for the first basin, but not properly for subsequent basins when these are either different ones or the same one with a different Hydrology or Routing resolution.
- mHM does not abort if x-axis of L0 (morphological data) and L2 (meteorological data) do not span over exactly the same range.

mHM 5.0 (Dec 2013)

New Features:

- Full modular version
- Automatic documentation by doxygen
- Running mHM for multiple basin simultaneously
- Definition of 8 major processes:
 - interception,
 - snow,
 - soil moisture,
 - direct runoff,
 - evapotranspiration,
 - interflow,
 - percolation,
 - routing
- Choice of different descriptions of processes possible
- Input in binary `*.bin` or netcdf `*.nc` format
- Various calibration routines and objective functions
- Consistent numerical precision handling of variables

Known bugs:

- None.

Chapter 12

Modules Index

12.1 Modules List

Here is a list of all modules with brief descriptions:

dummy	75
mo_anneal	75
mo_append	
Append values on existing arrays	79
mo_canopy_interc	
Canopy interception	87
mo_common_variables	
Provides structures needed by mHM and mRM	89
mo_constants	
Provides computational, mathematical, physical, and file constants	92
mo_corr	102
mo_dds	
Dynamically Dimensioned Search (DDS)	109
mo_errormeasures	113
mo_file	
Provides file names and units for mHM	136
mo_finish	
Finish a program gracefully	145
mo_global_variables	
Global variables ONLY used in reading, writing and startup	147
mo_init_states	
Initialization of all state variables of mHM	176
mo_julian	
Julian date conversion routines	183
mo_kind	
Define number representations	217
mo_linfit	
Fitting a straight line	220
mo_mcmc	
This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution	221
mo_message	
Write out concatenated strings	224
mo_meteo_forcings	
Prepare meteorological forcings data for mHM	226
mo_mhm	
Call all main processes of mHM	237
mo_mhm_constants	
Provides mHM specific constants	243

mo_mhm_eval	Runs mhm with a specific parameter set and returns required variables, e.g. runoff	255
mo_moment	258
mo_mpr_pet	scaling function for PET correction using LAI at level-0	264
mo_mpr_runoff	Multiscale parameter regionalization for runoff generation	272
mo_mpr_smhorizons	Setting up the soil moisture horizons	275
mo_mpr_soilmoist	Multiscale parameter regionalization (MPR) for soil moisture	279
mo_mrm_constants	289
mo_mrm_eval	Runs mrm with a specific parameter set and returns required variables, e.g. runoff	292
mo_mrm_file	Provides file names and units for mRM	294
mo_mrm_global_variables	Global variables for mRM only	300
mo_mrm_init	Wrapper for initializing Routing	320
mo_mrm_mpr	Perform Multiscale Parameter Regionalization on Routing Parameters	328
mo_mrm_net_startup	Startup drainage network for mHM	330
mo_mrm_objective_function_runoff	Objective Functions for Optimization of mHM/mRM against runoff. 344	
mo_mrm_read_config	Read mRM config	387
mo_mrm_read_data	This module contains all routines to read mRM data from file	391
mo_mrm_read_latlon	Reading latitude and longitude coordinates for each basin	398
mo_mrm_restart	Restart routines	399
mo_mrm_routing	Performs runoff routing for mHM at level L11	404
mo_mrm_signatures	Module with calculations for several hydrological signatures	413
mo_mrm_tools	Provide utility routines used within mRM	423
mo_mrm_write	Write of discharge and restart files	427
mo_mrm_write_fluxes_states	Creates NetCDF output for different fluxes and state variables of mHM	436
mo_multi_param_reg	Multiscale parameter regionalization (MPR)	446
mo_ncread	463
mo_ncwrite	483
mo_netcdf	NetCDF Fortran 90 interface wrapper	510
mo_neutrons	THIS MODULE IS WORK IN PROGRESS, DO NOT USE FOR RESEARCH	566
mo_nml	Deal with namelist files	574
mo_objective_function	Objective Functions for Optimization of mHM	579

mo_optimization	Wrapper subroutine for optimization against runoff and sm	610
mo_orderpack	Sort and ranking routines	613
mo_percentile		630
mo_pet	Module for calculating reference/potential evapotranspiration [mm s-1]	632
mo_prepare_gridded_lai	Prepare daily LAI fields (e.g., MODIS data) for mHM	644
mo_read_config	Reading of main model configurations	647
mo_read_forcing_nc	Reads forcing input data	649
mo_read_latlon	Reading latitude and longitude coordinates for each basin	656
mo_read_lut	Routines reading lookup tables (lut)	658
mo_read_meteo	Reads meteorological input data	660
mo_read_optional_data	Read optional data for mHM calibration	662
mo_read_spatial_data	Reads spatial input data	667
mo_read_timeseries	Routines to read files containing timeseries data	670
mo_read_wrapper	Wrapper for all reading routines	672
mo_restart	Reading and writing states, fluxes and configuration for restart of mHM	675
mo_runoff	Runoff generation for the unsaturated zone, saturated zone (or groundwater zone), and runoff accumulation	681
mo_sce	Shuffled Complex Evolution optimization algorithm	686
mo_set_netcdf_outputs	Defines the structure of the netCDF to write the output in	695
mo_snow_accum_melt	Snow melting and accumulation	696
mo_soil_database	Generating soil database from input file	698
mo_soil_moisture	Soil moisture of the different layers	701
mo_spatial_agg_disagg_forcing	Spatial aggregation or disaggregation of meteorological input data	708
mo_spatialsimilarity	Routines for bias insensitive comparison of spatial patterns	710
mo_standard_score	Routines for calculating the normalization (anomaly)/standard score/z score and the deseasonalized (standard score on monthly basis) values of a time series	711
mo_startup	Startup procedures for mHM	713
mo_string_utils	String utilities	720
mo_template	Template for future module developments	728
mo_temporal_aggregation	Temporal aggregation for time series (averaging)	730

mo_temporal_disagg_forcing	
Temporal disaggregation of daily input values	731
mo_timer	
Timing routines	734
mo_upscaling_operators	
Module containing upscaling operators	743
mo_utils	
General utilities for the CHS library	751
mo_write_ascii	
Module to write ascii file output	757
mo_write_fluxes_states	
Creates NetCDF output for different fluxes and state variables of mHM	760
mo_xor4096	775

Chapter 13

Data Type Index

13.1 Data Types List

Here are the data types with brief descriptions:

mo_moment::absdev	779
mo_anneal::anneal	
Anneal	779
mo_append::append	
Append (rows) scalars, vectors, and matrixes onto existing array	782
mo_corr::arth	786
mo_ncwrite::attribute	788
mo_corr::autocoeffk	789
mo_corr::autocorr	790
mo_moment::average	790
mo_global_variables::basininfo	791
mo_mrm_global_variables::basininfo	794
mo_errormeasures::bias	798
mo_moment::central_moment	799
mo_moment::central_moment_var	800
mo_standard_score::classified_standard_score	
Calculates the classified standard score (e.g. classes are months)	801
mo_corr::corr	802
mo_moment::correlation	803
mo_moment::covariance	803
mo_corr::crosscoeffk	804
mo_corr::crosscorr	804
mo_orderpack::ctrper	805
mo_temporal_aggregation::day2mon_average	
Day-to-month average (day2mon_average)	806
mo_ncwrite::dims	807
mo_ncwrite::dump_netcdf	808
mo_utils::eq	811
mo_utils::equal	
Comparison of real values	812
mo_orderpack::fndnth	813
mo_corr::four1	814
mo_corr::fourrow	814
mo_mrm_global_variables::gaugingstation	815
mo_utils::ge	816
mo_anneal::generate_neighborhood_weight	817
mo_ncread::get_ncvar	817
mo_xor4096::get_timeseed	823

mo_anneal::gettemperature	
GetTemperature	823
mo_utils::greaterequal	825
mo_global_variables::gridgeoref	826
mo_mrm_global_variables::gridgeoref	827
mo_temporal_aggregation::hour2day_average	
Hour-to-day average (hour2day_average)	828
mo_orderpack::indmed	829
mo_orderpack::indnth	830
mo_orderpack::inspar	831
mo_orderpack::inssor	832
mo_utils::is_finite	
.true. if not IEEE Inf, IEEE NaN, nor IEEE Inf nor IEEE NaN, respectively	832
mo_utils::is_nan	833
mo_utils::is_normal	834
mo_errormeasures::kge	
Kling-Gupta-Efficiency measure	834
mo_errormeasures::kgenocorr	
Kling-Gupta-Efficiency measure without correlation	836
mo_moment::kurtosis	838
mo_utils::le	839
mo_utils::lesserequal	840
mo_linfit::linfit	
Fits a straight line to input data by minimizing χ^2	840
mo_errormeasures::lnnse	841
mo_utils::locate	
Find closest values in a monotonic series, returns the indexes	843
mo_errormeasures::mae	844
mo_mcmc::mcmc	
This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are either known or modeled)	845
mo_mcmc::mcmc_stddev	
This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are neither known nor modeled)	849
mo_moment::mean	853
mo_template::mean	
The average	854
mo_percentile::median	855
mo_moment::mixed_central_moment	856
mo_moment::mixed_central_moment_var	856
mo_moment::moment	857
mo_orderpack::mrgref	858
mo_orderpack::mrgrnk	859
mo_errormeasures::mse	859
mo_orderpack::mulcnt	861
mo_percentile::n_element	861
mo_netcdf::ncdataset	
Provides basic file modification functionality	862
mo_netcdf::ncdimension	
Provides the dimension access functionality	873
mo_netcdf::ncvariable	890
mo_utils::ne	890
mo_orderpack::nearless	891
mo_spatialsimilarity::nndv	
Calculates the number of neighboring dominating values, a measure for spatial dissimilarity	891
mo_utils::notequal	893
mo_errormeasures::nse	894

mo_string_utils::num2str	
Convert to string	895
mo_string_utils::numarray2str	
Convert to string	897
mo_orderpack::omedian	898
mo_write_fluxes_states::outputdataset	899
mo_mrm_write_fluxes_states::outputdataset	903
mo_write_fluxes_states::outputvariable	907
mo_mrm_write_fluxes_states::outputvariable	912
mo_append::paste	
Paste (columns) scalars, vectors, and matrixes onto existing array	916
mo_spatialsimilarity::pd	
Calculates pattern dissimilarity (PD) measure	920
mo_percentile::percentile	922
mo_common_variables::period	923
mo_percentile::qmedian	925
mo_orderpack::rapknr	925
mo_read_spatial_data::read_spatial_data_ascii	
Reads spatial data files of ASCII format	926
mo_corr::realft	927
mo_orderpack::refpar	928
mo_orderpack::refsor	929
mo_orderpack::rinpar	929
mo_errormeasures::rmse	930
mo_orderpack::rnkpar	932
mo_errormeasures::sae	932
mo_julian::setcalendar	934
mo_moment::skewness	935
mo_global_variables::soiltype	936
mo_orderpack::sort	
Unconditional ranking	939
mo_orderpack::sort_index	942
mo_spatial_agg_disagg_forcing::spatial_aggregation	
Spatial aggregation of meterological variables	943
mo_spatial_agg_disagg_forcing::spatial_disaggregation	
Spatial disaggregation of meterological variables	944
mo_utils::special_value	
Special IEEE values	945
mo_kind::sprs2_dp	
Double Precision Numerical Recipes types for sparse arrays	946
mo_kind::sprs2_sp	
Single Precision Numerical Recipes types for sparse arrays	948
mo_errormeasures::sse	949
mo_standard_score::standard_score	
Calculates the standard score / normalization (anomaly) / z-score	950
mo_moment::stddev	952
mo_corr::swap	952
mo_utils::swap	
Swap to values or two elements in array	953
mo_global_variables::twssstructure	955
mo_orderpack::uniinv	956
mo_orderpack::unipar	956
mo_orderpack::unirnk	957
mo_orderpack::unista	958
mo_orderpack::valmed	959
mo_orderpack::valnth	959
mo_ncwrite::var2nc	
Extended dump_netcdf for multiple variables	960

mo_ncwrite::variable	967
mo_moment::variance	972
mo_xor4096::xor4096	973
mo_xor4096::xor4096g	974

Chapter 14

File Index

14.1 File List

Here is a list of all files with brief descriptions:

mhm_driver.f90	977
mo_anneal.f90	980
mo_append.f90	981
mo_canopy_interc.f90	982
mo_common_variables.f90	982
mo_constants.f90	983
mo_corr.f90	985
mo_dds.f90	986
mo_errormeasures.f90	987
mo_file.f90	988
mo_finish.f90	991
mo_global_variables.f90	991
mo_init_states.f90	995
mo_julian.f90	995
mo_kind.f90	996
mo_linfit.f90	997
mo_mcmc.f90	997
mo_message.f90	998
mo_meteo_forcings.f90	998
mo_mhm.f90	999
mo_mhm_constants.f90	999
mo_mhm_eval.f90	1001
mo_moment.f90	1001
mo_mpr_pet.f90	1002
mo_mpr_runoff.f90	1003
mo_mpr_smhorizons.f90	1003
mo_mpr_soilmoist.f90	1003
mo_mrm_constants.f90	
Provides mRM specific constants	1004
mo_mrm_eval.f90	1005
mo_mrm_file.f90	1005
mo_mrm_global_variables.f90	1006
mo_mrm_init.f90	1009
mo_mrm_mpr.f90	1009
mo_mrm_net_startup.f90	1009
mo_mrm_objective_function_runoff.f90	1010
mo_mrm_read_config.f90	1011
mo_mrm_read_data.f90	1012

mo_mrm_read_latlon.f90	1012
mo_mrm_restart.f90	1012
mo_mrm_routing.f90	1013
mo_mrm_signatures.f90	1013
mo_mrm_tools.f90	1014
mo_mrm_write.f90	1014
mo_mrm_write_fluxes_states.f90	1015
mo_multi_param_reg.f90	1016
mo_ncread.f90	1016
mo_ncwrite.f90	1017
mo_netcdf.f90	1019
mo_neutrons.f90	
Models to predict neutron intensities above soils	1021
mo_nml.f90	1022
mo_objective_function.f90	1023
mo_optimization.f90	1023
mo_orderpack.f90	1024
mo_percentile.f90	1026
mo_pet.f90	1026
mo_prepare_gridded_lai.f90	1027
mo_read_config.f90	1027
mo_read_forcing_nc.f90	1027
mo_read_latlon.f90	1028
mo_read_lut.f90	1028
mo_read_meteo.f90	1028
mo_read_optional_data.f90	1029
mo_read_spatial_data.f90	1029
mo_read_timeseries.f90	1029
mo_read_wrapper.f90	1030
mo_restart.f90	1030
mo_runoff.f90	1030
mo_sce.f90	1031
mo_set_netcdf_outputs.f90	1033
mo_snow_accum_melt.f90	1033
mo_soil_database.f90	1034
mo_soil_moisture.f90	1034
mo_spatial_agg_disagg_forcing.f90	1035
mo_spatialsimilarity.f90	1035
mo_standard_score.f90	1036
mo_startup.f90	1036
mo_string_utils.f90	1037
mo_template.f90	1037
mo_temporal_aggregation.f90	1038
mo_temporal_disagg_forcing.f90	1038
mo_timer.f90	1039
mo_upscaling_operators.f90	1039
mo_utils.f90	1040
mo_write_ascii.f90	1041
mo_write_fluxes_states.f90	1041
mo_xor4096.f90	1042
mrm_driver.f90	1043

Chapter 15

Module Documentation

15.1 dummy Module Reference

15.2 mo_anneal Module Reference

Data Types

- interface [anneal](#)
anneal
- interface [generate_neighborhood_weight](#)
- interface [gettemperature](#)
GetTemperature.

Functions/Subroutines

- real(dp) function, dimension(size(para, 1)) [anneal_dp](#) (cost, para, prange, prange_func, temp, Dt, nITERmax, Len, nST, eps, acc, seeds, printflag, maskpara, weight, changeParaMode, reflectionFlag, pertubFlexFlag, maxit, undef_funcval, tmp_file, funcbest, history)
- real(dp) function [gettemperature_dp](#) (paraset, cost, acc_goal, prange, prange_func, samplesize, maskpara, seeds, printflag, weight, maxit, undef_funcval)
- real(dp) function [pargen_anneal_dp](#) (old, dMax, oMin, oMax, RN)
- real(dp) function [pargen_dds_dp](#) (old, perturb, oMin, oMax, RN)
- real(dp) function [dchange_dp](#) (delta, iDigit, isZero)
- subroutine [generate_neighborhood_weight_dp](#) (truepara, cum_weight, save_state_xor, iTotalCounter, nIT↔ERmax, neighborhood)

15.2.1 Function/Subroutine Documentation

15.2.1.1 anneal_dp()

```
real(dp) function, dimension(size(para,1)) mo_anneal::anneal_dp (  
    cost,  
    real(dp), dimension(:), intent(in) para,  
    real(dp), dimension(size(para,1),2), intent(in), optional prange,  
    optional prange_func,  
    real(dp), intent(in), optional temp,
```

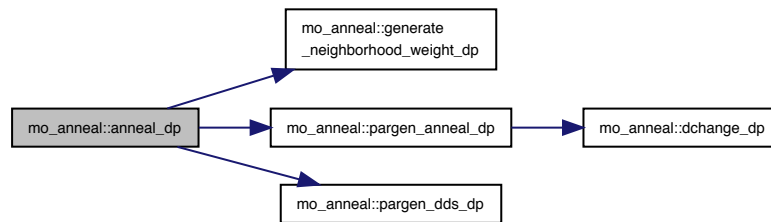
```

real(dp), intent(in), optional Dt,
integer(i4), intent(in), optional nITERmax,
integer(i4), intent(in), optional Len,
integer(i4), intent(in), optional nST,
real(dp), intent(in), optional eps,
real(dp), intent(in), optional acc,
integer(i8), dimension(3), intent(in), optional seeds,
logical, intent(in), optional printfFlag,
logical, dimension(size(para,1)), intent(in), optional maskpara,
real(dp), dimension(size(para,1)), intent(in), optional weight,
integer(i4), intent(in), optional changeParaMode,
logical, intent(in), optional reflectionFlag,
logical, intent(in), optional pertubFlexFlag,
logical, intent(in), optional maxit,
real(dp), intent(in), optional undef_funcval,
character(len=*), intent(in), optional tmp_file,
real(dp), intent(out), optional funcbest,
real(dp), dimension(:, :), intent(out), optional, allocatable history ) [private]

```

References `generate_neighborhood_weight_dp()`, `pargen_anneal_dp()`, and `pargen_dds_dp()`.

Here is the call graph for this function:



15.2.1.2 dchange_dp()

```

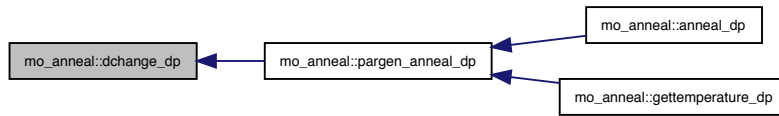
real(dp) function mo_anneal::dchange_dp (
    real(dp), intent(in) delta,
    integer(i4), intent(in) iDigit,
    integer(i4), intent(in) isZero ) [private]

```

References `mo_kind::dp`, `mo_kind::i4`, and `mo_kind::i8`.

Referenced by `pargen_anneal_dp()`.

Here is the caller graph for this function:



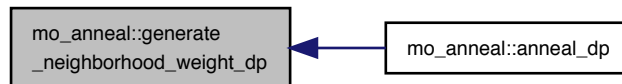
15.2.1.3 generate_neighborhood_weight_dp()

```

subroutine mo_anneal::generate_neighborhood_weight_dp (
    integer(i4), dimension(:), intent(in) truepara,
    real(dp), dimension(:), intent(in) cum_weight,
    integer(i8), dimension(n_save_state), intent(inout) save_state_xor,
    integer(i4), intent(in) iTotalCounter,
    integer(i4), intent(in) nITERmax,
    logical, dimension(size(cum_weight)), intent(out) neighborhood )
  
```

Referenced by `anneal_dp()`.

Here is the caller graph for this function:



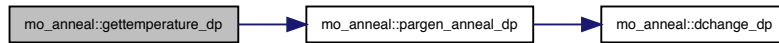
15.2.1.4 gettemperature_dp()

```

real(dp) function mo_anneal::gettemperature_dp (
    real(dp), dimension(:), intent(in) paraset,
    cost,
    real(dp), intent(in) acc_goal,
    real(dp), dimension(size(paraset,1),2), intent(in), optional prange,
    optional prange_func,
    integer(i4), intent(in), optional samplesize,
    logical, dimension(size(paraset,1)), intent(in), optional maskpara,
    integer(i8), dimension(2), intent(in), optional seeds,
    logical, intent(in), optional printflag,
    real(dp), dimension(size(paraset,1)), intent(in), optional weight,
    logical, intent(in), optional maxit,
    real(dp), intent(in), optional undef_funcval )
  
```

References `mo_kind::dp`, `mo_kind::i4`, `mo_kind::i8`, and `pargen_anneal_dp()`.

Here is the call graph for this function:



15.2.1.5 pargen_anneal_dp()

```

real(dp) function mo_anneal::pargen_anneal_dp (
    real(dp), intent(in) old,
    real(dp), intent(in) dMax,
    real(dp), intent(in) oMin,
    real(dp), intent(in) oMax,
    real(dp), intent(in) RN )
  
```

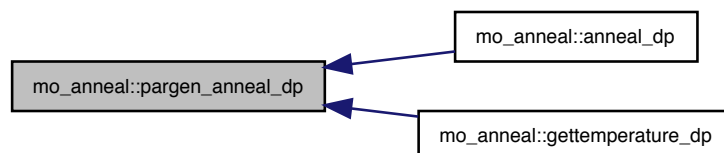
References `dchange_dp()`, `mo_kind::dp`, and `mo_kind::i4`.

Referenced by `anneal_dp()`, and `gettemperature_dp()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.2.1.6 pargen_dds_dp()

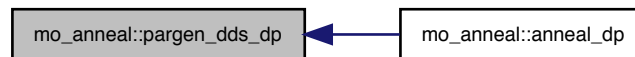
```

real(dp) function mo_anneal::pargen_dds_dp (
    real(dp), intent(in) old,
    real(dp), intent(in) perturb,
    real(dp), intent(in) oMin,
    real(dp), intent(in) oMax,
    real(dp), intent(in) RN )

```

Referenced by anneal_dp().

Here is the caller graph for this function:



15.3 mo_append Module Reference

Append values on existing arrays.

Data Types

- interface [append](#)
Append (rows) scalars, vectors, and matrixes onto existing array.
- interface [paste](#)
Paste (columns) scalars, vectors, and matrixes onto existing array.

Functions/Subroutines

- subroutine [append_i4_v_s](#) (vec1, sca2)
- subroutine [append_i4_v_v](#) (vec1, vec2)
- subroutine [append_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_i4_3d](#) (mat1, mat2, fill_value)
- subroutine [append_i8_v_s](#) (vec1, sca2)
- subroutine [append_i8_v_v](#) (vec1, vec2)
- subroutine [append_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_i8_3d](#) (mat1, mat2, fill_value)
- subroutine [append_sp_v_s](#) (vec1, sca2)
- subroutine [append_sp_v_v](#) (vec1, vec2)
- subroutine [append_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_sp_3d](#) (mat1, mat2, fill_value)
- subroutine [append_dp_v_s](#) (vec1, sca2)
- subroutine [append_dp_v_v](#) (vec1, vec2)
- subroutine [append_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_dp_3d](#) (mat1, mat2, fill_value)
- subroutine [append_char_v_s](#) (vec1, sca2)
- subroutine [append_char_v_v](#) (vec1, vec2)

- subroutine [append_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_char_3d](#) (mat1, mat2, fill_value)
- subroutine [append_lgt_v_s](#) (vec1, sca2)
- subroutine [append_lgt_v_v](#) (vec1, vec2)
- subroutine [append_lgt_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_lgt_3d](#) (mat1, mat2, fill_value)
- subroutine [paste_i4_m_s](#) (mat1, sca2)
- subroutine [paste_i4_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_i8_m_s](#) (mat1, sca2)
- subroutine [paste_i8_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_sp_m_s](#) (mat1, sca2)
- subroutine [paste_sp_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_dp_m_s](#) (mat1, sca2)
- subroutine [paste_dp_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_char_m_s](#) (mat1, sca2)
- subroutine [paste_char_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_lgt_m_s](#) (mat1, sca2)
- subroutine [paste_lgt_m_v](#) (mat1, vec2)
- subroutine [paste_lgt_m_m](#) (mat1, mat2)

15.3.1 Detailed Description

Append values on existing arrays.

Provides routines to append (rows) and paste (columns) scalars, vectors, and matrixes onto existing arrays.

Author

Juliane Mai

Date

Aug 2012

15.3.2 Function/Subroutine Documentation

15.3.2.1 `append_char_3d()`

```
subroutine mo_append::append_char_3d (
    character(len=*), dimension(:, :, :), intent(inout), allocatable mat1,
    character(len=*), dimension(:, :, :), intent(in) mat2,
    character(len=*), intent(in), optional fill_value ) [private]
```

15.3.2.2 append_char_m_m()

```

subroutine mo_append::append_char_m_m (
    character(len=*), dimension(:, :), intent(inout), allocatable mat1,
    character(len=*), dimension(:, :), intent(in) mat2,
    character(len=*), intent(in), optional fill_value ) [private]

```

15.3.2.3 append_char_v_s()

```

subroutine mo_append::append_char_v_s (
    character(len=*), dimension(:), intent(inout), allocatable vec1,
    character(len=*), intent(in) sca2 ) [private]

```

15.3.2.4 append_char_v_v()

```

subroutine mo_append::append_char_v_v (
    character(len=*), dimension(:), intent(inout), allocatable vec1,
    character(len=*), dimension(:), intent(in) vec2 ) [private]

```

15.3.2.5 append_dp_3d()

```

subroutine mo_append::append_dp_3d (
    real(dp), dimension(:, :, :), intent(inout), allocatable mat1,
    real(dp), dimension(:, :, :), intent(in) mat2,
    real(dp), intent(in), optional fill_value ) [private]

```

15.3.2.6 append_dp_m_m()

```

subroutine mo_append::append_dp_m_m (
    real(dp), dimension(:, :), intent(inout), allocatable mat1,
    real(dp), dimension(:, :), intent(in) mat2,
    real(dp), intent(in), optional fill_value ) [private]

```

15.3.2.7 append_dp_v_s()

```

subroutine mo_append::append_dp_v_s (
    real(dp), dimension(:), intent(inout), allocatable vec1,
    real(dp), intent(in) sca2 ) [private]

```

15.3.2.8 append_dp_v_v()

```

subroutine mo_append::append_dp_v_v (
    real(dp), dimension(:), intent(inout), allocatable vec1,
    real(dp), dimension(:), intent(in) vec2 ) [private]

```

15.3.2.9 append_i4_3d()

```

subroutine mo_append::append_i4_3d (
    integer(i4), dimension(:,:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:,:,:), intent(in) mat2,
    integer(i4), intent(in), optional fill_value ) [private]

```

15.3.2.10 append_i4_m_m()

```

subroutine mo_append::append_i4_m_m (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:,:), intent(in) mat2,
    integer(i4), intent(in), optional fill_value ) [private]

```

15.3.2.11 append_i4_v_s()

```

subroutine mo_append::append_i4_v_s (
    integer(i4), dimension(:), intent(inout), allocatable vec1,
    integer(i4), intent(in) sca2 ) [private]

```

15.3.2.12 append_i4_v_v()

```

subroutine mo_append::append_i4_v_v (
    integer(i4), dimension(:), intent(inout), allocatable vec1,
    integer(i4), dimension(:), intent(in) vec2 ) [private]

```

15.3.2.13 append_i8_3d()

```

subroutine mo_append::append_i8_3d (
    integer(i8), dimension(:,:,:), intent(inout), allocatable mat1,
    integer(i8), dimension(:,:,:), intent(in) mat2,
    integer(i8), intent(in), optional fill_value ) [private]

```

15.3.2.14 append_i8_m_m()

```

subroutine mo_append::append_i8_m_m (
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,
    integer(i8), dimension(:,:), intent(in) mat2,
    integer(i8), intent(in), optional fill_value ) [private]

```

15.3.2.15 append_i8_v_s()

```
subroutine mo_append::append_i8_v_s (
    integer(i8), dimension(:), intent(inout), allocatable vec1,
    integer(i8), intent(in) sca2 ) [private]
```

15.3.2.16 append_i8_v_v()

```
subroutine mo_append::append_i8_v_v (
    integer(i8), dimension(:), intent(inout), allocatable vec1,
    integer(i8), dimension(:), intent(in) vec2 ) [private]
```

15.3.2.17 append_lgt_3d()

```
subroutine mo_append::append_lgt_3d (
    logical, dimension(:,:,:), intent(inout), allocatable mat1,
    logical, dimension(:,:,:), intent(in) mat2,
    logical, intent(in), optional fill_value ) [private]
```

15.3.2.18 append_lgt_m_m()

```
subroutine mo_append::append_lgt_m_m (
    logical, dimension(:,:), intent(inout), allocatable mat1,
    logical, dimension(:,:), intent(in) mat2,
    logical, intent(in), optional fill_value ) [private]
```

15.3.2.19 append_lgt_v_s()

```
subroutine mo_append::append_lgt_v_s (
    logical, dimension(:), intent(inout), allocatable vec1,
    logical, intent(in) sca2 ) [private]
```

15.3.2.20 append_lgt_v_v()

```
subroutine mo_append::append_lgt_v_v (
    logical, dimension(:), intent(inout), allocatable vec1,
    logical, dimension(:), intent(in) vec2 ) [private]
```

15.3.2.21 append_sp_3d()

```
subroutine mo_append::append_sp_3d (
    real(sp), dimension(:,:,:), intent(inout), allocatable mat1,
    real(sp), dimension(:,:,:), intent(in) mat2,
    real(sp), intent(in), optional fill_value ) [private]
```

15.3.2.22 append_sp_m_m()

```
subroutine mo_append::append_sp_m_m (
    real(sp), dimension(:,:), intent(inout), allocatable mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    real(sp), intent(in), optional fill_value ) [private]
```

15.3.2.23 append_sp_v_s()

```
subroutine mo_append::append_sp_v_s (
    real(sp), dimension(:), intent(inout), allocatable vec1,
    real(sp), intent(in) sca2 ) [private]
```

15.3.2.24 append_sp_v_v()

```
subroutine mo_append::append_sp_v_v (
    real(sp), dimension(:), intent(inout), allocatable vec1,
    real(sp), dimension(:), intent(in) vec2 ) [private]
```

15.3.2.25 paste_char_m_m()

```
subroutine mo_append::paste_char_m_m (
    character(len=*), dimension(:,:), intent(inout), allocatable mat1,
    character(len=*), dimension(:,:), intent(in) mat2,
    character(len=*), intent(in), optional fill_value ) [private]
```

15.3.2.26 paste_char_m_s()

```
subroutine mo_append::paste_char_m_s (
    character(len=*), dimension(:,:), intent(inout), allocatable mat1,
    character(len=*), intent(in) sca2 ) [private]
```

15.3.2.27 paste_char_m_v()

```
subroutine mo_append::paste_char_m_v (
    character(len=*), dimension(:,:), intent(inout), allocatable mat1,
    character(len=*), dimension(:), intent(in) vec2,
    character(len=*), intent(in), optional fill_value ) [private]
```

15.3.2.28 paste_dp_m_m()

```
subroutine mo_append::paste_dp_m_m (
```

```

real(dp), dimension(:,:), intent(inout), allocatable mat1,
real(dp), dimension(:,:), intent(in) mat2,
real(dp), intent(in), optional fill_value ) [private]

```

15.3.2.29 paste_dp_m_s()

```

subroutine mo_append::paste_dp_m_s (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), intent(in) sca2 ) [private]

```

15.3.2.30 paste_dp_m_v()

```

subroutine mo_append::paste_dp_m_v (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), dimension(:), intent(in) vec2,
    real(dp), intent(in), optional fill_value ) [private]

```

15.3.2.31 paste_i4_m_m()

```

subroutine mo_append::paste_i4_m_m (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:,:), intent(in) mat2,
    integer(i4), intent(in), optional fill_value ) [private]

```

15.3.2.32 paste_i4_m_s()

```

subroutine mo_append::paste_i4_m_s (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), intent(in) sca2 ) [private]

```

15.3.2.33 paste_i4_m_v()

```

subroutine mo_append::paste_i4_m_v (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:), intent(in) vec2,
    integer(i4), intent(in), optional fill_value ) [private]

```

15.3.2.34 paste_i8_m_m()

```

subroutine mo_append::paste_i8_m_m (
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,
    integer(i8), dimension(:,:), intent(in) mat2,
    integer(i8), intent(in), optional fill_value ) [private]

```

15.3.2.35 paste_i8_m_s()

```
subroutine mo_append::paste_i8_m_s (
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,
    integer(i8), intent(in) sca2 ) [private]
```

15.3.2.36 paste_i8_m_v()

```
subroutine mo_append::paste_i8_m_v (
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,
    integer(i8), dimension(:), intent(in) vec2,
    integer(i8), intent(in), optional fill_value ) [private]
```

15.3.2.37 paste_lgt_m_m()

```
subroutine mo_append::paste_lgt_m_m (
    logical, dimension(:,:), intent(inout), allocatable mat1,
    logical, dimension(:,:), intent(in) mat2 ) [private]
```

15.3.2.38 paste_lgt_m_s()

```
subroutine mo_append::paste_lgt_m_s (
    logical, dimension(:,:), intent(inout), allocatable mat1,
    logical, intent(in) sca2 ) [private]
```

15.3.2.39 paste_lgt_m_v()

```
subroutine mo_append::paste_lgt_m_v (
    logical, dimension(:,:), intent(inout), allocatable mat1,
    logical, dimension(:), intent(in) vec2 ) [private]
```

15.3.2.40 paste_sp_m_m()

```
subroutine mo_append::paste_sp_m_m (
    real(sp), dimension(:,:), intent(inout), allocatable mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    real(sp), intent(in), optional fill_value ) [private]
```

15.3.2.41 paste_sp_m_s()

```
subroutine mo_append::paste_sp_m_s (
    real(sp), dimension(:,:), intent(inout), allocatable mat1,
    real(sp), intent(in) sca2 ) [private]
```

15.3.2.42 paste_sp_m_v()

```
subroutine mo_append::paste_sp_m_v (
    real(sp), dimension(:, :), intent(inout), allocatable mat1,
    real(sp), dimension(:), intent(in) vec2,
    real(sp), intent(in), optional fill_value ) [private]
```

15.4 mo_canopy_interc Module Reference

Canopy interception.

Functions/Subroutines

- elemental pure subroutine, public [canopy_interc](#) (pet, interc_max, precip, interc, throughfall, evap_canopy)
Canopy interception.

15.4.1 Detailed Description

Canopy interception.

This module deals with processes related to canopy interception, evaporation and throughfall.

Authors

Vladyslav Prykhodko

Date

Dec 2012

15.4.2 Function/Subroutine Documentation

15.4.2.1 canopy_interc()

```
elemental pure subroutine, public mo_canopy_interc::canopy_interc (
    real(dp), intent(in) pet,
    real(dp), intent(in) interc_max,
    real(dp), intent(in) precip,
    real(dp), intent(inout) interc,
    real(dp), intent(out) throughfall,
    real(dp), intent(out) evap_canopy )
```

Canopy interception.

Calculates throughfall. Updates interception and evaporation intensity from canopy.

Throughfall (F) is estimated as a function of the incoming precipitation (P), the current status of the canopy water content (C), and the max. water

$$F = \text{Max}((P + C - C_{\text{max}}), 0)$$

Evaporation (E) from canopy is estimated as a fraction of the potential evapotranspiration(E_p) depending on the current status of the canopy water content (C) and the max. water content(C_{max}) that can be intercepted by the vegetation.

$$E = E_p (C / C_{\text{max}})^{2/3}$$

15.5 mo_common_variables Module Reference

Provides structures needed by mHM and mRM.

Data Types

- type [period](#)

Variables

- integer(i4), parameter, public [nprocesses](#) = 10
- integer(i4), dimension([nprocesses](#), 3), public [processmatrix](#)
- integer(i4), public [opti_method](#)
- integer(i4), public [opti_function](#)
- logical, public [optimize](#)
- logical, public [optimize_restart](#)
- integer(i8), public [seed](#)
- integer(i4), public [niterations](#)
- real(dp), public [dds_r](#)
- real(dp), public [sa_temp](#)
- integer(i4), public [sce_ngs](#)
- integer(i4), public [sce_npg](#)
- integer(i4), public [sce_nps](#)
- logical, public [mcmc_opti](#)
- integer(i4), parameter, public [nerror_model](#) = 2
- real(dp), dimension([nerror_model](#)), public [mcmc_error_params](#)
- real(dp), dimension(:, :), allocatable, public [global_parameters](#)
- character(256), dimension(:), allocatable, public [global_parameters_name](#)
- logical [alma_convention](#)

15.5.1 Detailed Description

Provides structures needed by mHM and mRM.

Provides the global structure `period` that is used by both mHM and mRM.

Author

Stephan Thober

Date

Sep 2015

15.5.2 Variable Documentation

15.5.2.1 `alma_convention`

```
logical mo_common_variables::alma_convention
```

Referenced by `mo_mrm_read_data::mrm_read_total_runoff()`.

15.5.2.2 dds_r

```
real(dp), public mo_common_variables::dds_r
```

15.5.2.3 global_parameters

```
real(dp), dimension(:, :), allocatable, public mo_common_variables::global_parameters
```

Referenced by `mo_mrm_objective_function_runoff::loglikelihood_evin2013_2()`, `mrm_driver()`, `mo_mrm_init::mrm_init()`, `mo_mrm_read_config::read_mrm_routing_params()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.5.2.4 global_parameters_name

```
character(256), dimension(:), allocatable, public mo_common_variables::global_parameters_name
```

Referenced by `mrm_driver()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.5.2.5 mcmc_error_params

```
real(dp), dimension(nerror_model), public mo_common_variables::mcmc_error_params
```

15.5.2.6 mcmc_opti

```
logical, public mo_common_variables::mcmc_opti
```

15.5.2.7 nerror_model

```
integer(i4), parameter, public mo_common_variables::nerror_model = 2
```

15.5.2.8 niterations

```
integer(i4), public mo_common_variables::niterations
```

15.5.2.9 nprocesses

```
integer(i4), parameter, public mo_common_variables::nprocesses = 10
```

Referenced by `mo_read_config::read_config()`, `mo_mrm_read_config::read_mrm_config()`, and `mo_write_ascii::write_optinamelist()`.

15.5.2.10 opti_function

```
integer(i4), public mo_common_variables::opti_function
```

Referenced by mo_mrm_objective_function_runoff::loglikelihood(), mo_mrm_objective_function_runoff::multi_↵
objective_runoff(), mo_objective_function::objective(), and mo_mrm_objective_function_runoff::single_objective_↵
runoff().

15.5.2.11 opti_method

```
integer(i4), public mo_common_variables::opti_method
```

Referenced by mo_mrm_objective_function_runoff::loglikelihood(), and mo_optimization::optimization().

15.5.2.12 optimize

```
logical, public mo_common_variables::optimize
```

Referenced by mo_mhm_eval::mhm_eval(), mrm_driver(), mo_mrm_eval::mrm_eval(), mo_mrm_read_data_↵
::mrm_read_discharge(), mo_read_optional_data::read_basin_avg_tws(), and mo_mrm_read_config::read_mrm_↵
_config().

15.5.2.13 optimize_restart

```
logical, public mo_common_variables::optimize_restart
```

15.5.2.14 processmatrix

```
integer(i4), dimension(nprocesses, 3), public mo_common_variables::processmatrix
```

Referenced by mo_startup::constants_init(), mo_mrm_net_startup::l11_stream_features(), mhm_driver(), mo_↵
mhm_eval::mhm_eval(), mo_mrm_eval::mrm_eval(), mo_mrm_init::mrm_init(), mo_mrm_init::mrm_init_param(),
mo_mrm_read_data::mrm_read_l0_data(), mo_mrm_restart::mrm_read_restart_config(), mo_mrm_write::mrm_↵
write_optinamelist(), mo_mrm_restart::mrm_write_restart(), mo_meteo_forcings::prepare_meteo_forcings_data(),
mo_read_config::read_config(), mo_read_wrapper::read_data(), mo_mrm_read_config::read_mrm_config(), mo_↵
_mrm_read_config::read_mrm_routing_params(), mo_restart::read_restart_states(), mo_write_ascii::write_↵
configfile(), mo_mrm_write::write_configfile(), and mo_restart::write_restart_files().

15.5.2.15 sa_temp

```
real(dp), public mo_common_variables::sa_temp
```

15.5.2.16 sce_ngs

```
integer(i4), public mo_common_variables::sce_ngs
```


- radian to conversion (180/pi) in double precision*
- real(sp), parameter `rad2deg_sp` = 180._sp/PI_sp
- radian to degree conversion (180/pi) in single precision*
- real(dp), parameter `secday_dp` = 86400._dp
- Seconds per day [s] in double precision.*
- real(sp), parameter `secday_sp` = 86400._sp
- Seconds per day [s] in single precision.*
- real(dp), parameter `psychro_dp` = 0.0646_dp
- Psychrometric constant [kPa K⁻¹] in double precision.*
- real(sp), parameter `psychro_sp` = 0.0646_sp
- Psychrometric constant [kPa K⁻¹] in single precision.*
- real(dp), parameter `gravity_dp` = 9.81_dp
- Gravity acceleration [m² s⁻¹] in double precision.*
- real(sp), parameter `gravity_sp` = 9.81_sp
- Gravity acceleration [m² s⁻¹] in single precision.*
- real(dp), parameter `solarconst_dp` = 1367._dp
- Solar constant in [J m⁻² s⁻¹] in double precision.*
- real(sp), parameter `solarconst_sp` = 1367._sp
- Solar constant in [J m⁻² s⁻¹] in single precision.*
- real(dp), parameter `specheatet_dp` = 2.45e06_dp
- Specific heat for vaporization of water in [J m⁻² mm⁻¹] in double precision.*
- real(sp), parameter `specheatet_sp` = 2.45e06_sp
- Specific heat for vaporization of water in [J m⁻² mm⁻¹] in single precision.*
- real(dp), parameter `t0_dp` = 273.15_dp
- Standard temperature [K] in double precision.*
- real(sp), parameter `t0_sp` = 273.15_sp
- Standard temperature [K] in single precision.*
- real(dp), parameter `sigma_dp` = 5.67e-08_dp
- Stefan-Boltzmann constant [W m⁻² K⁻⁴] in double precision.*
- real(sp), parameter `sigma_sp` = 5.67e-08_sp
- Stefan-Boltzmann constant [W m⁻² K⁻⁴] in single precision.*
- real(sp), parameter `radiusearth_sp` = 6371228._sp
- real(dp), parameter `radiusearth_dp` = 6371228._dp
- real(dp), parameter `p0_dp` = 101325._dp
- standard atmosphere Standard pressure [Pa] in double precision*
- real(sp), parameter `p0_sp` = 101325._sp
- Standard pressure [Pa] in single precision.*
- real(dp), parameter `rho0_dp` = 1.225_dp
- standard density [kg m⁻³] in double precision*
- real(sp), parameter `rho0_sp` = 1.225_sp
- standard density [kg m⁻³] in single precision*
- real(dp), parameter `cp0_dp` = 1005.0_dp
- specific heat capacity of air [J kg⁻¹ K⁻¹] in double precision*
- real(sp), parameter `cp0_sp` = 1005.0_sp
- specific heat capacity of air [J kg⁻¹ K⁻¹] in single precision*
- real(dp), parameter `pi_d` = 3.141592653589793238462643383279502884197_dp
- Pi in double precision.*
- real(sp), parameter `pi` = 3.141592653589793238462643383279502884197_sp
- Pi in single precision.*
- real(dp), parameter `pio2_d` = 1.57079632679489661923132169163975144209858_dp
- Pi/2 in double precision.*

- real(sp), parameter `pio2` = 1.57079632679489661923132169163975144209858_sp
Pi/2 in single precision.
- real(dp), parameter `twopi_d` = 6.283185307179586476925286766559005768394_dp
*2*Pi in double precision*
- real(sp), parameter `twopi` = 6.283185307179586476925286766559005768394_sp
*2*Pi in single precision*
- real(dp), parameter `sqrt2_d` = 1.41421356237309504880168872420969807856967_dp
Square root of 2 in double precision.
- real(sp), parameter `sqrt2` = 1.41421356237309504880168872420969807856967_sp
Square root of 2 in single precision.
- real(dp), parameter `euler_d` = 0.5772156649015328606065120900824024310422_dp
Euler's constant in double precision.
- real(sp), parameter `euler` = 0.5772156649015328606065120900824024310422_sp
Euler's constant in single precision.
- integer, parameter `nin` = input_unit
Standard input file unit.
- integer, parameter `nout` = output_unit
Standard output file unit.
- integer, parameter `nerr` = error_unit
Standard error file unit.
- integer, parameter `nnml` = 100
Standard file unit for namelist.

15.6.1 Detailed Description

Provides computational, mathematical, physical, and file constants.

Provides computational constants like epsilon, mathematical constants such as Pi, physical constants such as the Stefan-Boltzmann constant, and file units for some standard streams such as standard in.

Author

Matthias Cuntz

Date

Nov 2011

15.6.2 Variable Documentation

15.6.2.1 cp0_dp

`real(dp), parameter mo_constants::cp0_dp = 1005.0_dp`

specific heat capacity of air [J kg⁻¹ K⁻¹] in double precision

Referenced by `mo_pet::pet_penman()`.

15.6.2.2 cp0_sp

```
real(sp), parameter mo_constants::cp0_sp = 1005.0_sp
```

specific heat capacity of air [$\text{J kg}^{-1} \text{K}^{-1}$] in single precision

15.6.2.3 deg2rad_dp

```
real(dp), parameter mo_constants::deg2rad_dp = PI_dp/180._dp
```

degree to radian conversion ($\pi/180$) in double precision

Referenced by mo_pet::pet_hargreaves().

15.6.2.4 deg2rad_sp

```
real(sp), parameter mo_constants::deg2rad_sp = PI_sp/180._sp
```

degree to radian conversion ($\pi/180$) in double precision

15.6.2.5 eps_dp

```
real(dp), parameter mo_constants::eps_dp = epsilon(1.0_dp)
```

epsilon(1.0) in double precision

Referenced by mo_multi_param_reg::aerodynamical_resistance(), mo_mpr_pet::bulksurface_resistance(), mo_canopy_interc::canopy_interc(), mo_temporal_aggregation::day2mon_average_dp(), mo_objective_function::extract_basin_avg_tws(), mo_startup::l0_check_input(), mo_objective_function::objective_kge_q_rmse_et(), mo_objective_function::objective_kge_q_rmse_tws(), mo_mpr_pet::priestley_taylor_alpha(), mo_read_config::read_config(), mo_soil_database::read_soil_lut(), mo_runoff::runoff_unsat_zone(), and mo_soil_moisture::soil_moisture().

15.6.2.6 eps_sp

```
real(sp), parameter mo_constants::eps_sp = epsilon(1.0_sp)
```

epsilon(1.0) in single precision

15.6.2.7 euler

```
real(sp), parameter mo_constants::euler = 0.5772156649015328606065120900824024310422_sp
```

Euler's constant in single precision.

15.6.2.8 euler_d

```
real(dp), parameter mo_constants::euler_d = 0.5772156649015328606065120900824024310422_dp
```

Euler's constant in double precision.

15.6.2.9 gravity_dp

```
real(dp), parameter mo_constants::gravity_dp = 9.81_dp
```

Gravity acceleration [$\text{m}^2 \text{s}^{-1}$] in double precision.

15.6.2.10 gravity_sp

```
real(sp), parameter mo_constants::gravity_sp = 9.81_sp
```

Gravity acceleration [$\text{m}^2 \text{s}^{-1}$] in single precision.

15.6.2.11 nerr

```
integer, parameter mo_constants::nerr = error_unit
```

Standard error file unit.

15.6.2.12 nin

```
integer, parameter mo_constants::nin = input_unit
```

Standard input file unit.

15.6.2.13 nnml

```
integer, parameter mo_constants::nnml = 100
```

Standard file unit for namelist.

15.6.2.14 nout

```
integer, parameter mo_constants::nout = output_unit
```

Standard output file unit.

Referenced by `mo_message::message()`.

15.6.2.15 p0_dp

```
real(dp), parameter mo_constants::p0_dp = 101325._dp
```

standard atmosphere Standard pressure [Pa] in double precision

15.6.2.16 p0_sp

```
real(sp), parameter mo_constants::p0_sp = 101325._sp
```

Standard pressure [Pa] in single precision.

15.6.2.17 pi

```
real(sp), parameter mo_constants::pi = 3.141592653589793238462643383279502884197_sp
```

Pi in single precision.

15.6.2.18 pi_d

```
real(dp), parameter mo_constants::pi_d = 3.141592653589793238462643383279502884197_dp
```

Pi in double precision.

Referenced by mo_pet::extraterr_rad_approx().

15.6.2.19 pi_dp

```
real(dp), parameter mo_constants::pi_dp = 3.141592653589793238462643383279502884197_dp
```

Pi in double precision.

Referenced by mo_neutrons::cosmic(), mo_corr::fourrow_dp(), mo_corr::fourrow_sp(), mo_mrm_objective_↵
function_runoff::loglikelihood_evin2013_2(), mo_neutrons::lookupintegral(), mo_neutrons::oldintegration(), mo_↵
mrm_objective_function_runoff::parameter_regularization(), and mo_neutrons::tabularintegralafast().

15.6.2.20 pi_sp

```
real(sp), parameter mo_constants::pi_sp = 3.141592653589793238462643383279502884197_sp
```

Pi in single precision.

15.6.2.21 pio2

```
real(sp), parameter mo_constants::pio2 = 1.57079632679489661923132169163975144209858_sp
```

Pi/2 in single precision.

15.6.2.22 pio2_d

```
real(dp), parameter mo_constants::pio2_d = 1.57079632679489661923132169163975144209858_dp
```

Pi/2 in double precision.

15.6.2.23 pio2_dp

```
real(dp), parameter mo_constants::pio2_dp = 1.57079632679489661923132169163975144209858_dp
```

Pi/2 in double precision.

15.6.2.24 pio2_sp

```
real(sp), parameter mo_constants::pio2_sp = 1.57079632679489661923132169163975144209858_sp
```

Pi/2 in single precision.

15.6.2.25 psychro_dp

```
real(dp), parameter mo_constants::psychro_dp = 0.0646_dp
```

Psychrometric constant [kPa K⁻¹] in double precision.

Referenced by mo_pet::pet_penman(), and mo_pet::pet_priestly().

15.6.2.26 psychro_sp

```
real(sp), parameter mo_constants::psychro_sp = 0.0646_sp
```

Psychrometric constant [kPa K⁻¹] in single precision.

15.6.2.27 rad2deg_dp

```
real(dp), parameter mo_constants::rad2deg_dp = 180._dp/PI_dp
```

radian to conversion (180/pi) in double precision

15.6.2.28 rad2deg_sp

```
real(sp), parameter mo_constants::rad2deg_sp = 180._sp/PI_sp
```

radian to degree conversion (180/pi) in single precision

15.6.2.29 radiusearth_dp

```
real(dp), parameter mo_constants::radiusearth_dp = 6371228._dp
```

Referenced by mo_mrm_net_startup::get_distance_two_lat_lon_points(), mo_startup::l0_variable_init(), and mo←
_mrm_read_data::mrm_l0_variable_init().

15.6.2.30 radiusearth_sp

```
real(sp), parameter mo_constants::radiusearth_sp = 6371228._sp
```

15.6.2.31 rho0_dp

```
real(dp), parameter mo_constants::rho0_dp = 1.225_dp
```

standard density [kg m^{-3}] in double precision

Referenced by mo_pet::pet_penman().

15.6.2.32 rho0_sp

```
real(sp), parameter mo_constants::rho0_sp = 1.225_sp
```

standard density [kg m^{-3}] in single precision

15.6.2.33 secday_dp

```
real(dp), parameter mo_constants::secday_dp = 86400._dp
```

Seconds per day [s] in double precision.

15.6.2.34 secday_sp

```
real(sp), parameter mo_constants::secday_sp = 86400._sp
```

Seconds per day [s] in single precision.

15.6.2.35 sigma_dp

```
real(dp), parameter mo_constants::sigma_dp = 5.67e-08_dp
```

Stefan-Boltzmann constant [$\text{W m}^{-2} \text{K}^{-4}$] in double precision.

15.6.2.36 sigma_sp

```
real(sp), parameter mo_constants::sigma_sp = 5.67e-08_sp
```

Stefan-Boltzmann constant [$\text{W m}^{-2} \text{K}^{-4}$] in single precision.

15.6.2.37 solarconst_dp

```
real(dp), parameter mo_constants::solarconst_dp = 1367._dp
```

Solar constant in [$\text{J m}^{-2} \text{s}^{-1}$] in double precision.

Referenced by `mo_pet::extraterr_rad_approx()`.

15.6.2.38 `solarconst_sp`

```
real(sp), parameter mo_constants::solarconst_sp = 1367._sp
```

Solar constant in $[\text{J m}^{-2} \text{s}^{-1}]$ in single precision.

15.6.2.39 `specheatet_dp`

```
real(dp), parameter mo_constants::specheatet_dp = 2.45e06_dp
```

Specific heat for vaporization of water in $[\text{J m}^{-2} \text{mm}^{-1}]$ in double precision.

Referenced by `mo_pet::extraterr_rad_approx()`, `mo_pet::pet_penman()`, and `mo_pet::pet_priestly()`.

15.6.2.40 `specheatet_sp`

```
real(sp), parameter mo_constants::specheatet_sp = 2.45e06_sp
```

Specific heat for vaporization of water in $[\text{J m}^{-2} \text{mm}^{-1}]$ in single precision.

15.6.2.41 `sqrt2`

```
real(sp), parameter mo_constants::sqrt2 = 1.41421356237309504880168872420969807856967_sp
```

Square root of 2 in single precision.

15.6.2.42 `sqrt2_d`

```
real(dp), parameter mo_constants::sqrt2_d = 1.41421356237309504880168872420969807856967_dp
```

Square root of 2 in double precision.

15.6.2.43 `sqrt2_dp`

```
real(dp), parameter mo_constants::sqrt2_dp = 1.41421356237309504880168872420969807856967_dp
```

Square root of 2 in double precision.

Referenced by `mo_mrm_net_startup::celllength()`.

15.6.2.44 `sqrt2_sp`

```
real(sp), parameter mo_constants::sqrt2_sp = 1.41421356237309504880168872420969807856967_sp
```

Square root of 2 in single precision.

Variables

- integer(i4), parameter `npar_arth` =16
- integer(i4), parameter `npar2_arth` =8

15.7.1 Function/Subroutine Documentation

15.7.1.1 arth_dp()

```
real(dp) function, dimension(n) mo_corr::arth_dp (
    real(dp), intent(in) first,
    real(dp), intent(in) increment,
    integer(i4), intent(in) n ) [private]
```

References `npar2_arth`, and `npar_arth`.

15.7.1.2 arth_i4()

```
integer(i4) function, dimension(n) mo_corr::arth_i4 (
    integer(i4), intent(in) first,
    integer(i4), intent(in) increment,
    integer(i4), intent(in) n ) [private]
```

References `npar2_arth`, and `npar_arth`.

15.7.1.3 arth_sp()

```
real(sp) function, dimension(n) mo_corr::arth_sp (
    real(sp), intent(in) first,
    real(sp), intent(in) increment,
    integer(i4), intent(in) n ) [private]
```

References `npar2_arth`, and `npar_arth`.

15.7.1.4 autocoeffk_1d_dp()

```
real(dp) function, dimension(size(k)) mo_corr::autocoeffk_1d_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.5 autocoeffk_1d_sp()

```
real(sp) function, dimension(size(k)) mo_corr::autocoeffk_1d_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.6 autocoeffk_dp()

```
real(dp) function mo_corr::autocoeffk_dp (  
    real(dp), dimension(:), intent(in) x,  
    integer(i4), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.7 autocoeffk_sp()

```
real(sp) function mo_corr::autocoeffk_sp (  
    real(sp), dimension(:), intent(in) x,  
    integer(i4), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.8 autocorr_1d_dp()

```
real(dp) function, dimension(size(k)) mo_corr::autocorr_1d_dp (  
    real(dp), dimension(:), intent(in) x,  
    integer(i4), dimension(:), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.9 autocorr_1d_sp()

```
real(sp) function, dimension(size(k)) mo_corr::autocorr_1d_sp (  
    real(sp), dimension(:), intent(in) x,  
    integer(i4), dimension(:), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.10 autocorr_dp()

```
real(dp) function mo_corr::autocorr_dp (  
    real(dp), dimension(:), intent(in) x,  
    integer(i4), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.11 autocorr_sp()

```
real(sp) function mo_corr::autocorr_sp (  
    real(sp), dimension(:), intent(in) x,  
    integer(i4), intent(in) k,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.12 corr_dp()

```
real(dp) function, dimension(size(data1)) mo_corr::corr_dp (
    real(dp), dimension(:), intent(in) data1,
    real(dp), dimension(:), intent(in) data2,
    integer(i4), intent(out), optional nadjust,
    integer(i4), intent(in), optional nhigh,
    integer(i4), intent(in), optional nwin ) [private]
```

15.7.1.13 corr_sp()

```
real(sp) function, dimension(size(data1)) mo_corr::corr_sp (
    real(sp), dimension(:), intent(in) data1,
    real(sp), dimension(:), intent(in) data2,
    integer(i4), intent(out), optional nadjust,
    integer(i4), intent(in), optional nhigh,
    integer(i4), intent(in), optional nwin ) [private]
```

15.7.1.14 crosscoeffk_dp()

```
real(dp) function mo_corr::crosscoeffk_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.15 crosscoeffk_sp()

```
real(sp) function mo_corr::crosscoeffk_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.16 crosscorr_dp()

```
real(dp) function mo_corr::crosscorr_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.17 crosscorr_sp()

```
real(sp) function mo_corr::crosscorr_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
```

```
integer(i4), intent(in) k,
logical, dimension(:), intent(in), optional mask ) [private]
```

15.7.1.18 four1_dp()

```
subroutine mo_corr::four1_dp (
    complex(dpc), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

References mo_kind::dpc, and mo_constants::twopi_dp.

15.7.1.19 four1_sp()

```
subroutine mo_corr::four1_sp (
    complex(spc), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

References mo_kind::dpc, mo_kind::spc, and mo_constants::twopi_dp.

15.7.1.20 fourrow_dp()

```
subroutine mo_corr::fourrow_dp (
    complex(dpc), dimension(:,:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

References mo_kind::dpc, and mo_constants::pi_dp.

15.7.1.21 fourrow_sp()

```
subroutine mo_corr::fourrow_sp (
    complex(spc), dimension(:,:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

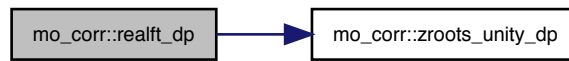
References mo_kind::dpc, mo_constants::pi_dp, and mo_kind::spc.

15.7.1.22 realft_dp()

```
subroutine mo_corr::realft_dp (
    real(dp), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign,
    complex(dpc), dimension(:), optional, target zdata ) [private]
```

References mo_kind::dpc, and roots_unity_dp().

Here is the call graph for this function:



15.7.1.23 realft_sp()

```

subroutine mo_corr::realft_sp (
    real(sp), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign,
    complex(spc), dimension(:), optional, target zdata ) [private]
  
```

References mo_kind::spc, and zroots_unity_sp().

Here is the call graph for this function:



15.7.1.24 swap_1d_dpc()

```

subroutine mo_corr::swap_1d_dpc (
    complex(dpc), dimension(:), intent(inout) a,
    complex(dpc), dimension(:), intent(inout) b ) [private]
  
```

15.7.1.25 swap_1d_spc()

```

subroutine mo_corr::swap_1d_spc (
    complex(spc), dimension(:), intent(inout) a,
    complex(spc), dimension(:), intent(inout) b ) [private]
  
```

15.7.1.26 zroots_unity_dp()

```

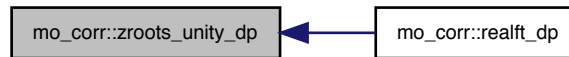
complex(dpc) function, dimension(nn) mo_corr::zroots_unity_dp (
  
```

```
integer(i4), intent(in) n,
integer(i4), intent(in) nn ) [private]
```

References `mo_kind::dpc`, and `mo_constants::twopi_dp`.

Referenced by `realft_dp()`.

Here is the caller graph for this function:



15.7.1.27 `zroots_unity_sp()`

```
complex(spc) function, dimension(nn) mo_corr::zroots_unity_sp (
    integer(i4), intent(in) n,
    integer(i4), intent(in) nn ) [private]
```

References `mo_kind::spc`, and `mo_constants::twopi_sp`.

Referenced by `realft_sp()`.

Here is the caller graph for this function:



15.7.2 Variable Documentation

15.7.2.1 `npar2_arth`

```
integer(i4), parameter mo_corr::npar2_arth =8 [private]
```

Referenced by `arth_dp()`, `arth_i4()`, and `arth_sp()`.

15.7.2.2 `npar_arth`

```
integer(i4), parameter mo_corr::npar_arth =16 [private]
```

Referenced by `arth_dp()`, `arth_i4()`, and `arth_sp()`.

15.8 mo_dds Module Reference

Dynamically Dimensioned Search (DDS)

Functions/Subroutines

- `real(dp)` function, `dimension(size(pini))`, public `dds` (`obj_func`, `pini`, `prange`, `r`, `seed`, `maxiter`, `maxit`, `mask`, `tmp_file`, `funcbest`, `history`)
DDS.
- `real(dp)` function, `dimension(size(pini))`, public `mdds` (`obj_func`, `pini`, `prange`, `seed`, `maxiter`, `maxit`, `mask`, `tmp_file`, `funcbest`, `history`)
MDDS.
- subroutine `neigh_value` (`x_cur`, `x_min`, `x_max`, `r`, `new_value`)

15.8.1 Detailed Description

Dynamically Dimensioned Search (DDS)

This module provides routines for Dynamically Dimensioned Search (DDS) of Tolson and Shoemaker (2007). It searches the minimum or maximum of a user-specified function, using an n-dimensional continuous global optimization algorithm (DDS).

Authors

Original by Bryan Tolson and later modified by Rohini Kumar. Matthias Cuntz and Juliane Mai for the module, MDDS, etc.

Date

Jul 2012

15.8.2 Function/Subroutine Documentation

15.8.2.1 dds()

```
real(dp) function, dimension(size(pini)), public mo_dds::dds (
    obj_func,
    real(dp), dimension(:), intent(in) pini,
    real(dp), dimension(:,:), intent(in) prange,
    real(dp), intent(in), optional r,
    integer(i8), intent(in), optional seed,
    integer(i8), intent(in), optional maxiter,
    logical, intent(in), optional maxit,
    logical, dimension(:), intent(in), optional mask,
    character(len=*), intent(in), optional tmp_file,
    real(dp), intent(out), optional funcbest,
    real(dp), dimension(:), intent(out), optional, allocatable history )
```

DDS.

Searches Minimum or Maximum of a user-specified function using Dynamically Dimensioned Search (DDS).

DDS is an n-dimensional continuous global optimization algorithm. It is coded as a minimizer but one can give `maxit=True` in a maximization problem, so that the algorithm minimizes the negative of the objective function $F=(-1 * F)$. The function to be minimized is the first argument of DDS and must be defined as

```
function func(p)
  use mo_kind, only: dp
  implicit none
  real(dp), dimension(:), intent(in) :: p
  real(dp) :: func
end function func
```

Parameters

in	<i>real(dp) :: obj_func(p)</i>	Function on which to search the minimum
in	<i>real(dp) :: pini(:)</i>	inital value of decision variables
in	<i>real(dp) :: prange(size(pini),2)</i>	Min/max range of decision variables
in	<i>real(dp), optional :: r</i>	DDS perturbation parameter (default: 0.2)
in	<i>integer(i8), optional :: seed</i>	User seed to initialise the random number generator (default: None)
in	<i>integer(i8), optional :: maxiter</i>	Maximum number of iteration or function evaluation (default: 1000)
in	<i>logical, optional :: maxit</i>	Maximization (.True.) or minimization (.False.) of function (default: .False.)
in	<i>logical, optional :: mask(size(pini))</i>	parameter to be optimized (true or false) (default: .True.)
in	<i>character(len=*) , optional :: tmp_file</i>	file with temporal output
out	<i>real(dp), optional :: funcbest</i>	the best value of the function.
out	<i>real(dp), optional, allocatable :: history(:)</i>	the history of best function values, history(maxiter)=funcbest allocatable only to be in correspondance with other optimization routines

Returns

`real(dp) :: DDS` — The parameters of the point which is estimated to minimize the function.

Author

Written original Bryan Tolson - DDS v1.1
Modified Rohini Kumar, Matthias Cuntz, Juliane Mai

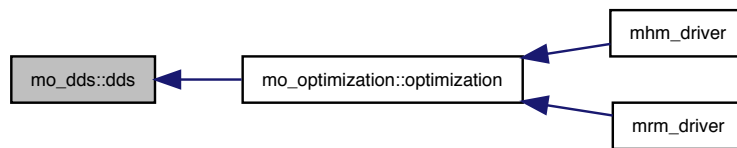
References `mo_kind::dp`, `mo_kind::i4`, `mo_kind::i8`, and `neigh_value()`.

Referenced by `mo_optimization::optimization()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.8.2.2 mdds()

```

real(dp) function, dimension(size(pini)), public mo_dds::mdds (
    obj_func,
    real(dp), dimension(:), intent(in) pini,
    real(dp), dimension(:, :), intent(in) prange,
    integer(i8), intent(in), optional seed,
    integer(i8), intent(in), optional maxiter,
    logical, intent(in), optional maxit,
    logical, dimension(:), intent(in), optional mask,
    character(len=*), intent(in), optional tmp_file,
    real(dp), intent(out), optional funcbest,
    real(dp), dimension(:), intent(out), optional, allocatable history )
  
```

MDDS.

Searches Minimum or Maximum of a user-specified function using the Modified Dynamically Dimensioned Search (DDS). DDS is an n-dimensional continuous global optimization algorithm. It is coded as a minimizer but one can give `maxit=True` in a maximization problem, so that the algorithm minimizes the negative of the objective function $F=(-1 * F)$.

The function to be minimized is the first argument of DDS and must be defined as

```

function func(p)
  use mo_kind, only: dp
  implicit none
  real(dp), dimension(:), intent(in) :: p
  real(dp) :: func
end function func
  
```

MDDS extents normal DDS by a continuous reduction of the DDS perturbation parameter r from 0.3 to 0.05, and by allowing a larger function value with a certain probability.

Parameters

in	<i>real(dp) :: obj_func(p)</i>	Function on which to search the minimum
in	<i>real(dp) :: pini(:)</i>	inital value of decision variables
in	<i>real(dp) :: prange(size(pini),2)</i>	Min/max range of decision variables
in	<i>integer(i8), optional :: seed</i>	User seed to initialise the random number generator (default: None)
in	<i>integer(i8), optional :: maxiter</i>	Maximum number of iteration or function evaluation (default: 1000)

Parameters

in	<i>logical, optional :: maxit</i>	Maximization (.True.) or minimization (.False.) of function (default: .False.)
in	<i>logical, optional :: mask(size(pini))</i>	parameter to be optimized (true or false) (default: .True.)
in	<i>character(len=*) , optional :: tmp_file</i>	file with temporal output
out	<i>real(dp), optional :: funcbest</i>	the best value of the function.
out	<i>real(dp), optional, allocatable :: history(:)</i>	the history of best function values,

Returns

`real(dp) :: MDDS` — The parameters of the point which is estimated to minimize the function.

Author

Written Matthias Cuntz and Juliane Mai

References `mo_kind::dp`, `mo_kind::i4`, `mo_kind::i8`, and `neigh_value()`.

Here is the call graph for this function:

**15.8.2.3 neigh_value()**

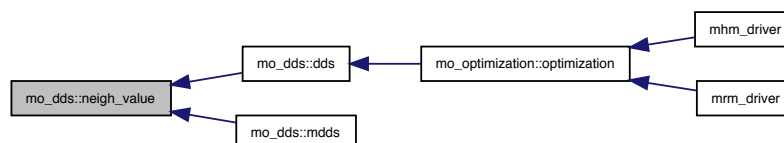
```

subroutine mo_dds::neigh_value (
    real(dp), intent(in) x_cur,
    real(dp), intent(in) x_min,
    real(dp), intent(in) x_max,
    real(dp), intent(in) r,
    real(dp), intent(out) new_value )
  
```

References `mo_kind::dp`, and `mo_kind::i8`.

Referenced by `dds()`, and `mdds()`.

Here is the caller graph for this function:



15.9 mo_errormeasures Module Reference

Data Types

- interface [bias](#)
- interface [kge](#)
 - Kling-Gupta-Efficiency measure.*
- interface [kgenocorr](#)
 - Kling-Gupta-Efficiency measure without correlation.*
- interface [lnnse](#)
- interface [mae](#)
- interface [mse](#)
- interface [nse](#)
- interface [rmse](#)
- interface [sae](#)
- interface [sse](#)

Functions/Subroutines

- real(sp) function [bias_sp_1d](#) (x, y, mask)
- real(dp) function [bias_dp_1d](#) (x, y, mask)
- real(sp) function [bias_sp_2d](#) (x, y, mask)
- real(dp) function [bias_dp_2d](#) (x, y, mask)
- real(sp) function [bias_sp_3d](#) (x, y, mask)
- real(dp) function [bias_dp_3d](#) (x, y, mask)
- real(sp) function [kge_sp_1d](#) (x, y, mask)
- real(sp) function [kge_sp_2d](#) (x, y, mask)
- real(sp) function [kge_sp_3d](#) (x, y, mask)
- real(dp) function [kge_dp_1d](#) (x, y, mask)
- real(dp) function [kge_dp_2d](#) (x, y, mask)
- real(dp) function [kge_dp_3d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_1d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_2d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_3d](#) (x, y, mask)
- real(dp) function [kgenocorr_dp_1d](#) (x, y, mask)
- real(dp) function [kgenocorr_dp_2d](#) (x, y, mask)
- real(dp) function [kgenocorr_dp_3d](#) (x, y, mask)
- real(sp) function [lnnse_sp_1d](#) (x, y, mask)
- real(dp) function [lnnse_dp_1d](#) (x, y, mask)
- real(sp) function [lnnse_sp_2d](#) (x, y, mask)
- real(dp) function [lnnse_dp_2d](#) (x, y, mask)
- real(sp) function [lnnse_sp_3d](#) (x, y, mask)
- real(dp) function [lnnse_dp_3d](#) (x, y, mask)
- real(sp) function [mae_sp_1d](#) (x, y, mask)
- real(dp) function [mae_dp_1d](#) (x, y, mask)
- real(sp) function [mae_sp_2d](#) (x, y, mask)
- real(dp) function [mae_dp_2d](#) (x, y, mask)
- real(sp) function [mae_sp_3d](#) (x, y, mask)
- real(dp) function [mae_dp_3d](#) (x, y, mask)
- real(sp) function [mse_sp_1d](#) (x, y, mask)
- real(dp) function [mse_dp_1d](#) (x, y, mask)
- real(sp) function [mse_sp_2d](#) (x, y, mask)
- real(dp) function [mse_dp_2d](#) (x, y, mask)
- real(sp) function [mse_sp_3d](#) (x, y, mask)

- real(dp) function [mse_dp_3d](#) (x, y, mask)
- real(sp) function [mse_sp_1d](#) (x, y, mask)
- real(dp) function [mse_dp_1d](#) (x, y, mask)
- real(sp) function [mse_sp_2d](#) (x, y, mask)
- real(dp) function [mse_dp_2d](#) (x, y, mask)
- real(sp) function [mse_sp_3d](#) (x, y, mask)
- real(dp) function [mse_dp_3d](#) (x, y, mask)
- real(sp) function [sae_sp_1d](#) (x, y, mask)
- real(dp) function [sae_dp_1d](#) (x, y, mask)
- real(sp) function [sae_sp_2d](#) (x, y, mask)
- real(dp) function [sae_dp_2d](#) (x, y, mask)
- real(sp) function [sae_sp_3d](#) (x, y, mask)
- real(dp) function [sae_dp_3d](#) (x, y, mask)
- real(sp) function [sse_sp_1d](#) (x, y, mask)
- real(dp) function [sse_dp_1d](#) (x, y, mask)
- real(sp) function [sse_sp_2d](#) (x, y, mask)
- real(dp) function [sse_dp_2d](#) (x, y, mask)
- real(sp) function [sse_sp_3d](#) (x, y, mask)
- real(dp) function [sse_dp_3d](#) (x, y, mask)
- real(sp) function [rmse_sp_1d](#) (x, y, mask)
- real(dp) function [rmse_dp_1d](#) (x, y, mask)
- real(sp) function [rmse_sp_2d](#) (x, y, mask)
- real(dp) function [rmse_dp_2d](#) (x, y, mask)
- real(sp) function [rmse_sp_3d](#) (x, y, mask)
- real(dp) function [rmse_dp_3d](#) (x, y, mask)

15.9.1 Function/Subroutine Documentation

15.9.1.1 [bias_dp_1d\(\)](#)

```
real(dp) function mo_errormeasures::bias_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

15.9.1.2 [bias_dp_2d\(\)](#)

```
real(dp) function mo_errormeasures::bias_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )
```

15.9.1.3 [bias_dp_3d\(\)](#)

```
real(dp) function mo_errormeasures::bias_dp_3d (
    real(dp), dimension(:, :, :), intent(in) x,
    real(dp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask )
```

15.9.1.4 bias_sp_1d()

```
real(sp) function mo_errormeasures::bias_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.9.1.5 bias_sp_2d()

```
real(sp) function mo_errormeasures::bias_sp_2d (  
    real(sp), dimension(:, :), intent(in) x,  
    real(sp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

15.9.1.6 bias_sp_3d()

```
real(sp) function mo_errormeasures::bias_sp_3d (  
    real(sp), dimension(:, :, :), intent(in) x,  
    real(sp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

15.9.1.7 kge_dp_1d()

```
real(dp) function mo_errormeasures::kge_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

15.9.1.8 kge_dp_2d()

```
real(dp) function mo_errormeasures::kge_dp_2d (  
    real(dp), dimension(:, :), intent(in) x,  
    real(dp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

15.9.1.9 kge_dp_3d()

```
real(dp) function mo_errormeasures::kge_dp_3d (  
    real(dp), dimension(:, :, :), intent(in) x,  
    real(dp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

15.9.1.10 kge_sp_1d()

```
real(sp) function mo_errormeasures::kge_sp_1d (  

```

```

real(sp), dimension(:), intent(in) x,
real(sp), dimension(:), intent(in) y,
logical, dimension(:), intent(in), optional mask )

```

15.9.1.11 kge_sp_2d()

```

real(sp) function mo_errormeasures::kge_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )

```

15.9.1.12 kge_sp_3d()

```

real(sp) function mo_errormeasures::kge_sp_3d (
    real(sp), dimension(:, :, :), intent(in) x,
    real(sp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask )

```

15.9.1.13 kgenocorr_dp_1d()

```

real(dp) function mo_errormeasures::kgenocorr_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )

```

15.9.1.14 kgenocorr_dp_2d()

```

real(dp) function mo_errormeasures::kgenocorr_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )

```

15.9.1.15 kgenocorr_dp_3d()

```

real(dp) function mo_errormeasures::kgenocorr_dp_3d (
    real(dp), dimension(:, :, :), intent(in) x,
    real(dp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask )

```

15.9.1.16 kgenocorr_sp_1d()

```

real(sp) function mo_errormeasures::kgenocorr_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )

```

15.9.1.17 kgenocorr_sp_2d()

```
real(sp) function mo_errormeasures::kgenocorr_sp_2d (  
    real(sp), dimension(:, :), intent(in) x,  
    real(sp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

15.9.1.18 kgenocorr_sp_3d()

```
real(sp) function mo_errormeasures::kgenocorr_sp_3d (  
    real(sp), dimension(:, :, :), intent(in) x,  
    real(sp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

15.9.1.19 lnnse_dp_1d()

```
real(dp) function mo_errormeasures::lnnse_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(inout), optional mask )
```

15.9.1.20 lnnse_dp_2d()

```
real(dp) function mo_errormeasures::lnnse_dp_2d (  
    real(dp), dimension(:, :), intent(in) x,  
    real(dp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(inout), optional mask )
```

15.9.1.21 lnnse_dp_3d()

```
real(dp) function mo_errormeasures::lnnse_dp_3d (  
    real(dp), dimension(:, :, :), intent(in) x,  
    real(dp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(inout), optional mask )
```

15.9.1.22 lnnse_sp_1d()

```
real(sp) function mo_errormeasures::lnnse_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(inout), optional mask )
```

15.9.1.23 lnnse_sp_2d()

```
real(sp) function mo_errormeasures::lnnse_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(inout), optional mask )
```

15.9.1.24 lnnse_sp_3d()

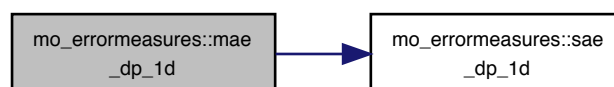
```
real(sp) function mo_errormeasures::lnnse_sp_3d (
    real(sp), dimension(:, :, :), intent(in) x,
    real(sp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(inout), optional mask )
```

15.9.1.25 mae_dp_1d()

```
real(dp) function mo_errormeasures::mae_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
```

References sae_dp_1d().

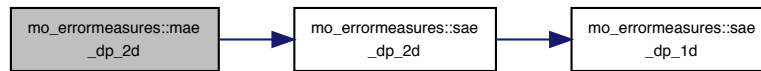
Here is the call graph for this function:

**15.9.1.26 mae_dp_2d()**

```
real(dp) function mo_errormeasures::mae_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
```

References sae_dp_2d().

Here is the call graph for this function:



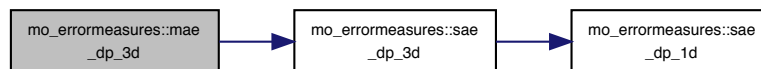
15.9.1.27 mae_dp_3d()

```

real(dp) function mo_errormeasures::mae_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

References `sae_dp_3d()`.

Here is the call graph for this function:



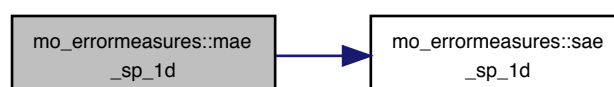
15.9.1.28 mae_sp_1d()

```

real(sp) function mo_errormeasures::mae_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
  
```

References `sae_sp_1d()`.

Here is the call graph for this function:

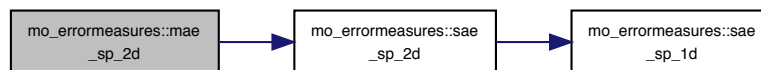


15.9.1.29 mae_sp_2d()

```
real(sp) function mo_errormeasures::mae_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask ) [private]
```

References sae_sp_2d().

Here is the call graph for this function:

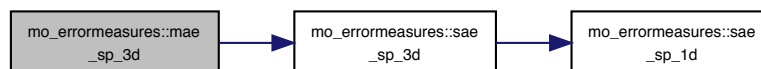


15.9.1.30 mae_sp_3d()

```
real(sp) function mo_errormeasures::mae_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
```

References sae_sp_3d().

Here is the call graph for this function:



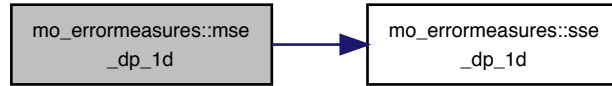
15.9.1.31 mse_dp_1d()

```
real(dp) function mo_errormeasures::mse_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
```

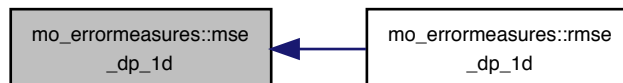
References sse_dp_1d().

Referenced by rmse_dp_1d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.32 mse_dp_2d()

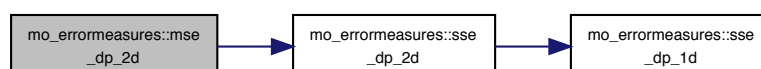
```

real(dp) function mo_errormeasures::mse_dp_2d (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask ) [private]
  
```

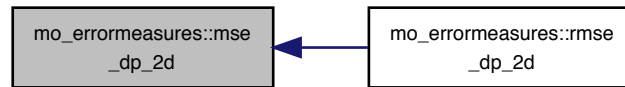
References `sse_dp_2d()`.

Referenced by `rmse_dp_2d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.33 mse_dp_3d()

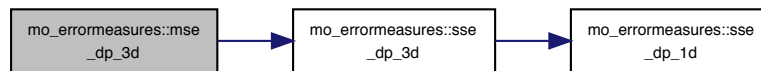
```

real(dp) function mo_errormeasures::mse_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

References `sse_dp_3d()`.

Referenced by `rmse_dp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.34 mse_sp_1d()

```

real(sp) function mo_errormeasures::mse_sp_1d (
    real(sp), dimension(:), intent(in) x,
  
```

```

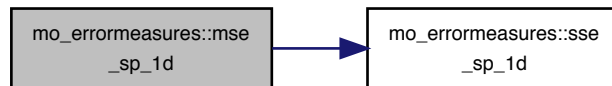
real(sp), dimension(:), intent(in) y,
logical, dimension(:), intent(in), optional mask ) [private]

```

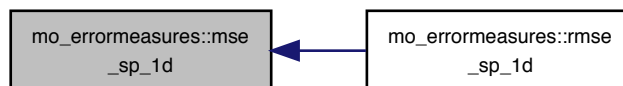
References sse_sp_1d().

Referenced by rmse_sp_1d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.35 mse_sp_2d()

```

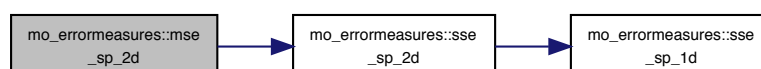
real(sp) function mo_errormeasures::mse_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]

```

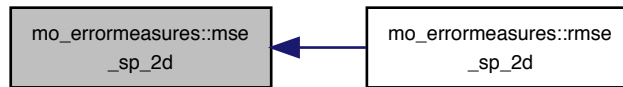
References sse_sp_2d().

Referenced by rmse_sp_2d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.36 mse_sp_3d()

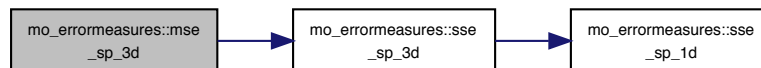
```

real(sp) function mo_errormeasures::mse_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

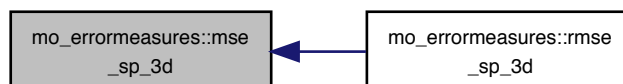
References `sse_sp_3d()`.

Referenced by `rmse_sp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.37 nse_dp_1d()

```

real(dp) function mo_errormeasures::nse_dp_1d (
    real(dp), dimension(:), intent(in) x,
  
```

```
real(dp), dimension(:), intent(in) y,  
logical, dimension(:), intent(in), optional mask )
```

15.9.1.38 nse_dp_2d()

```
real(dp) function mo_errormeasures::nse_dp_2d (  
    real(dp), dimension(:,:), intent(in) x,  
    real(dp), dimension(:,:), intent(in) y,  
    logical, dimension(:,:), intent(in), optional mask )
```

15.9.1.39 nse_dp_3d()

```
real(dp) function mo_errormeasures::nse_dp_3d (  
    real(dp), dimension(:,:,:), intent(in) x,  
    real(dp), dimension(:,:,:), intent(in) y,  
    logical, dimension(:,:,:), intent(in), optional mask )
```

15.9.1.40 nse_sp_1d()

```
real(sp) function mo_errormeasures::nse_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.9.1.41 nse_sp_2d()

```
real(sp) function mo_errormeasures::nse_sp_2d (  
    real(sp), dimension(:,:), intent(in) x,  
    real(sp), dimension(:,:), intent(in) y,  
    logical, dimension(:,:), intent(in), optional mask )
```

15.9.1.42 nse_sp_3d()

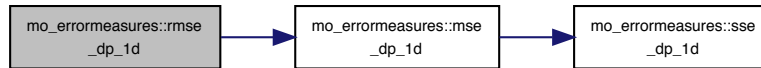
```
real(sp) function mo_errormeasures::nse_sp_3d (  
    real(sp), dimension(:,:,:), intent(in) x,  
    real(sp), dimension(:,:,:), intent(in) y,  
    logical, dimension(:,:,:), intent(in), optional mask )
```

15.9.1.43 rmse_dp_1d()

```
real(dp) function mo_errormeasures::rmse_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

References `mse_dp_1d()`.

Here is the call graph for this function:



15.9.1.44 `rmse_dp_2d()`

```

real(dp) function mo_errormeasures::rmse_dp_2d (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask ) [private]
  
```

References `mse_dp_2d()`.

Here is the call graph for this function:



15.9.1.45 `rmse_dp_3d()`

```

real(dp) function mo_errormeasures::rmse_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

References `mse_dp_3d()`.

Here is the call graph for this function:

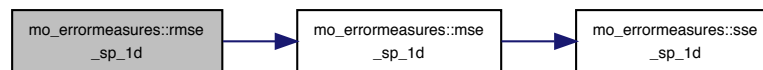


15.9.1.46 rmse_sp_1d()

```
real(sp) function mo_errormeasures::rmse_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
```

References mse_sp_1d().

Here is the call graph for this function:



15.9.1.47 rmse_sp_2d()

```
real(sp) function mo_errormeasures::rmse_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
```

References mse_sp_2d().

Here is the call graph for this function:

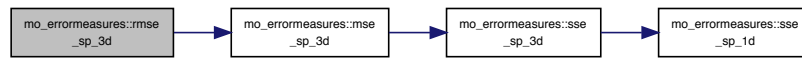


15.9.1.48 rmse_sp_3d()

```
real(sp) function mo_errormeasures::rmse_sp_3d (
    real(sp), dimension(:, :, :), intent(in) x,
    real(sp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask ) [private]
```

References mse_sp_3d().

Here is the call graph for this function:



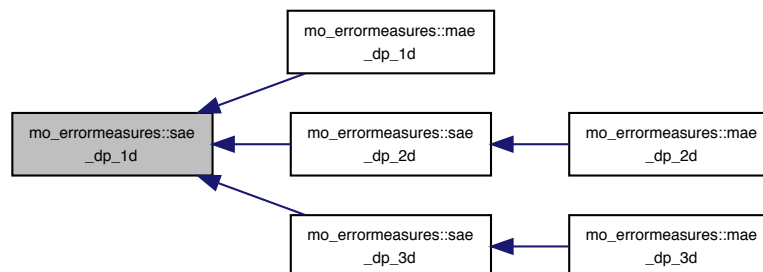
15.9.1.49 sae_dp_1d()

```

real(dp) function mo_errormeasures::sae_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
  
```

Referenced by mae_dp_1d(), sae_dp_2d(), and sae_dp_3d().

Here is the caller graph for this function:



15.9.1.50 sae_dp_2d()

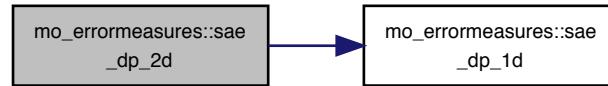
```

real(dp) function mo_errormeasures::sae_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
  
```

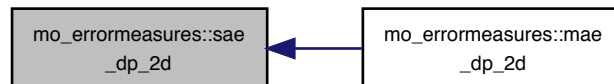
References sae_dp_1d().

Referenced by mae_dp_2d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.51 sae_dp_3d()

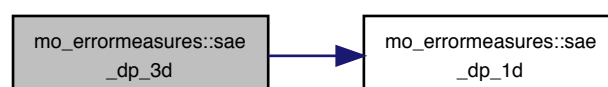
```

real(dp) function mo_errormeasures::sae_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

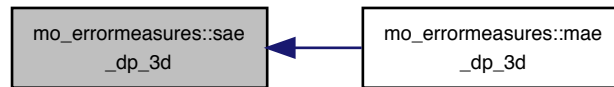
References `sae_dp_1d()`.

Referenced by `mae_dp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



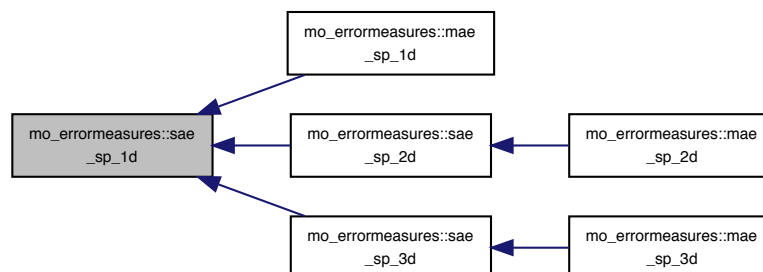
15.9.1.52 sae_sp_1d()

```

real(sp) function mo_errormeasures::sae_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
  
```

Referenced by `mae_sp_1d()`, `sae_sp_2d()`, and `sae_sp_3d()`.

Here is the caller graph for this function:



15.9.1.53 sae_sp_2d()

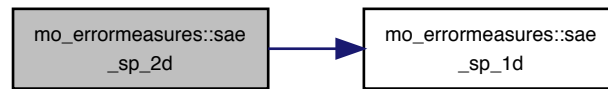
```

real(sp) function mo_errormeasures::sae_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
  
```

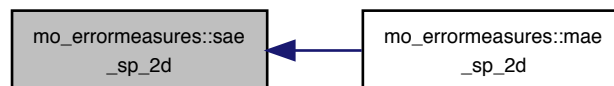
References `sae_sp_1d()`.

Referenced by `mae_sp_2d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.54 sae_sp_3d()

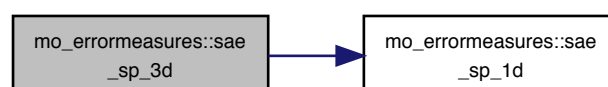
```

real(sp) function mo_errormeasures::sae_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

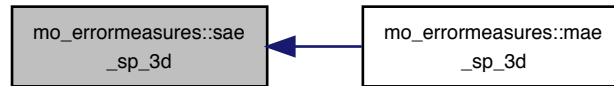
References `sae_sp_1d()`.

Referenced by `mae_sp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



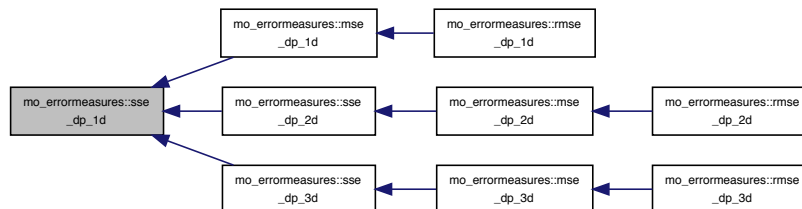
15.9.1.55 sse_dp_1d()

```

real(dp) function mo_errormeasures::sse_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
  
```

Referenced by mse_dp_1d(), sse_dp_2d(), and sse_dp_3d().

Here is the caller graph for this function:



15.9.1.56 sse_dp_2d()

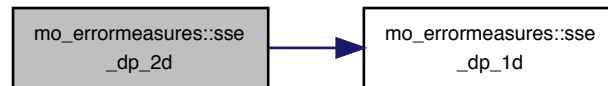
```

real(dp) function mo_errormeasures::sse_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
  
```

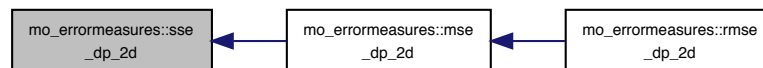
References sse_dp_1d().

Referenced by mse_dp_2d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.57 sse_dp_3d()

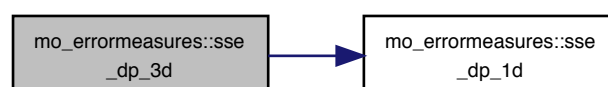
```

real(dp) function mo_errormeasures::sse_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

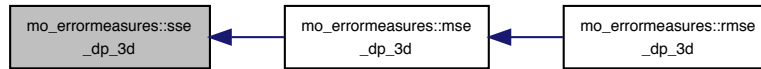
References `sse_dp_1d()`.

Referenced by `mse_dp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



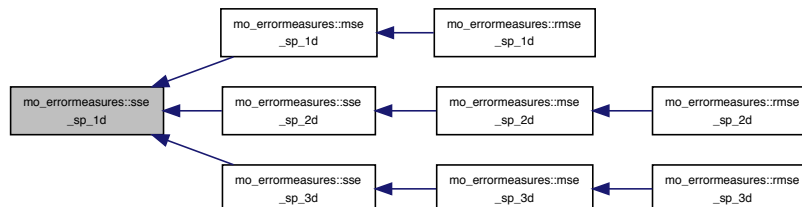
15.9.1.58 sse_sp_1d()

```

real(sp) function mo_errormeasures::sse_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask ) [private]
  
```

Referenced by mse_sp_1d(), sse_sp_2d(), and sse_sp_3d().

Here is the caller graph for this function:



15.9.1.59 sse_sp_2d()

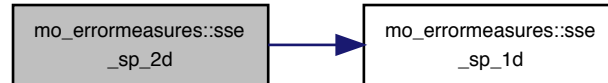
```

real(sp) function mo_errormeasures::sse_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask ) [private]
  
```

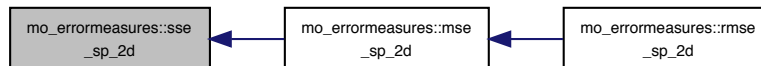
References sse_sp_1d().

Referenced by mse_sp_2d().

Here is the call graph for this function:



Here is the caller graph for this function:



15.9.1.60 sse_sp_3d()

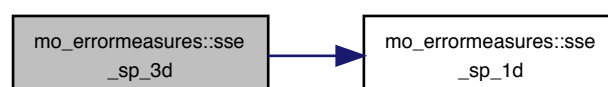
```

real(sp) function mo_errormeasures::sse_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask ) [private]
  
```

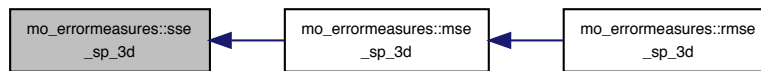
References `sse_sp_1d()`.

Referenced by `mse_sp_3d()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.10 mo_file Module Reference

Provides file names and units for mHM.

Variables

- character(len= *), parameter `version` = '5.8'
Current mHM model version.
- character(len= *), parameter `version_date` = 'December 2017'
Time of current mHM model version release.
- character(len= *), parameter `file_main` = 'mhm_driver.f90'
Driver file.
- character(len= *), parameter `file_namelist` = 'mhm.nml'
Namelist file name.
- integer, parameter `unamelist` = 30
Unit for namelist.
- character(len= *), parameter `file_namelist_param` = 'mhm_parameter.nml'
Parameter namelists file name.
- integer, parameter `unamelist_param` = 31
Unit for namelist.
- character(len= *), parameter `file_meteo_header` = 'header.txt'
Input nCols and nRows of binary meteo files are in header file.
- integer, parameter `umeteo_header` = 50
Unit for meteo header file.
- character(len= *), parameter `file_meteo_binary_end` = '.bin'
File ending of meteo files.
- integer, parameter `umeteo` = 51
Unit for meteo files.
- character(len= *), parameter `file_soil_database` = 'soil_classdefinition.txt'
Soil database file (iFlag_soilDB = 0) = classical mHM format.
- character(len= *), parameter `file_soil_database_1` = 'soil_classdefinition_iFlag_soilDB_1.txt'
Soil database file (iFlag_soilDB = 1)
- integer, parameter `usoil_database` = 52
Unit for soil data base.
- character(len= *), parameter `file_dem` = 'dem.asc'
DEM input data file.
- integer, parameter `udem` = 53
Unit for DEM input data file.
- character(len= *), parameter `file_slope` = 'slope.asc'

- slope input data file*

 - integer, parameter `uslope` = 54
 - Unit for slope input data file.*
- character(len= *), parameter `file_aspect` = 'aspect.asc'

aspect input data file

 - integer, parameter `uaspect` = 55
 - Unit for aspect input data file.*
- character(len= *), parameter `file_facc` = 'facc.asc'

flow accumulation input data file

 - integer, parameter `ufacc` = 56
 - Unit for flow accumulation input data file.*
- character(len= *), parameter `file_fdir` = 'fdir.asc'

flow direction input data file

 - integer, parameter `ufdir` = 57
 - Unit for flow direction input data file.*
- character(len= *), parameter `file_hydrogeoclass` = 'geology_class.asc'

hydrogeological classes input data file

 - integer, parameter `uhydrogeoclass` = 58
 - Unit for hydrogeological classes input data file.*
- character(len= *), parameter `file_soilclass` = 'soil_class.asc'

soil classes input data file

 - integer, parameter `usoilclass` = 59
 - Unit for soil classes input data file.*
- character(len= *), parameter `file_laiclass` = 'LAI_class.asc'

LAI classes input data file.

 - integer, parameter `ulaiclass` = 60
 - Unit for LAI input data file.*
- integer, parameter `ulcoverclass` = 61

Unit for LCover input data file.
- character(len= *), parameter `file_geolut` = 'geology_classdefinition.txt'

geological formation lookup table file

 - integer, parameter `ugeolut` = 64
 - Unit for geological formation lookup table file.*
- character(len= *), parameter `file_lailut` = 'LAI_classdefinition.txt'

LAI classes lookup table file.

 - integer, parameter `ulailut` = 65
 - Unit for LAI classes lookup table file.*
- integer, parameter `udischarge` = 66

unit for discharge time series
- character(len= *), parameter `file_defoutput` = 'mhm_outputs.nml'

file defining mHM's outputs

 - integer, parameter `udefoutput` = 67
 - Unit for file defining mHM's outputs.*
- character(len= *), parameter `file_config` = 'ConfigFile.log'

file containing mHM configuration

 - integer, parameter `uconfig` = 68
 - Unit for file containing mHM configuration.*
- character(len= *), parameter `file_opti` = 'FinalParam.out'

file defining optimization outputs (objective and parameter set)

 - integer, parameter `uopti` = 72
 - Unit for file optimization outputs (objective and parameter set)*

- character(len= *), parameter `file_opti_nml` = 'FinalParam.nml'
file defining optimization outputs in a namelist format (parameter set)
- integer, parameter `uopti_nml` = 73
Unit for file optimization outputs in a namelist format (parameter set)
- character(len= *), parameter `file_daily_discharge` = 'daily_discharge.out'
file defining optimization outputs
- integer, parameter `udaily_discharge` = 74
Unit for file optimization outputs.
- character(len= *), parameter `file_lai_header` = 'header.txt'
Input nCols and nRows of binary gridded LAI files are in header file.
- integer, parameter `ulai_header` = 75
Unit for LAI header file.
- character(len= *), parameter `file_lai_binary_end` = '.bin'
Binary file ending of LAI files.
- integer, parameter `ulai` = 76
Unit for binary LAI files.
- integer, parameter `utws` = 77
unit for tws time series

15.10.1 Detailed Description

Provides file names and units for mHM.

Provides all filenames as well as all units used for the hydrologic model mHM.

Author

Matthias Cuntz

Date

Jan 2012

15.10.2 Variable Documentation

15.10.2.1 file_aspect

```
character(len=*), parameter mo_file::file_aspect = 'aspect.asc'
```

aspect input data file

15.10.2.2 file_config

```
character(len=*), parameter mo_file::file_config = 'ConfigFile.log'
```

file containing mHM configuration

Referenced by `mo_write_ascii::write_configfile()`.

15.10.2.3 file_daily_discharge

```
character(len=*), parameter mo_file::file_daily_discharge = 'daily_discharge.out'
```

file defining optimization outputs

15.10.2.4 file_defoutput

```
character(len=*), parameter mo_file::file_defoutput = 'mhm_outputs.nml'
```

file defining mHM's outputs

15.10.2.5 file_dem

```
character(len=*), parameter mo_file::file_dem = 'dem.asc'
```

DEM input data file.

15.10.2.6 file_facc

```
character(len=*), parameter mo_file::file_facc = 'facc.asc'
```

flow accumulation input data file

15.10.2.7 file_fdir

```
character(len=*), parameter mo_file::file_fdir = 'fdir.asc'
```

flow direction input data file

15.10.2.8 file_geolut

```
character(len=*), parameter mo_file::file_geolut = 'geology_classdefinition.txt'
```

geological formation lookup table file

Referenced by mo_read_wrapper::read_data().

15.10.2.9 file_hydrogeoclass

```
character(len=*), parameter mo_file::file_hydrogeoclass = 'geology_class.asc'
```

hydrogeological classes input data file

15.10.2.10 file_lai_binary_end

```
character(len=*), parameter mo_file::file_lai_binary_end = '.bin'
```

Binary file ending of LAI files.

15.10.2.11 file_lai_header

```
character(len=*), parameter mo_file::file_lai_header = 'header.txt'
```

Input nCols and nRows of binary gridded LAI files are in header file.

15.10.2.12 file_laiclass

```
character(len=*), parameter mo_file::file_laiclass = 'LAI_class.asc'
```

LAI classes input data file.

15.10.2.13 file_lailut

```
character(len=*), parameter mo_file::file_lailut = 'LAI_classdefinition.txt'
```

LAI classes lookup table file.

15.10.2.14 file_main

```
character(len=*), parameter mo_file::file_main = 'mhm_driver.f90'
```

Driver file.

Referenced by mhm_driver().

15.10.2.15 file_meteo_binary_end

```
character(len=*), parameter mo_file::file_meteo_binary_end = '.bin'
```

File ending of meteo files.

Referenced by mo_read_meteo::read_meteo_bin().

15.10.2.16 file_meteo_header

```
character(len=*), parameter mo_file::file_meteo_header = 'header.txt'
```

Input nCols and nRows of binary meteo files are in header file.

Referenced by mo_startup::l2_variable_init(), and mo_read_meteo::read_meteo_bin().

15.10.2.17 file_namelist

```
character(len=*), parameter mo_file::file_namelist = 'mhm.nml'
```

Namelist file name.

Referenced by mo_read_config::read_config().

15.10.2.18 file_namelist_param

```
character(len=*), parameter mo_file::file_namelist_param = 'mhm_parameter.nml'
```

Parameter namelists file name.

15.10.2.19 file_opti

```
character(len=*), parameter mo_file::file_opti = 'FinalParam.out'
```

file defining optimization outputs (objective and parameter set)

Referenced by mo_write_ascii::write_optifile().

15.10.2.20 file_opti_nml

```
character(len=*), parameter mo_file::file_opti_nml = 'FinalParam.nml'
```

file defining optimization outputs in a namelist format (parameter set)

Referenced by mo_write_ascii::write_optinamelist().

15.10.2.21 file_slope

```
character(len=*), parameter mo_file::file_slope = 'slope.asc'
```

slope input data file

15.10.2.22 file_soil_database

```
character(len=*), parameter mo_file::file_soil_database = 'soil_classdefinition.txt'
```

Soil database file (iFlag_soilDB = 0) = classical mHM format.

15.10.2.23 file_soil_database_1

```
character(len=*), parameter mo_file::file_soil_database_1 = 'soil_classdefinition_iFlag_soilDB=1_B_1.txt'
```

Soil database file (iFlag_soilDB = 1)

15.10.2.24 file_soilclass

```
character(len=*), parameter mo_file::file_soilclass = 'soil_class.asc'
```

soil classes input data file

15.10.2.25 uaspect

```
integer, parameter mo_file::uaspect = 55
```

Unit for aspect input data file.

15.10.2.26 uconfig

```
integer, parameter mo_file::uconfig = 68
```

Unit for file containing mHM configuration.

Referenced by mo_write_ascii::write_configfile().

15.10.2.27 udaily_discharge

```
integer, parameter mo_file::udaily_discharge = 74
```

Unit for file optimazation outputs.

15.10.2.28 udefoutput

```
integer, parameter mo_file::udefoutput = 67
```

Unit for file defining mHM's outputs.

15.10.2.29 udem

```
integer, parameter mo_file::udem = 53
```

Unit for DEM input data file.

15.10.2.30 udischarge

```
integer, parameter mo_file::udischarge = 66
```

unit for discharge time series

15.10.2.31 ufacc

```
integer, parameter mo_file::ufacc = 56
```

Unit for flow accumulation input data file.

15.10.2.32 ufdir

```
integer, parameter mo_file::ufdir = 57
```

Unit for flow direction input data file.

15.10.2.33 ugeolut

```
integer, parameter mo_file::ugeolut = 64
```

Unit for geological formation lookup table file.

Referenced by mo_read_wrapper::read_data().

15.10.2.34 uhydrogeoclass

```
integer, parameter mo_file::uhydrogeoclass = 58
```

Unit for hydrogeological classes input data file.

15.10.2.35 ulai

```
integer, parameter mo_file::ulai = 76
```

Unit for binary LAI files.

15.10.2.36 ulai_header

```
integer, parameter mo_file::ulai_header = 75
```

Unit for LAI header file.

15.10.2.37 ulaiclass

```
integer, parameter mo_file::ulaiclass = 60
```

Unit for LAI input data file.

15.10.2.38 ulailut

```
integer, parameter mo_file::ulailut = 65
```

Unit for LAI classes lookup table file.

15.10.2.39 ulcoverclass

```
integer, parameter mo_file::ulcoverclass = 61
```

Unit for LCover input data file.

15.10.2.40 umeteo

```
integer, parameter mo_file::umeteo = 51
```

Unit for meteo files.

Referenced by `mo_read_meteo::read_meteo_bin()`.

15.10.2.41 umeteo_header

```
integer, parameter mo_file::umeteo_header = 50
```

Unit for meteo header file.

Referenced by `mo_startup::l2_variable_init()`, and `mo_read_meteo::read_meteo_bin()`.

15.10.2.42 unamelist

```
integer, parameter mo_file::unamelist = 30
```

Unit for namelist.

Referenced by `mo_read_config::read_config()`.

15.10.2.43 unamelist_param

```
integer, parameter mo_file::unamelist_param = 31
```

Unit for namelist.

15.10.2.44 uopti

```
integer, parameter mo_file::uopti = 72
```

Unit for file optimization outputs (objective and parameter set)

Referenced by `mo_write_ascii::write_optifile()`.

15.10.2.45 uopti_nml

```
integer, parameter mo_file::uopti_nml = 73
```

Unit for file optimization outputs in a namelist format (parameter set)

Referenced by `mo_write_ascii::write_optinamelist()`.

15.10.2.46 uslope

```
integer, parameter mo_file::uslope = 54
```

Unit for slope input data file.

15.10.2.47 usoil_database

```
integer, parameter mo_file::usoil_database = 52
```

Unit for soil data base.

Referenced by mo_soil_database::read_soil_lut().

15.10.2.48 usoilclass

```
integer, parameter mo_file::usoilclass = 59
```

Unit for soil classes input data file.

15.10.2.49 utws

```
integer, parameter mo_file::utws = 77
```

unit for tws time series

Referenced by mo_read_optional_data::read_basin_avg_tws().

15.10.2.50 version

```
character(len=*), parameter mo_file::version = '5.8'
```

Current mHM model version.

Referenced by mo_write_fluxes_states::createoutputfile(), mhm_driver(), and mo_write_ascii::write_configfile().

15.10.2.51 version_date

```
character(len=*), parameter mo_file::version_date = 'December 2017'
```

Time of current mHM model version release.

Referenced by mhm_driver().

15.11 mo_finish Module Reference

Finish a program gracefully.

Functions/Subroutines

- subroutine, public [finish](#) (name, text, unit)

Finish a program gracefully.

15.11.1 Detailed Description

Finish a program gracefully.

This module supplies a routine that writes out final comments and then stops.

Authors

Original of Echam5, (C) MPI-MET, Hamburg, Germany.

Modified Matthias Cuntz

Date

Jan 2011

15.11.2 Function/Subroutine Documentation

15.11.2.1 [finish\(\)](#)

```
subroutine, public mo_finish::finish (
    character(len=*), intent(in) name,
    character(len=*), intent(in), optional text,
    integer, intent(in), optional unit )
```

Finish a program gracefully.

Stop a program but writing out a message first that is separated from earlier output by `-----` (i.e. the separator of [mo_string_utils](#))

Parameters

in	<code>character(len=*) :: name</code>	First string separated from optional second by <code>:</code>
in	<code>character(len=*), optional :: text</code>	Second string separated by <code>:</code>
in	<code>integer, optional :: unit</code>	File unit for write (default: *)

Author

Written, Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

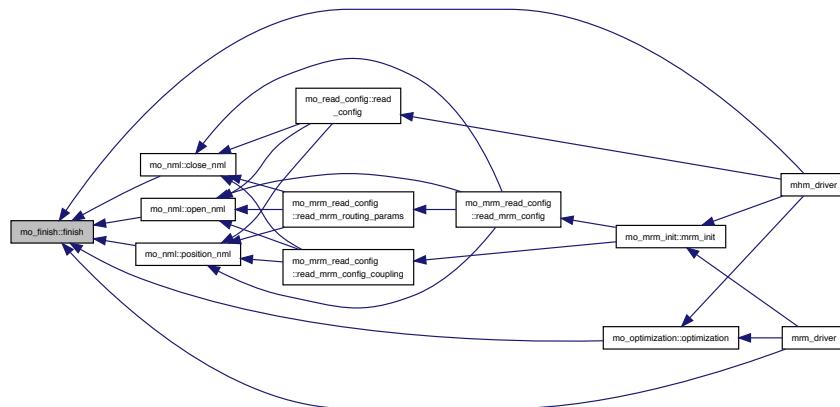
Date

Dec 2011

References [mo_string_utils::separator](#).

Referenced by [mo_nml::close_nml\(\)](#), [mhm_driver\(\)](#), [mrm_driver\(\)](#), [mo_nml::open_nml\(\)](#), [mo_optimization<->::optimization\(\)](#), and [mo_nml::position_nml\(\)](#).

Here is the caller graph for this function:



15.12 mo_global_variables Module Reference

Global variables ONLY used in reading, writing and startup.

Data Types

- type [basininfo](#)
- type [gridgeoref](#)
- type [soiltype](#)
- type [twstructure](#)

Variables

- character(1024), public [project_details](#)
- character(1024), public [setup_description](#)
- character(1024), public [simulation_type](#)
- character(256), public [conventions](#)
- character(1024), public [contact](#)
- character(1024), public [mhm_details](#)
- character(1024), public [history](#)
- integer(i4), public [timestep](#)
- integer(i4), dimension(:), allocatable, public [timestep_model_inputs](#)
- real(dp), dimension(:), allocatable, public [resolutionhydrology](#)
- real(dp), dimension(:), allocatable, public [resolutionrouting](#)
- integer(i4), dimension(:), allocatable, public [l0_basin](#)
- logical, public [read_restart](#)
- logical, public [write_restart](#)
- logical, public [perform_mpr](#)
- logical, public [read_meteo_weights](#)
- character(256), public [inputformat_meteo_forcings](#)
- character(256), public [inputformat_gridded_lai](#)
- integer(i4), public [timestep_lai_input](#)
- integer(i4), public [timestep_sm_input](#)
- integer(i4), public [timestep_neutrons_input](#)

- integer(i4), public [timestep_et_input](#)
- integer(i4), public [iflag_cordinate_sys](#)
- integer(i4), public [iflag_soildb](#)
- character(256), dimension(:), allocatable, public [dirmorpho](#)
- character(256), dimension(:), allocatable, public [dirlcover](#)
- character(256), dimension(:), allocatable, public [dirprecipitation](#)
- character(256), dimension(:), allocatable, public [dirtemperature](#)
- character(256), dimension(:), allocatable, public [dirmintemperature](#)
- character(256), dimension(:), allocatable, public [dirmaxtemperature](#)
- character(256), dimension(:), allocatable, public [dirnetradiation](#)
- character(256), dimension(:), allocatable, public [dirabsvapppressure](#)
- character(256), dimension(:), allocatable, public [dirwindspeed](#)
- character(256), dimension(:), allocatable, public [dirreferencecet](#)
- character(256), dimension(:), allocatable, public [dirout](#)
- character(256), dimension(:), allocatable, public [dirrestartout](#)
- character(256), dimension(:), allocatable, public [dirrestartin](#)
- character(256), dimension(:), allocatable, public [dirgridded_lai](#)
- character(256), dimension(:), allocatable, public [filelatlon](#)
- character(256), dimension(:), allocatable, public [dirsoil_moisture](#)
- character(256), dimension(:), allocatable, public [filetws](#)
- character(256), dimension(:), allocatable, public [dirneutrons](#)
- character(256), dimension(:), allocatable, public [direvapotranspiration](#)
- character(256), public [dirconfigout](#)
- character(256), public [dircommonfiles](#)
- real(dp), dimension(:), allocatable, target, public [ycoor](#)
- real(dp), dimension(:), allocatable, target, public [xcoor](#)
- real(dp), dimension(:, :), allocatable, target, public [lons](#)
- real(dp), dimension(:, :), allocatable, target, public [lats](#)
- real(dp), public [c2tstu](#)
- integer(i4), public [ntstepday](#)
- integer(i4), parameter, public [routingstates](#) = 2
- real(dp), public [tillagedepth](#)
- integer(i4), public [nsoilhorizons_mhm](#)
- real(dp), dimension(:), allocatable, public [horizonddepth_mhm](#)
- integer(i4), public [nsoiltypes](#)
- type([soiltype](#)), public [soildb](#)
- type([twsstructure](#)), public [basin_avg_tws_obs](#)
- real(dp), dimension(:, :), allocatable, public [basin_avg_tws_sim](#)
- integer(i4), public [nmeasperday_tws](#)
- integer(i4), public [ngeounits](#)
- integer(i4), dimension(:), allocatable, public [geounitlist](#)
- integer(i4), dimension(:), allocatable, public [geounitkar](#)
- real(dp), public [fracsealed_cityarea](#)
- integer(i4), public [nlcoverscene](#)
- character(256), dimension(:), allocatable, public [lcfilename](#)
- integer(i4), dimension(:, :), allocatable, public [lcyearid](#)
- integer(i4), public [nlaiclass](#)
- integer(i4), dimension(:), allocatable, public [laiunitlist](#)
- real(dp), dimension(:, :), allocatable, public [lailut](#)
- type([gridgeoref](#)), public [level0](#)
- type([gridgeoref](#)), public [level1](#)
- type([gridgeoref](#)), public [level2](#)
- real(dp), dimension(:), allocatable, public [l0_longitude](#)
- real(dp), dimension(:), allocatable, public [l0_latitude](#)
- real(dp), dimension(:), allocatable, public [l1_longitude](#)

- real(dp), dimension(:), allocatable, public [l1_latitude](#)
- real(dp), dimension(:), allocatable, public [l1_rect_longitude](#)
- real(dp), dimension(:), allocatable, public [l1_rect_latitude](#)
- type([period](#)), dimension(:), allocatable, public [warmper](#)
- type([period](#)), dimension(:), allocatable, public [evalper](#)
- type([period](#)), dimension(:), allocatable, public [simper](#)
- type([period](#)), public [readper](#)
- integer(i4), dimension(:), allocatable, public [warmingdays](#)
- integer(i4), public [nbasins](#)
- type([basininfo](#)), public [basin](#)
- logical, dimension(:), allocatable, target [l0_mask](#)
- real(dp), dimension(:), allocatable, target, public [l0_elev](#)
- real(dp), dimension(:), allocatable, public [l0_slope](#)
- real(dp), dimension(:), allocatable, public [l0_asp](#)
- integer(i4), dimension(:,:), allocatable, public [l0_soilid](#)
- integer(i4), dimension(:), allocatable, public [l0_geounit](#)
- integer(i4), dimension(:), allocatable, public [l0_lcover_lai](#)
- integer(i4), dimension(:,:), allocatable, target, public [l0_lcover](#)
- integer(i4), public [l0_ncells](#)
- integer(i4), dimension(:,:), allocatable, public [l0_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [l0_id](#)
- real(dp), dimension(:), allocatable, public [l0_slope_emp](#)
- real(dp), dimension(:,:), allocatable, public [l0_gridded_lai](#)
- real(dp), dimension(:), allocatable, public [l0_areacell](#)
- integer(i4), public [l1_ncells](#)
- integer(i4), dimension(:,:), allocatable, public [l1_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [l1_id](#)
- real(dp), dimension(:), allocatable, public [l1_areacell](#)
- integer(i4), dimension(:), allocatable, public [l1_upbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l1_downbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l1_leftbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l1_rightbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l1_ntcells_l0](#)
- real(dp), dimension(:,:,:), allocatable, public [l1_temp_weights](#)
- real(dp), dimension(:,:,:), allocatable, public [l1_pet_weights](#)
- real(dp), dimension(:,:,:), allocatable, public [l1_pre_weights](#)
- real(dp), dimension(:,:), allocatable, public [l1_pre](#)
- real(dp), dimension(:,:), allocatable, public [l1_temp](#)
- real(dp), dimension(:,:), allocatable, public [l1_pet](#)
- real(dp), dimension(:,:), allocatable, public [l1_tmin](#)
- real(dp), dimension(:,:), allocatable, public [l1_tmax](#)
- real(dp), dimension(:,:), allocatable, public [l1_netrad](#)
- real(dp), dimension(:,:), allocatable, public [l1_absvappress](#)
- real(dp), dimension(:,:), allocatable, public [l1_windspeed](#)
- real(dp), dimension(:,:), allocatable, public [l1_sm](#)
- logical, dimension(:,:), allocatable, public [l1_sm_mask](#)
- integer(i4) [ntimesteps_l1_sm](#)
- integer(i4) [nsoilhorizons_sm_input](#)
- real(dp), dimension(:,:), allocatable, public [l1_neutronsdata](#)
- logical, dimension(:,:), allocatable, public [l1_neutronsdata_mask](#)
- integer(i4) [ntimesteps_l1_neutrons](#)
- real(dp), dimension(:,:), allocatable, public [l1_et](#)
- logical, dimension(:,:), allocatable, public [l1_et_mask](#)
- integer(i4) [ntimesteps_l1_et](#)
- real(dp), dimension(:), allocatable, public [l1_fsealed](#)

- `real(dp), dimension(:), allocatable, public l1_fforest`
- `real(dp), dimension(:), allocatable, public l1_fperm`
- `real(dp), dimension(:), allocatable, public l1_inter`
- `real(dp), dimension(:), allocatable, public l1_snowpack`
- `real(dp), dimension(:), allocatable, public l1_sealstw`
- `real(dp), dimension(:, :), allocatable, public l1_soilmoist`
- `real(dp), dimension(:), allocatable, public l1_unsatstw`
- `real(dp), dimension(:), allocatable, public l1_satstw`
- `real(dp), dimension(:), allocatable, public l1_neutrons`
- `real(dp), dimension(:), allocatable, public l1_pet_calc`
- `real(dp), dimension(:, :), allocatable, public l1_aetsoil`
- `real(dp), dimension(:), allocatable, public l1_aetcanopy`
- `real(dp), dimension(:), allocatable, public l1_aetsealed`
- `real(dp), dimension(:), allocatable, public l1_baseflow`
- `real(dp), dimension(:, :), allocatable, public l1_infilsoil`
- `real(dp), dimension(:), allocatable, public l1_fastrunoff`
- `real(dp), dimension(:), allocatable, public l1_melt`
- `real(dp), dimension(:), allocatable, public l1_percol`
- `real(dp), dimension(:), allocatable, public l1_preeffect`
- `real(dp), dimension(:), allocatable, public l1_rain`
- `real(dp), dimension(:), allocatable, public l1_runoffseal`
- `real(dp), dimension(:), allocatable, public l1_slowrunoff`
- `real(dp), dimension(:), allocatable, public l1_snow`
- `real(dp), dimension(:), allocatable, public l1_throughfall`
- `real(dp), dimension(:), allocatable, public l1_total_runoff`
- `real(dp), dimension(:), allocatable, public l1_alpha`
- `real(dp), dimension(:), allocatable, public l1_degdayinc`
- `real(dp), dimension(:), allocatable, public l1_degdaymax`
- `real(dp), dimension(:), allocatable, public l1_degdaynopre`
- `real(dp), dimension(:), allocatable, public l1_degday`
- `real(dp), dimension(:), allocatable, public l1_karstloss`
- `real(dp), dimension(:), allocatable, public l1_fasp`
- `real(dp), dimension(:), allocatable, public l1_petlaicorfactor`
- `real(dp), dimension(:), allocatable, public l1_harsamcoeff`
- `real(dp), dimension(:, :), allocatable, public l1_prietayalpha`
- `real(dp), dimension(:, :), allocatable, public l1_aeroresist`
- `real(dp), dimension(:, :), allocatable, public l1_surfresist`
- `real(dp), dimension(:, :), allocatable, public l1_froots`
- `real(dp), dimension(:), allocatable, public l1_maxinter`
- `real(dp), dimension(:), allocatable, public l1_kfastflow`
- `real(dp), dimension(:), allocatable, public l1_kslowflow`
- `real(dp), dimension(:), allocatable, public l1_kbaseflow`
- `real(dp), dimension(:), allocatable, public l1_kperco`
- `real(dp), dimension(:, :), allocatable, public l1_soilmoistfc`
- `real(dp), dimension(:, :), allocatable, public l1_soilmoistsat`
- `real(dp), dimension(:, :), allocatable, public l1_soilmoistexp`
- `real(dp), dimension(:), allocatable, public l1_jarvis_thresh_c1`
- `real(dp), dimension(:), allocatable, public l1_tempthresh`
- `real(dp), dimension(:), allocatable, public l1_unsatthresh`
- `real(dp), dimension(:), allocatable, public l1_sealedthresh`
- `real(dp), dimension(:, :), allocatable, public l1_wiltingpoint`
- `real(dp), dimension(int(yearmonths, i4)), public evap_coeff`
- `real(dp), dimension(int(yearmonths, i4)), public fday_prec`
- `real(dp), dimension(int(yearmonths, i4)), public fnight_prec`
- `real(dp), dimension(int(yearmonths, i4)), public fday_pet`

- real(dp), dimension(int(yearmonths, i4)), public [fnight_pet](#)
- real(dp), dimension(int(yearmonths, i4)), public [fdlay_temp](#)
- real(dp), dimension(int(yearmonths, i4)), public [fnight_temp](#)
- integer(i4) [timestep_model_outputs](#)
- logical, dimension(noutflxstate) [outputflxstate](#)
- real(dp), dimension(:), allocatable, public [neutron_integral_afast](#)

15.12.1 Detailed Description

Global variables ONLY used in reading, writing and startup.

Authors

Luis Samaniego

Date

Dec 2012

15.12.2 Variable Documentation

15.12.2.1 basin

```
type(basininfo), public mo_global_variables::basin
```

Referenced by `mo_init_states::get_basin_info()`, `mo_startup::l0_check_input()`, `mo_startup::l1_variable_init()`, `mo_startup::l2_variable_init()`, `mo_read_latlon::read_latlon()`, `mo_write_ascii::write_configfile()`, and `mo_restart::write_restart_files()`.

15.12.2.2 basin_avg_tws_obs

```
type(twstructure), public mo_global_variables::basin_avg_tws_obs
```

Referenced by `mo_objective_function::extract_basin_avg_tws()`, and `mo_read_optional_data::read_basin_avg_tws()`.

15.12.2.3 basin_avg_tws_sim

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::basin_avg_tws_sim
```

Referenced by `mo_read_optional_data::read_basin_avg_tws()`.

15.12.2.4 c2tstu

```
real(dp), public mo_global_variables::c2tstu
```

Referenced by `mo_startup::constants_init()`.

15.12.2.5 contact

```
character(1024), public mo_global_variables::contact
```

15.12.2.6 conventions

```
character(256), public mo_global_variables::conventions
```

15.12.2.7 dirabsvappressure

```
character(256), dimension(:), allocatable, public mo_global_variables::dirabsvappressure
```

15.12.2.8 dircommonfiles

```
character(256), public mo_global_variables::dircommonfiles
```

15.12.2.9 dirconfigout

```
character(256), public mo_global_variables::dirconfigout
```

Referenced by `mo_write_ascii::write_configfile()`, `mo_write_ascii::write_optifile()`, and `mo_write_ascii::write_optinamelist()`.

15.12.2.10 direvapotranspiration

```
character(256), dimension(:), allocatable, public mo_global_variables::direvapotranspiration
```

Referenced by `mo_read_optional_data::read_evapotranspiration()`.

15.12.2.11 dirgridded_lai

```
character(256), dimension(:), allocatable, public mo_global_variables::dirgridded_lai
```

Referenced by `mo_prepare_gridded_lai::prepare_gridded_daily_lai_data()`, and `mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data()`.

15.12.2.12 dirlcover

```
character(256), dimension(:), allocatable, public mo_global_variables::dirlcover
```

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.13 dirmaxtemperature

```
character(256), dimension(:), allocatable, public mo_global_variables::dirmaxtemperature
```

15.12.2.14 dirminttemperature

```
character(256), dimension(:), allocatable, public mo_global_variables::dirminttemperature
```

15.12.2.15 dirmorpho

```
character(256), dimension(:), allocatable, public mo_global_variables::dirmorpho
```

Referenced by mo_write_ascii::write_configfile().

15.12.2.16 dirnetradiation

```
character(256), dimension(:), allocatable, public mo_global_variables::dirnetradiation
```

15.12.2.17 dirneutrons

```
character(256), dimension(:), allocatable, public mo_global_variables::dirneutrons
```

Referenced by mo_read_optional_data::read_neutrons().

15.12.2.18 dirout

```
character(256), dimension(:), allocatable, public mo_global_variables::dirout
```

Referenced by mo_write_fluxes_states::close(), mo_write_fluxes_states::createoutputfile(), and mo_write_ascii↵
::write_configfile().

15.12.2.19 dirprecipitation

```
character(256), dimension(:), allocatable, public mo_global_variables::dirprecipitation
```

Referenced by mo_startup::l2_variable_init(), and mo_write_ascii::write_configfile().

15.12.2.20 dirreferencecet

```
character(256), dimension(:), allocatable, public mo_global_variables::dirreferencecet
```

Referenced by mo_write_ascii::write_configfile().

15.12.2.21 dirrestartin

`character(256), dimension(:), allocatable, public mo_global_variables::dirrestartin`

Referenced by `mo_startup::initialise()`.

15.12.2.22 dirrestartout

`character(256), dimension(:), allocatable, public mo_global_variables::dirrestartout`

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.23 dirsoil_moisture

`character(256), dimension(:), allocatable, public mo_global_variables::dirsoil_moisture`

Referenced by `mo_read_optional_data::read_soil_moisture()`.

15.12.2.24 dirtemperature

`character(256), dimension(:), allocatable, public mo_global_variables::dirtemperature`

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.25 dirwindspeed

`character(256), dimension(:), allocatable, public mo_global_variables::dirwindspeed`

15.12.2.26 evalper

`type(period), dimension(:), allocatable, public mo_global_variables::evalper`

Referenced by `mo_write_fluxes_states::createoutputfile()`, `mo_objective_function::extract_basin_avg_tws()`, `mo←
_objective_function::objective_kge_q_rmse_et()`, `mo_objective_function::objective_kge_q_rmse_tws()`, and `mo←
write_ascii::write_configfile()`.

15.12.2.27 evap_coeff

`real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::evap_coeff`

15.12.2.28 fday_pet

`real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fday_pet`

15.12.2.29 fday_prec

```
real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fday_prec
```

15.12.2.30 fday_temp

```
real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fday_temp
```

15.12.2.31 filelatlon

```
character(256), dimension(:), allocatable, public mo_global_variables::filelatlon
```

Referenced by mo_read_latlon::read_latlon().

15.12.2.32 filetw

```
character(256), dimension(:), allocatable, public mo_global_variables::filetw
```

15.12.2.33 fnight_pet

```
real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fnight_pet
```

15.12.2.34 fnight_prec

```
real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fnight_prec
```

15.12.2.35 fnight_temp

```
real(dp), dimension(int(yearmonths,i4)), public mo_global_variables::fnight_temp
```

15.12.2.36 fracsealed_cityarea

```
real(dp), public mo_global_variables::fracsealed_cityarea
```

15.12.2.37 geounitkar

```
integer(i4), dimension(:), allocatable, public mo_global_variables::geounitkar
```

Referenced by mo_read_wrapper::read_data().

15.12.2.38 geounitlist

```
integer(i4), dimension(:), allocatable, public mo_global_variables::geounitlist
```

Referenced by `mo_read_wrapper::read_data()`.

15.12.2.39 history

```
character(1024), public mo_global_variables::history
```

15.12.2.40 horizondepth_mhm

```
real(dp), dimension(:), allocatable, public mo_global_variables::horizondepth_mhm
```

Referenced by `mo_soil_database::generate_soil_database()`, `mo_soil_database::read_soil_lut()`, and `mo_init_↔ states::variables_default_init()`.

15.12.2.41 iflag_cordinate_sys

```
integer(i4), public mo_global_variables::iflag_cordinate_sys
```

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.42 iflag_soildb

```
integer(i4), public mo_global_variables::iflag_soildb
```

Referenced by `mo_startup::initialise()`.

15.12.2.43 inputformat_gridded_lai

```
character(256), public mo_global_variables::inputformat_gridded_lai
```

Referenced by `mo_prepare_gridded_lai::prepare_gridded_daily_lai_data()`.

15.12.2.44 inputformat_meteo_forcings

```
character(256), public mo_global_variables::inputformat_meteo_forcings
```

15.12.2.45 l0_areacell

```
real(dp), dimension(:), allocatable, public mo_global_variables::l0_areacell
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.46 l0_asp

```
real(dp), dimension(:), allocatable, public mo_global_variables::l0_asp
```

Referenced by mo_startup::l0_check_input().

15.12.2.47 l0_basin

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l0_basin
```

Referenced by mo_startup::initialise(), and mo_restart::read_restart_config().

15.12.2.48 l0_cellcoor

```
integer(i4), dimension(:,:), allocatable, public mo_global_variables::l0_cellcoor
```

Referenced by mo_startup::l0_variable_init(), and mo_restart::write_restart_files().

15.12.2.49 l0_elev

```
real(dp), dimension(:), allocatable, target, public mo_global_variables::l0_elev
```

Referenced by mo_startup::l0_check_input().

15.12.2.50 l0_geounit

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l0_geounit
```

Referenced by mo_startup::l0_check_input().

15.12.2.51 l0_gridded_lai

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l0_gridded_lai
```

Referenced by mo_prepare_gridded_lai::prepare_gridded_daily_lai_data(), and mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data().

15.12.2.52 l0_id

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l0_id
```

Referenced by mo_startup::l0_variable_init(), and mo_restart::write_restart_files().

15.12.2.53 l0_latitude

```
real(dp), dimension(:), allocatable, public mo_global_variables::l0_latitude
```

Referenced by `mo_read_latlon::read_latlon()`.

15.12.2.54 `l0_lcover`

`integer(i4), dimension(:, :), allocatable, target, public mo_global_variables::l0_lcover`

15.12.2.55 `l0_lcover_lai`

`integer(i4), dimension(:), allocatable, public mo_global_variables::l0_lcover_lai`

15.12.2.56 `l0_longitude`

`real(dp), dimension(:), allocatable, public mo_global_variables::l0_longitude`

Referenced by `mo_read_latlon::read_latlon()`.

15.12.2.57 `l0_mask`

`logical, dimension(:), allocatable, target mo_global_variables::l0_mask`

15.12.2.58 `l0_ncells`

`integer(i4), public mo_global_variables::l0_ncells`

Referenced by `mo_startup::l0_variable_init()`, and `mo_write_ascii::write_configfile()`.

15.12.2.59 `l0_slope`

`real(dp), dimension(:), allocatable, public mo_global_variables::l0_slope`

Referenced by `mo_startup::l0_check_input()`, and `mo_startup::l0_variable_init()`.

15.12.2.60 `l0_slope_emp`

`real(dp), dimension(:), allocatable, public mo_global_variables::l0_slope_emp`

Referenced by `mo_startup::l0_variable_init()`.

15.12.2.61 `l0_soilid`

`integer(i4), dimension(:, :), allocatable, public mo_global_variables::l0_soilid`

Referenced by `mo_startup::l0_check_input()`, and `mo_startup::l0_variable_init()`.

15.12.2.62 l1_absvappress

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l1_absvappress
```

15.12.2.63 l1_aeroresist

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l1_aeroresist
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.64 l1_aetcanopy

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_aetcanopy
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.65 l1_aetsealed

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_aetsealed
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.66 l1_aetsoil

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l1_aetsoil
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.67 l1_alpha

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_alpha
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.68 l1_areacell

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_areacell
```

Referenced by mo_startup::l1_variable_init().

15.12.2.69 l1_baseflow

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_baseflow
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.70 l1_cellcoor

```
integer(i4), dimension(:, :), allocatable, public mo_global_variables::l1_cellcoor
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.71 l1_degday

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_degday
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.72 l1_degdayinc

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_degdayinc
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.73 l1_degdaymax

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_degdaymax
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.74 l1_degdaynopre

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_degdaynopre
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.75 l1_downbound_l0

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l1_downbound_l0
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.76 l1_et

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l1_et
```

Referenced by mo_objective_function::objective_kge_q_et(), and mo_objective_function::objective_kge_q_rmse←_et().

15.12.2.77 l1_et_mask

```
logical, dimension(:,:), allocatable, public mo_global_variables::l1_et_mask
```

Referenced by mo_objective_function::objective_kge_q_et(), and mo_objective_function::objective_kge_q_rmse←_et().

15.12.2.78 l1_fasp

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_fasp
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_←default_init(), and mo_restart::write_restart_files().

15.12.2.79 l1_fastrunoff

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_fastrunoff
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_←default_init(), and mo_restart::write_restart_files().

15.12.2.80 l1_fforest

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_fforest
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_←default_init(), and mo_restart::write_restart_files().

15.12.2.81 l1_fperm

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_fperm
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_←default_init(), and mo_restart::write_restart_files().

15.12.2.82 l1_froots

```
real(dp), dimension(:,:), allocatable, public mo_global_variables::l1_froots
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_←default_init(), and mo_restart::write_restart_files().

15.12.2.83 l1_fsealed

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_fsealed
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.84 l1_harsamcoeff

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_harsamcoeff
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.85 l1_id

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l1_id
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.86 l1_infilsoil

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_infilsoil
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.87 l1_inter

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_inter
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.88 l1_jarvis_thresh_c1

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_jarvis_thresh_c1
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.89 l1_karstloss

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_karstloss
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.90 l1_kbaseflow

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_kbaseflow
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.91 l1_kfastflow

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_kfastflow
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.92 l1_kperco

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_kperco
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.93 l1_kslowflow

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_kslowflow
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.94 l1_latitude

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_latitude
```

Referenced by `mo_read_latlon::read_latlon()`.

15.12.2.95 l1_leftbound_l0

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l1_leftbound_l0
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.96 l1_longitude

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_longitude
```

Referenced by `mo_read_latlon::read_latlon()`.

15.12.2.97 l1_maxinter

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_maxinter
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.98 l1_melt

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_melt
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.99 l1_ncells

```
integer(i4), public mo_global_variables::l1_ncells
```

Referenced by `mo_startup::l1_variable_init()`, and `mo_write_ascii::write_configfile()`.

15.12.2.100 l1_netrad

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_netrad
```

15.12.2.101 l1_neutrons

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_neutrons
```

Referenced by `mo_init_states::variables_alloc()`, and `mo_init_states::variables_default_init()`.

15.12.2.102 l1_neutronsdata

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_neutronsdata
```

15.12.2.103 l1_neutronsdata_mask

```
logical, dimension(:, :), allocatable, public mo_global_variables::l1_neutronsdata_mask
```

15.12.2.104 l1_ntcells_l0

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l1_ntcells_l0
```

Referenced by mo_startup::l1_variable_init().

15.12.2.105 l1_percol

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_percol
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_default_init(), and mo_restart::write_restart_files().

15.12.2.106 l1_pet

```
real(dp), dimension(:, :, :), allocatable, public mo_global_variables::l1_pet
```

15.12.2.107 l1_pet_calc

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_pet_calc
```

Referenced by mo_init_states::variables_alloc(), and mo_init_states::variables_default_init().

15.12.2.108 l1_pet_weights

```
real(dp), dimension(:, :, :), allocatable, public mo_global_variables::l1_pet_weights
```

15.12.2.109 l1_petlaicorfactor

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_petlaicorfactor
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_default_init(), and mo_restart::write_restart_files().

15.12.2.110 l1_pre

```
real(dp), dimension(:, :, :), allocatable, public mo_global_variables::l1_pre
```

15.12.2.111 l1_pre_weights

```
real(dp), dimension(:, :, :), allocatable, public mo_global_variables::l1_pre_weights
```

15.12.2.112 l1_preeffect

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_preeffect
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.113 l1_prietayalpha

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_prietayalpha
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.114 l1_rain

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_rain
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.115 l1_rect_latitude

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_rect_latitude
```

Referenced by `mo_write_fluxes_states::geocoordinates()`, and `mo_read_latlon::read_latlon()`.

15.12.2.116 l1_rect_longitude

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_rect_longitude
```

Referenced by `mo_write_fluxes_states::geocoordinates()`, and `mo_read_latlon::read_latlon()`.

15.12.2.117 l1_rightbound_l0

```
integer(i4), dimension(:), allocatable, public mo_global_variables::l1_rightbound_l0
```

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.118 l1_runoffseal

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_runoffseal
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.119 l1_satstw

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_satstw
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.120 l1_sealedthresh

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_sealedthresh
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.121 l1_sealstw

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_sealstw
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.122 l1_slowrunoff

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_slowrunoff
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.123 l1_sm

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_sm
```

Referenced by mo_objective_function::objective_kge_q_sm_corr().

15.12.2.124 l1_sm_mask

```
logical, dimension(:, :), allocatable, public mo_global_variables::l1_sm_mask
```

Referenced by mo_objective_function::objective_kge_q_sm_corr().

15.12.2.125 l1_snow

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_snow
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↵ default_init(), and mo_restart::write_restart_files().

15.12.2.126 l1_snowpack

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_snowpack
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.127 l1_soilmoist

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_soilmoist
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.128 l1_soilmoistexp

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_soilmoistexp
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.129 l1_soilmoistfc

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_soilmoistfc
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.130 l1_soilmoistsat

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_soilmoistsat
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.131 l1_surfresist

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_surfresist
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↔`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.132 l1_temp

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_temp
```

15.12.2.133 l1_temp_weights

```
real(dp), dimension(:, :, :), allocatable, public mo_global_variables::l1_temp_weights
```

15.12.2.134 l1_tempthresh

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_tempthresh
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↔ default_init(), and mo_restart::write_restart_files().

15.12.2.135 l1_throughfall

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_throughfall
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↔ default_init(), and mo_restart::write_restart_files().

15.12.2.136 l1_tmax

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_tmax
```

15.12.2.137 l1_tmin

```
real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_tmin
```

15.12.2.138 l1_total_runoff

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_total_runoff
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↔ default_init(), and mo_restart::write_restart_files().

15.12.2.139 l1_unsatstw

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_unsatstw
```

Referenced by mo_restart::read_restart_states(), mo_init_states::variables_alloc(), mo_init_states::variables_↔ default_init(), and mo_restart::write_restart_files().

15.12.2.140 l1_unsatthresh

```
real(dp), dimension(:), allocatable, public mo_global_variables::l1_unsatthresh
```

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.141 `l1_upbound_l0`

`integer(i4), dimension(:), allocatable, public mo_global_variables::l1_upbound_l0`

Referenced by `mo_startup::l1_variable_init()`.

15.12.2.142 `l1_wiltingpoint`

`real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_wiltingpoint`

Referenced by `mo_restart::read_restart_states()`, `mo_init_states::variables_alloc()`, `mo_init_states::variables_↵`
`default_init()`, and `mo_restart::write_restart_files()`.

15.12.2.143 `l1_windspeed`

`real(dp), dimension(:, :), allocatable, public mo_global_variables::l1_windspeed`

15.12.2.144 `lailut`

`real(dp), dimension(:, :), allocatable, public mo_global_variables::lailut`

15.12.2.145 `laiunitlist`

`integer(i4), dimension(:), allocatable, public mo_global_variables::laiunitlist`

15.12.2.146 `lats`

`real(dp), dimension(:, :), allocatable, target, public mo_global_variables::lats`

15.12.2.147 `lcfilename`

`character(256), dimension(:), allocatable, public mo_global_variables::lcfilename`

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.148 `lcyearid`

`integer(i4), dimension(:, :), allocatable, public mo_global_variables::lcyearid`

Referenced by mo_write_ascii::write_configfile().

15.12.2.149 level0

```
type(gridgeoref), public mo_global_variables::level0
```

Referenced by mo_init_states::get_basin_info(), mo_startup::l0_variable_init(), mo_startup::l1_variable_init(), mo_startup::l2_variable_init(), and mo_read_latlon::read_latlon().

15.12.2.150 level1

```
type(gridgeoref), public mo_global_variables::level1
```

Referenced by mo_write_fluxes_states::createoutputfile(), mo_init_states::get_basin_info(), mo_startup::l1_variable_init(), mo_meteo_forcings::meteo_forcings_wrapper(), mo_meteo_forcings::meteo_weights_wrapper(), and mo_read_latlon::read_latlon().

15.12.2.151 level2

```
type(gridgeoref), public mo_global_variables::level2
```

Referenced by mo_init_states::get_basin_info(), mo_startup::l2_variable_init(), mo_meteo_forcings::meteo_forcings_wrapper(), and mo_meteo_forcings::meteo_weights_wrapper().

15.12.2.152 lons

```
real(dp), dimension(:, :), allocatable, target, public mo_global_variables::lons
```

15.12.2.153 mhm_details

```
character(1024), public mo_global_variables::mhm_details
```

15.12.2.154 nbasins

```
integer(i4), public mo_global_variables::nbasins
```

Referenced by mo_startup::l1_variable_init(), mo_startup::l2_variable_init(), mhm_driver(), mo_objective_function::objective_et_kge_catchment_avg(), mo_objective_function::objective_kge_q_et(), mo_objective_function::objective_kge_q_rmse_et(), mo_objective_function::objective_kge_q_rmse_tws(), mo_objective_function::objective_kge_q_sm_corr(), mo_objective_function::objective_neutrons_kge_catchment_avg(), mo_objective_function::objective_sm_corr(), mo_objective_function::objective_sm_kge_catchment_avg(), mo_objective_function::objective_sm_pd(), mo_objective_function::objective_sm_sse_standard_score(), mo_read_optional_data::read_basin_avg_tws(), and mo_write_ascii::write_configfile().

15.12.2.155 neutron_integral_afast

```
real(dp), dimension(:), allocatable, public mo_global_variables::neutron_integral_afast
```

Referenced by `mo_startup::constants_init()`.

15.12.2.156 ngeounits

```
integer(i4), public mo_global_variables::ngeounits
```

Referenced by `mo_read_wrapper::read_data()`.

15.12.2.157 nlaiclass

```
integer(i4), public mo_global_variables::nlaiclass
```

15.12.2.158 nlcoverscene

```
integer(i4), public mo_global_variables::nlcoverscene
```

15.12.2.159 nmeasperday_tws

```
integer(i4), public mo_global_variables::nmeasperday_tws
```

Referenced by `mo_objective_function::extract_basin_avg_tws()`.

15.12.2.160 nsoilhorizons_mhm

```
integer(i4), public mo_global_variables::nsoilhorizons_mhm
```

Referenced by `mo_soil_database::generate_soil_database()`, `mo_write_fluxes_states::newoutputdataset()`, `mo_restart::read_restart_states()`, `mo_soil_database::read_soil_lut()`, `mo_write_fluxes_states::updatedataset()`, `mo_init_states::variables_alloc()`, and `mo_init_states::variables_default_init()`.

15.12.2.161 nsoilhorizons_sm_input

```
integer(i4) mo_global_variables::nsoilhorizons_sm_input
```

15.12.2.162 nsoiltypes

```
integer(i4), public mo_global_variables::nsoiltypes
```

Referenced by `mo_soil_database::generate_soil_database()`, and `mo_soil_database::read_soil_lut()`.

15.12.2.163 ntimesteps_l1_et

```
integer(i4) mo_global_variables::ntimesteps_l1_et
```

15.12.2.164 ntimesteps_l1_neutrons

```
integer(i4) mo_global_variables::ntimesteps_l1_neutrons
```

15.12.2.165 ntimesteps_l1_sm

```
integer(i4) mo_global_variables::ntimesteps_l1_sm
```

15.12.2.166 ntstepday

```
integer(i4), public mo_global_variables::ntstepday
```

Referenced by `mo_startup::constants_init()`, `mo_objective_function::extract_basin_avg_tws()`, `mo_write_fluxes_states::fluxesunit()`, `mo_meteo_forcings::is_read()`, and `mo_mhm_eval::mhm_eval()`.

15.12.2.167 outputflxstate

```
logical, dimension(noutflxstate) mo_global_variables::outputflxstate
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_write_fluxes_states::newoutputdataset()`, and `mo_write_fluxes_states::updatedataset()`.

15.12.2.168 perform_mpr

```
logical, public mo_global_variables::perform_mpr
```

Referenced by `mo_startup::initialise()`.

15.12.2.169 project_details

```
character(1024), public mo_global_variables::project_details
```

15.12.2.170 read_meteo_weights

```
logical, public mo_global_variables::read_meteo_weights
```

Referenced by `mo_meteo_forcings::prepare_meteo_forcings_data()`.

15.12.2.171 read_restart

logical, public mo_global_variables::read_restart

Referenced by mo_startup::initialise(), and mo_write_ascii::write_configfile().

15.12.2.172 readper

type(period), public mo_global_variables::readper

Referenced by mo_meteo_forcings::meteo_forcings_wrapper().

15.12.2.173 resolutionhydrology

real(dp), dimension(:), allocatable, public mo_global_variables::resolutionhydrology

Referenced by mo_startup::l1_variable_init(), and mo_write_ascii::write_configfile().

15.12.2.174 resolutionrouting

real(dp), dimension(:), allocatable, public mo_global_variables::resolutionrouting

15.12.2.175 routingstates

integer(i4), parameter, public mo_global_variables::routingstates = 2

15.12.2.176 setup_description

character(1024), public mo_global_variables::setup_description

15.12.2.177 simper

type(period), dimension(:), allocatable, public mo_global_variables::simper

Referenced by mo_meteo_forcings::chunk_size(), mo_write_fluxes_states::fluxesunit(), mo_meteo_forcings::is_read(), mo_prepare_gridded_lai::prepare_gridded_daily_lai_data(), and mo_write_ascii::write_configfile().

15.12.2.178 simulation_type

character(1024), public mo_global_variables::simulation_type

15.12.2.179 soildb

```
type(soiltype), public mo_global_variables::soildb
```

Referenced by mo_startup::initialise().

15.12.2.180 tillagedepth

```
real(dp), public mo_global_variables::tillagedepth
```

Referenced by mo_soil_database::read_soil_lut().

15.12.2.181 timestep

```
integer(i4), public mo_global_variables::timestep
```

Referenced by mo_startup::constants_init(), mo_write_fluxes_states::fluxesunit(), mo_meteo_forcings::is_read(), mo_read_config::read_config(), and mo_write_ascii::write_configfile().

15.12.2.182 timestep_et_input

```
integer(i4), public mo_global_variables::timestep_et_input
```

Referenced by mo_objective_function::objective_kge_q_rmse_et().

15.12.2.183 timestep_lai_input

```
integer(i4), public mo_global_variables::timestep_lai_input
```

Referenced by mo_prepare_gridded_lai::prepare_gridded_daily_lai_data().

15.12.2.184 timestep_model_inputs

```
integer(i4), dimension(:), allocatable, public mo_global_variables::timestep_model_inputs
```

Referenced by mo_meteo_forcings::is_read(), and mhm_driver().

15.12.2.185 timestep_model_outputs

```
integer(i4) mo_global_variables::timestep_model_outputs
```

Referenced by mo_write_fluxes_states::fluxesunit(), and mo_mhm_eval::mhm_eval().

15.12.2.186 timestep_neutrons_input

```
integer(i4), public mo_global_variables::timestep_neutrons_input
```

15.12.2.187 timestep_sm_input

`integer(i4), public mo_global_variables::timestep_sm_input`

15.12.2.188 warmingdays

`integer(i4), dimension(:), allocatable, public mo_global_variables::warmingdays`

Referenced by `mo_objective_function::extract_basin_avg_tws()`.

15.12.2.189 warmper

`type(period), dimension(:), allocatable, public mo_global_variables::warmper`

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.190 write_restart

`logical, public mo_global_variables::write_restart`

Referenced by `mo_write_ascii::write_configfile()`.

15.12.2.191 xcoor

`real(dp), dimension(:), allocatable, target, public mo_global_variables::xcoor`

15.12.2.192 ycoor

`real(dp), dimension(:), allocatable, target, public mo_global_variables::ycoor`

15.13 mo_init_states Module Reference

Initialization of all state variables of mHM.

Functions/Subroutines

- subroutine, public [variables_alloc](#) (iBasin)
Allocation of space for mHM related L1 and L11 variables.
- subroutine, public [variables_default_init](#) ()
Default initialization mHM related L1 variables.
- subroutine, public [get_basin_info](#) (iBasin, iLevel, nrows, ncols, ncells, iStart, iEnd, iStartMask, iEndMask, mask, xllcorner, yllcorner, cellsize)

Get basic basin information (e.g., nrows, ncols, indices, mask)

- subroutine, public [calculate_grid_properties](#) (nrowsIn, ncolsIn, xllcornerIn, yllcornerIn, cellsizeIn, nodata_valueIn, aimingResolution, nrowsOut, ncolsOut, xllcornerOut, yllcornerOut, cellsizeOut, nodata_valueOut)

Calculates basic grid properties at a required coarser level using information of a given finer level.

15.13.1 Detailed Description

Initialization of all state variables of mHM.

This module initializes all state variables required to run mHM. Two options are provided:

- (1) default values
- (2) from nc file

Author

Luis Samaniego & Rohini Kumar

Date

Dec 2012

15.13.2 Function/Subroutine Documentation

15.13.2.1 calculate_grid_properties()

```
subroutine, public mo_init_states::calculate_grid_properties (
    integer(i4), intent(in) nrowsIn,
    integer(i4), intent(in) ncolsIn,
    real(dp), intent(in) xllcornerIn,
    real(dp), intent(in) yllcornerIn,
    real(dp), intent(in) cellsizeIn,
    real(dp), intent(in) nodata_valueIn,
    real(dp), intent(in) aimingResolution,
    integer(i4), intent(out) nrowsOut,
    integer(i4), intent(out) ncolsOut,
    real(dp), intent(out) xllcornerOut,
    real(dp), intent(out) yllcornerOut,
    real(dp), intent(out) cellsizeOut,
    real(dp), intent(out) nodata_valueOut )
```

Calculates basic grid properties at a required coarser level using information of a given finer level.

Calculates basic grid properties at a required coarser level (e.g., L11) using information of a given finer level (e.g., L0). Basic grid properties such as nrows, ncols, xllcorner, yllcorner cellsize are estimated in this routine.

Parameters

in	<i>integer(i4) :: nrowsIn</i>	no. of rows at an input level
in	<i>integer(i4) :: ncolsIn</i>	no. of cols at an input level
in	<i>real(dp) :: xllcornerIn</i>	xllcorner at an input level
in	<i>real(dp) :: yllcornerIn</i>	yllcorner at an input level
in	<i>real(dp) :: cellsizeIn</i>	cell size at an input level
in	<i>real(dp) :: nodata_valueIn</i>	nodata value at an input level

Parameters

in	<i>real(dp) :: aimingResolution</i>	resolution of an output level
out	<i>integer(i4) :: nrowsOut</i>	no. of rows at an output level
out	<i>integer(i4) :: ncolsOut</i>	no. of cols at an output level
out	<i>real(dp) :: xllcornerOut</i>	xllcorner at an output level
out	<i>real(dp) :: yllcornerOut</i>	yllcorner at an output level
out	<i>real(dp) :: cellsizeOut</i>	cell size at an output level
out	<i>real(dp) :: nodata_valueOut</i>	nodata value at an output level

Note

resolutions of input and output levels should confirm each other.

Author

Matthias Zink & Rohini Kumar

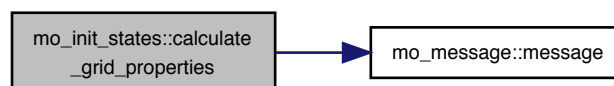
Date

Feb 2013

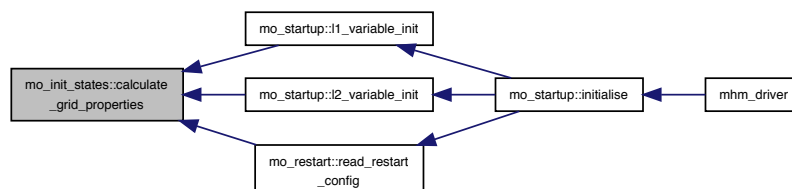
References `mo_message::message()`.

Referenced by `mo_startup::l1_variable_init()`, `mo_startup::l2_variable_init()`, and `mo_restart::read_restart_config()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.2.2 get_basin_info()

```

subroutine, public mo_init_states::get_basin_info (
    integer(i4), intent(in) iBasin,
    integer(i4), intent(in) iLevel,
    integer(i4), intent(out) nRows,
    integer(i4), intent(out) nCols,
    integer(i4), intent(out), optional nCells,
    integer(i4), intent(out), optional iStart,
    integer(i4), intent(out), optional iEnd,
    integer(i4), intent(out), optional iStartMask,
    integer(i4), intent(out), optional iEndMask,
    logical, dimension(:, :), intent(out), optional, allocatable mask,
    real(dp), intent(out), optional xllcorner,
    real(dp), intent(out), optional yllcorner,
    real(dp), intent(out), optional cellsize )

```

Get basic basin information (e.g., nRows, nCols, indices, mask)

Get basic basin information (e.g., nRows, nCols, indices, mask) for different levels (L0, L1, and L2).

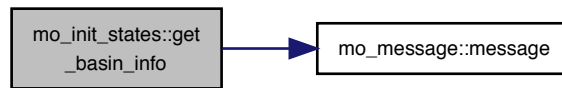
Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
in	<i>integer(i4) :: iLevel</i>	level id (e.g., 0, 1, 11, 2)
out	<i>integer(i4) :: nRows</i>	no. of rows
out	<i>integer(i4) :: nCols</i>	no. of columns
out	<i>integer(i4) :: nCells</i>	no. of cells
out	<i>integer(i4) :: iStart</i>	start cell index of a given basin at a given level
out	<i>integer(i4) :: iEnd</i>	end cell index of a given basin at a given level
out	<i>integer(i4) :: iStartMask</i>	start cell index of mask a given basin at a given level
out	<i>integer(i4) :: iEndMask</i>	end cell index of mask a given basin at a given level
out	<i>logical, optional :: mask</i>	Mask at a given level
out	<i>real(dp), optional :: xllcorner</i>	x coordinate of the lowerleft corner at a given level
out	<i>real(dp), optional :: yllcorner</i>	y coordinate of the lowerleft corner at a given level
out	<i>real(dp), optional :: cellsize</i>	cell size at a given level

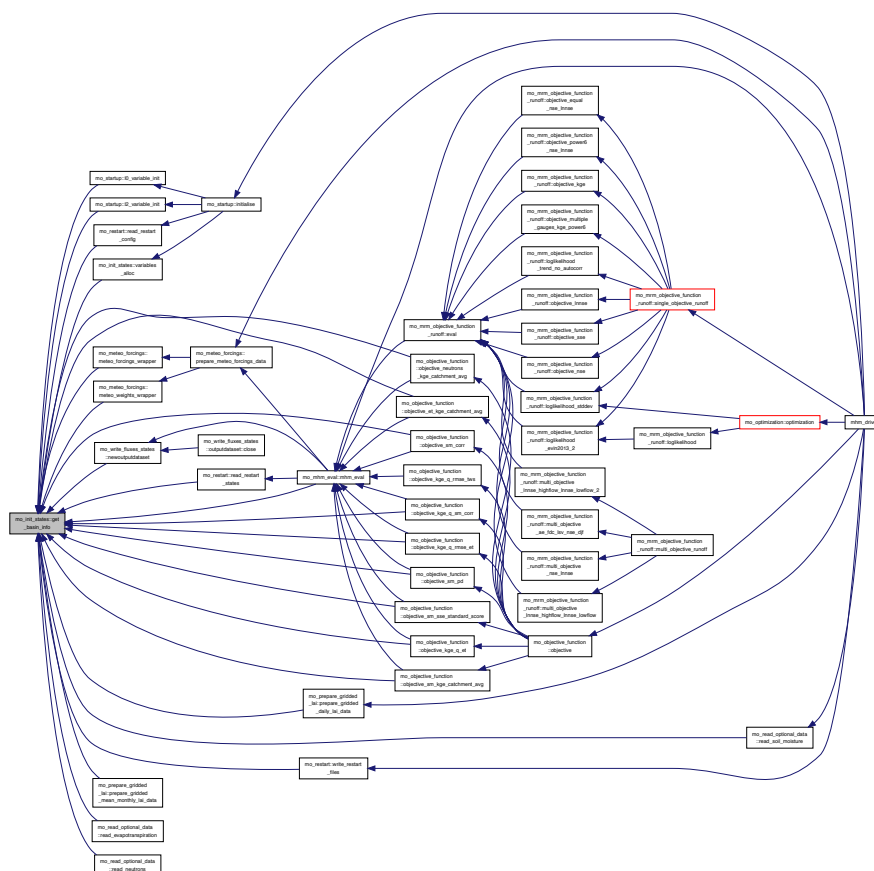
References mo_global_variables::basin, mo_global_variables::level0, mo_global_variables::level1, mo_global_variables::level2, and mo_message::message().

Referenced by mo_startup::l0_variable_init(), mo_startup::l2_variable_init(), mo_meteo_forcings::meteo_forcings←_wrapper(), mo_meteo_forcings::meteo_weights_wrapper(), mo_mhm_eval::mhm_eval(), mo_write_fluxes←states::newoutputdataset(), mo_objective_function::objective_et_kge_catchment_avg(), mo_objective_function←::objective_kge_q_et(), mo_objective_function::objective_kge_q_rmse_et(), mo_objective_function::objective←_kge_q_sm_corr(), mo_objective_function::objective_neutrons_kge_catchment_avg(), mo_objective_function←::objective_sm_corr(), mo_objective_function::objective_sm_kge_catchment_avg(), mo_objective_function←::objective_sm_pd(), mo_objective_function::objective_sm_sse_standard_score(), mo_prepare_gridded_lai←::prepare_gridded_daily_lai_data(), mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data(), mo←_read_optional_data::read_evapotranspiration(), mo_read_optional_data::read_neutrons(), mo_restart::read←restart_config(), mo_restart::read_restart_states(), mo_read_optional_data::read_soil_moisture(), variables←alloc(), and mo_restart::write_restart_files().

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.2.3 variables_alloc()

```

subroutine, public mo_init_states::variables_alloc (
    integer(i4), intent(in) iBasin )
  
```

Allocation of space for mHM related L1 and L11 variables.

Allocation of space for mHM related L1 and L11 variables (e.g., states, fluxes, and parameters) for a given basin. Variables allocated here is defined in them [mo_global_variables.f90](#) file. After allocating any variable in this routine, initialize them in the following `variables_default_init` subroutine:

Parameters

in	<i>integer(i4) :: iBasin</i>	- basin id
----	------------------------------	------------

Author

Rohini Kumar

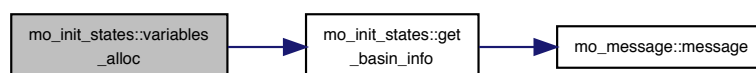
Date

Jan 2013

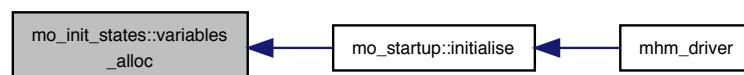
References get_basin_info(), mo_global_variables::l1_aeroresist, mo_global_variables::l1_aetcanopy, mo_global_variables::l1_aetsealed, mo_global_variables::l1_aetsoil, mo_global_variables::l1_alpha, mo_global_variables::l1_baseflow, mo_global_variables::l1_degday, mo_global_variables::l1_degdayinc, mo_global_variables::l1_degdaymax, mo_global_variables::l1_degdaynopre, mo_global_variables::l1_fasp, mo_global_variables::l1_fastrunoff, mo_global_variables::l1_fforest, mo_global_variables::l1_fperm, mo_global_variables::l1_froots, mo_global_variables::l1_fsealed, mo_global_variables::l1_harsamcoeff, mo_global_variables::l1_infilsoil, mo_global_variables::l1_inter, mo_global_variables::l1_jarvis_thresh_c1, mo_global_variables::l1_karstloss, mo_global_variables::l1_kbaseflow, mo_global_variables::l1_kfastflow, mo_global_variables::l1_kperco, mo_global_variables::l1_kslowflow, mo_global_variables::l1_maxinter, mo_global_variables::l1_melt, mo_global_variables::l1_neutrons, mo_global_variables::l1_percol, mo_global_variables::l1_pet_calc, mo_global_variables::l1_petlaicorfactor, mo_global_variables::l1_preeffect, mo_global_variables::l1_prietayalpha, mo_global_variables::l1_rain, mo_global_variables::l1_runoffseal, mo_global_variables::l1_satstw, mo_global_variables::l1_sealedthresh, mo_global_variables::l1_sealstw, mo_global_variables::l1_slowrunoff, mo_global_variables::l1_snow, mo_global_variables::l1_snowpack, mo_global_variables::l1_soilmoist, mo_global_variables::l1_soilmoistexp, mo_global_variables::l1_soilmoistfc, mo_global_variables::l1_soilmoistsat, mo_global_variables::l1_surfresist, mo_global_variables::l1_tempthresh, mo_global_variables::l1_throughfall, mo_global_variables::l1_total_runoff, mo_global_variables::l1_unsatstw, mo_global_variables::l1_unsatthresh, mo_global_variables::l1_wiltingpoint, mo_global_variables::soilhorizons_mhm, and mo_mhm_constants::yearmonths_i4.

Referenced by mo_startup::initialise().

Here is the call graph for this function:



Here is the caller graph for this function:



15.13.2.4 variables_default_init()

```
subroutine, public mo_init_states::variables_default_init ( )
```

Default initialization mHM related L1 variables.

Default initialization of mHM related L1 variables (e.g., states, fluxes, and parameters) as per given constant values given in [mo_mhm_constants](#). Variables initialized here is defined in the [mo_global_variables.f90](#) file. Only Variables that are defined in the variables_alloc subroutine are initialized here. If a variable is added or removed here, then it also has to be added or removed in the subroutine state_variables_set in the module [mo_restart](#) and in the subroutine set_state in the module mo_set_netcdf_restart.

References mo_mhm_constants::c1_initstatesm, mo_global_variables::horizondepth_mhm, mo_global_variables::l1_aeroresist, mo_global_variables::l1_aetcanopy, mo_global_variables::l1_aetsealed, mo_global_variables::l1_aetsoil, mo_global_variables::l1_alpha, mo_global_variables::l1_baseflow, mo_global_variables::l1_degday, mo_global_variables::l1_degdayinc, mo_global_variables::l1_degdaymax, mo_global_variables::l1_degdaynopre, mo_global_variables::l1_fasp, mo_global_variables::l1_fastrunoff, mo_global_variables::l1_fforest, mo_global_variables::l1_fperm, mo_global_variables::l1_froots, mo_global_variables::l1_fsealed, mo_global_variables::l1_harsamcoeff, mo_global_variables::l1_infilsoil, mo_global_variables::l1_inter, mo_global_variables::l1_jarvis_thresh_c1, mo_global_variables::l1_karstloss, mo_global_variables::l1_kbaseflow, mo_global_variables::l1_kfastflow, mo_global_variables::l1_kperco, mo_global_variables::l1_kslowflow, mo_global_variables::l1_maxinter, mo_global_variables::l1_melt, mo_global_variables::l1_neutrons, mo_global_variables::l1_percol, mo_global_variables::l1_pet_calc, mo_global_variables::l1_petlaicorfactor, mo_global_variables::l1_preeffect, mo_global_variables::l1_prietayalpha, mo_global_variables::l1_rain, mo_global_variables::l1_runoffseal, mo_global_variables::l1_satstw, mo_global_variables::l1_sealedthresh, mo_global_variables::l1_sealstw, mo_global_variables::l1_slowrunoff, mo_global_variables::l1_snow, mo_global_variables::l1_snowpack, mo_global_variables::l1_soilmoist, mo_global_variables::l1_soilmoistexp, mo_global_variables::l1_soilmoistfc, mo_global_variables::l1_soilmoistsat, mo_global_variables::l1_surfsesist, mo_global_variables::l1_tempthresh, mo_global_variables::l1_throughfall, mo_global_variables::l1_total_runoff, mo_global_variables::l1_unsatstw, mo_global_variables::l1_unsatthresh, mo_global_variables::l1_wiltingpoint, mo_global_variables::nsoilhorizons_mhm, mo_mhm_constants::p1_initstatefluxes, mo_mhm_constants::p2_initstatefluxes, mo_mhm_constants::p3_initstatefluxes, mo_mhm_constants::p4_initstatefluxes, and mo_mhm_constants::p5_initstatefluxes.

Referenced by mo_mhm_eval::mhm_eval().

[illegible]

Julian date conversion routines.

- interface **setcalendar**

- subroutine `setcalendarstring` (selector)
Set module private variable calendar.

- subroutine [setcalendarinteger](#) (selector)
Set module private variable calendar.
- pure integer(i4) function [selectcalendar](#) (selector)
Select a calendar.
- elemental subroutine, public [caldat](#) (julian, dd, mm, yy, [calendar](#))
Day, month and year from Julian day in the current or given calendar.
- elemental subroutine, public [dec2date](#) (julian, dd, mm, yy, hh, nn, ss, [calendar](#))
Day, month, year, hour, minute, and second from fractional Julian day in the current or given calendar.
- elemental real(dp) function, public [date2dec](#) (dd, mm, yy, hh, nn, ss, [calendar](#))
Fractional Julian day from day, month, year, hour, minute, second in the current calendar.
- elemental integer(i4) function, public [julday](#) (dd, mm, yy, [calendar](#))
Julian day from day, month and year in the current or given calendar.
- elemental subroutine, public [caldatjulian](#) (julian, dd, mm, yy)
Day, month and year from Julian day.
- elemental real(dp) function [date2decjulian](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second.
- elemental subroutine [dec2datejulian](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day.
- elemental integer(i4) function [juldayjulian](#) (dd, mm, yy)
Julian day from day, month and year.
- elemental integer(i4) function, public [ndays](#) (dd, mm, yy)
IMSL Julian day from day, month and year.
- elemental subroutine, public [ndyin](#) (julian, dd, mm, yy)
Day, month and year from IMSL Julian day.
- elemental subroutine [caldat360](#) (julian, dd, mm, yy)
Day, month and year from Julian day in a 360 day calendar.
- elemental integer(i4) function [julday360](#) (dd, mm, yy)
Julian day from day, month and year in a 360_day calendar.
- elemental subroutine [dec2date360](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day in a 360_day calendar.
- elemental real(dp) function [date2dec360](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second in 360 day calendar.
- elemental subroutine [caldat365](#) (julian, dd, mm, yy)
Day, month and year from Julian day in a 365 day calendar.
- elemental integer(i4) function [julday365](#) (dd, mm, yy)
Julian day from day, month and year in a 365_day calendar.
- elemental subroutine [dec2date365](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day in a 365_day calendar.
- elemental real(dp) function [date2dec365](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second in 365 day calendar.

Variables

- integer(i4), save, private [calendar](#) = 1

15.14.1 Detailed Description

Julian date conversion routines.

Julian date to and from day, month, year, and also from day, month, year, hour, minute, and second. Also convenience routines for Julian dates of IMSL are provided.

Note

Julian day definition starts at noon of the 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc.

This means that Julian day definition starts as 01.01.-4712 in astronomical units.

julday and caldat start at midnight of the 1st January 4713 BC. So date2dec and julday as well as dec2date and caldat are shifted by half a day.

Use date2dec with dec2date together for fractional Julian dates and use julday with caldat together for integer Julian days.

Author

Matthias Cuntz

Date

Dec 2011

15.14.2 Function/Subroutine Documentation

15.14.2.1 caldat()

```
elemental subroutine, public mo_julian::caldat (
    integer(i4), intent(in) julian,
    integer(i4), intent(out) dd,
    integer(i4), intent(out) mm,
    integer(i4), intent(out) yy,
    integer(i4), intent(in), optional calendar )
```

Day, month and year from Julian day in the current or given calendar.

Wrapper around the calendar specific caldat procedures. Inverse of the function julday. Here julian is input as a Julian Day Number, and the routine outputs dd, mm, and yy as the day, month, and year on which the specified Julian Day started at noon. The zeroth Julian Day depends on the called procedure. See their documentation for details.

Parameters

in	<i>integer(i4) :: julday</i>	Julian day
out	<i>integer(i4) :: dd</i>	Day in month of Julian day
out	<i>integer(i4) :: mm</i>	Month in year of Julian day
out	<i>integer(i4) :: yy</i>	Year of Julian day
in	<i>integer(i4) :: calendar</i>	The calendar to use, the global calendar will be used by default

Author

Written, David Schaefer

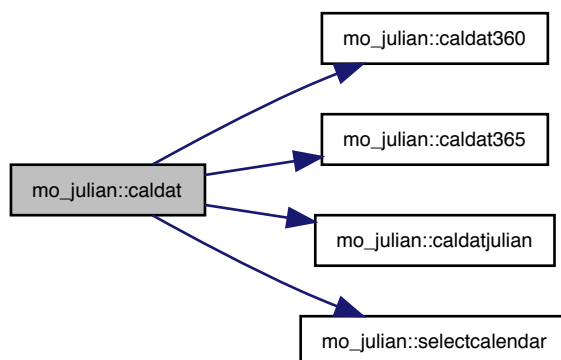
Date

Jan 2015

References `caldat360()`, `caldat365()`, `caldatjulian()`, and `selectcalendar()`.

Referenced by `mo_meteo_forcings::chunk_size()`, `dec2datejulian()`, `mo_meteo_forcings::is_read()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_eval::mrm_eval()`, `mo_mrm_write::mrm_write_output_fluxes()`, `ndyin()`, `mo_objective_function::objective_kge_q_rmse_et()`, `mo_objective_function::objective_kge_q_rmse_tws()`, and `mo_read_forcing_nc::read_forcing_nc()`.

Here is the call graph for this function:



[illegible]

```

elemental subroutine mo_julian::caldat360 (
    integer(i4), intent(in)  julian,
    integer(i4), intent(out) dd,
    integer(i4), intent(out) mm,
    integer(i4), intent(out) yy ) [private]

```

Inverse of the function julday360. Here julian is input as a Julian Day Number, and the routine outputs dd, mm, and yy as the day, month, and year on which the specified Julian Day started at noon. The zeroth Julian Day here is 01.01.0000

in	<i>integer(i4) :: julday</i>	Julian day
out	<i>integer(i4) :: dd</i>	Day in month of Julian day

Parameters

out	<i>integer(i4) :: mm</i>	Month in year of Julian day
out	<i>integer(i4) :: yy</i>	Year of Julian day

Author

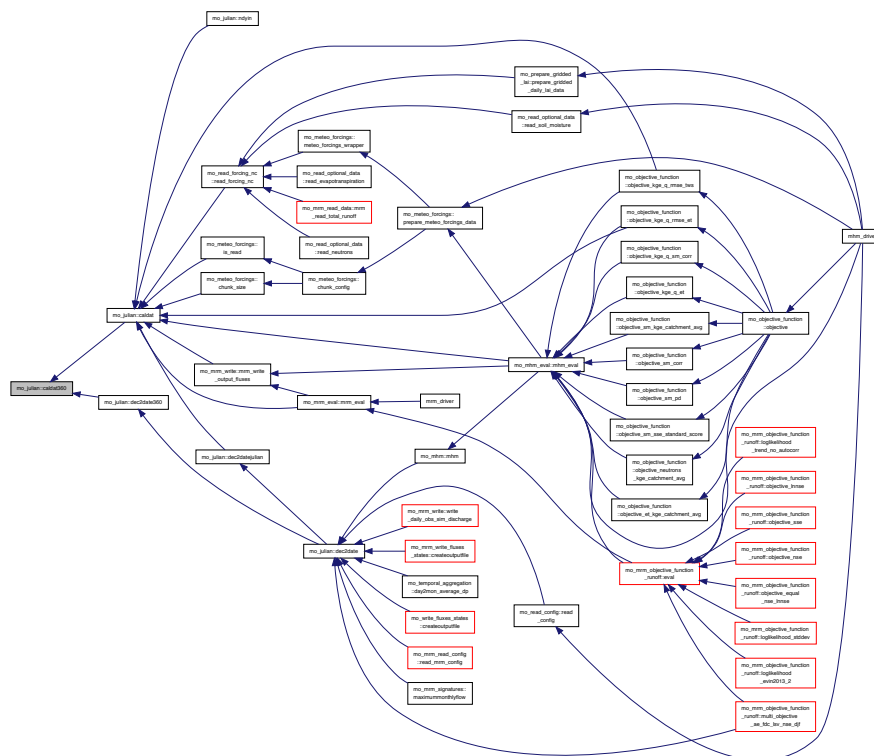
Written, David Schaefer

Date

Oct 2015

Referenced by caldat(), and dec2date360().

Here is the caller graph for this function:



15.14.2.3 caldat365()

```

elemental subroutine mo_julian::caldat365 (
  integer(i4), intent(in)  julian,
  integer(i4), intent(out) dd,
  integer(i4), intent(out) mm,
  integer(i4), intent(out) yy ) [private]

```

Day, month and year from Julian day in a 365 day calendar.

Inverse of the function `julday365`. Here `julian` is input as a Julian Day Number, and the routine outputs `dd`, `mm`, and `yy` as the day, month, and year on which the specified Julian Day started at noon. The zeroth Julian Day here is 01.01.0000

Parameters

in	<i>integer(i4) :: julian</i>	Julian day
out	<i>integer(i4) :: dd</i>	Day in month of Julian day
out	<i>integer(i4) :: mm</i>	Month in year of Julian day
out	<i>integer(i4) :: yy</i>	Year of Julian day

Author

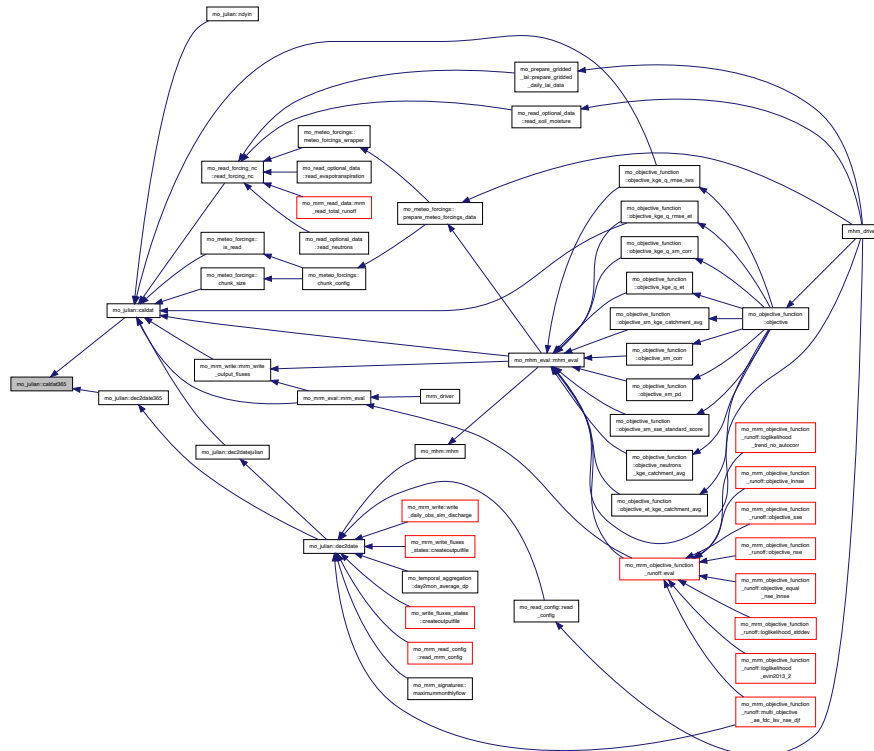
Written, David Schaefer

Date

Dec 2015

Referenced by `caldat()`, and `dec2date365()`.

Here is the caller graph for this function:



15.14.2.4 caldatjulian()

```

elemental subroutine, public mo_julian::caldatjulian (
    integer(i4), intent(in)  julian,
    integer(i4), intent(out) dd,
    integer(i4), intent(out) mm,
    integer(i4), intent(out) yy )

```

Day, month and year from Julian day.

Inverse of the function `juldayJulian`. Here `julian` is input as a Julian Day Number, and the routine outputs `id`, `mm`, and `yy` as the day, month, and year on which the specified Julian Day started at noon. The zeroth Julian Day is 01.01.-4712, i.e. the 1st January 4713 BC. Julian day definition starts at 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc. This means that Julian day definition starts as 01.01.-4712 in astronomical units.

Parameters

in	<i>integer(i4) :: Julday</i>	Julian day
out	<i>integer(i4) :: dd</i>	Day in month of Julian day
out	<i>integer(i4) :: mm</i>	Month in year of Julian day
out	<i>integer(i4) :: yy</i>	Year of Julian day

Note

Julian day definition starts at noon of the 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc.

This means that Julian day definition starts as 01.01.-4712 in astronomical units.

`julday` and `caldat` start at midnight of the 1st January 4713 BC. So `date2decJulian` and `juldayJulian` as well as `dec2dateJulian` and `caldatJulian` are shifted by half a day.

Use `date2decJulian` with `dec2dateJulian` together for fractional Julian dates and use `juldayJulian` with `caldatJulian` together for integer Julian days.

Author

Written, Matthias Cuntz - modified `julday` from Numerical Recipes

Date

Dec 2011 Modified Matthias Cuntz, May 2014 - changed to new algorithm with astronomical units removed
numerical recipes David Schaefer, Jan 2016 - renamed procedure

References `mo_kind::i4`, and `mo_kind::i8`.

Referenced by `caldat()`.

[illegible]

```

elemental real(dp) function, public mo_julian::date2dec (
    integer(i4), intent(in), optional dd,
    integer(i4), intent(in), optional mm,
    integer(i4), intent(in), optional yy,
    integer(i4), intent(in), optional hh,
    integer(i4), intent(in), optional nn,
    integer(i4), intent(in), optional ss,
    integer(i4), intent(in), optional calendar )

```

Wrapper around the calendar specific date2dec procedures. In this routine date2dec returns the fractional Julian Day that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day depends on the called procedure. See their documentation for details.

in	<i>integer(i4), optional :: dd</i>	Day in month of Julian day (default: 1)
in	<i>integer(i4), optional :: mm</i>	Month in year of Julian day (default: 1)
in	<i>integer(i4), optional :: yy</i>	Year of Julian day (default: 1)

Parameters

in	<i>integer(i4), optional :: hh</i>	Hours of Julian day (default: 0)
in	<i>integer(i4), optional :: nn</i>	Minutes of hour of Julian day (default: 0)
in	<i>integer(i4), optional :: ss</i>	Seconds of minute of hour of Julian day (default: 0)
in	<i>integer(i4), optional :: calendar</i>	The calendar to use, the global calendar will be used by default

Returns

real(dp) :: date2dec ! Fractional Julian day

Author

Written, David Schaefer

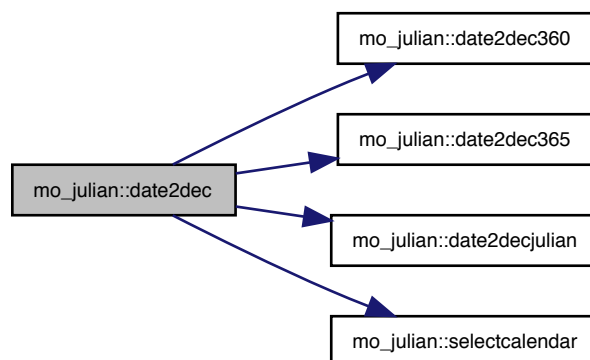
Date

Jan 2015

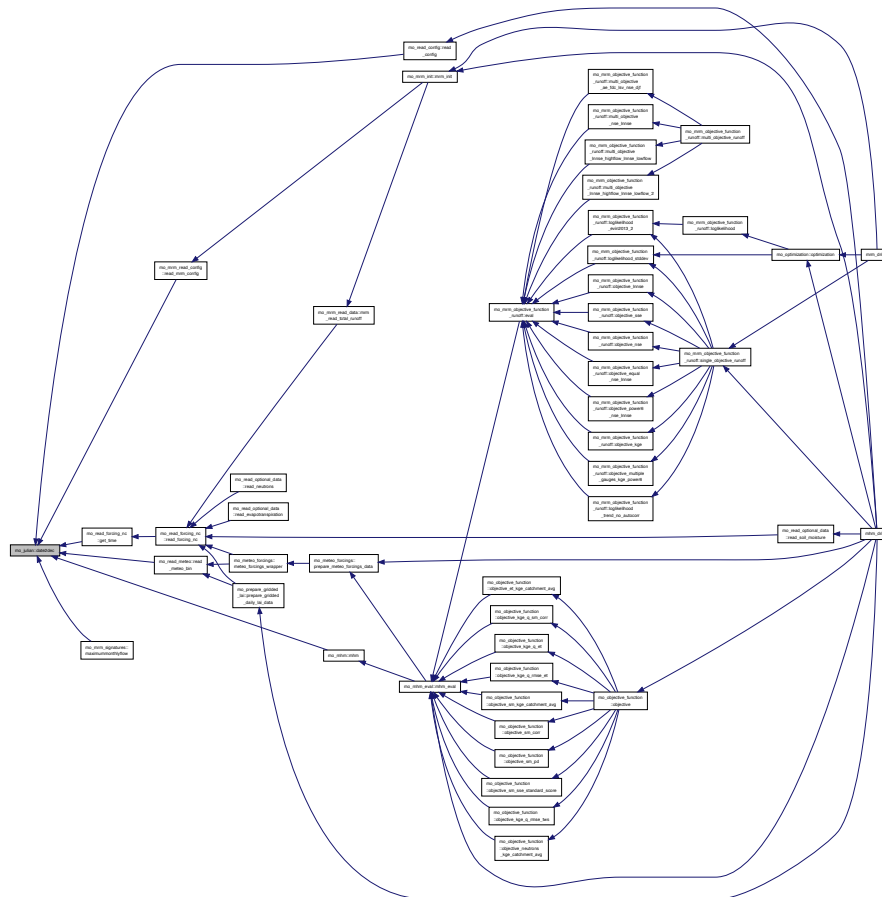
References date2dec360(), date2dec365(), date2decjulian(), and selectcalendar().

Referenced by mo_read_forcing_nc::get_time(), mo_mrm_signatures::maximummonthlyflow(), mo_mhm::mhm(), mo_read_config::read_config(), mo_read_meteo::read_meteo_bin(), and mo_mrm_read_config::read_mrm_config().

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.2.6 date2dec360()

```

elemental real(dp) function mo_julian::date2dec360 (
  integer(i4), intent(in), optional dd,
  integer(i4), intent(in), optional mm,
  integer(i4), intent(in), optional yy,
  integer(i4), intent(in), optional hh,
  integer(i4), intent(in), optional nn,
  integer(i4), intent(in), optional ss ) [private]

```

Fractional Julian day from day, month, year, hour, minute, second in 360 day calendar.

In this routine date2dec360 returns the fractional Julian Day that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day is 01.01.0000 at noon.

Parameters

in	<i>integer(i4), optional :: dd</i>	Day in month of Julian day (default: 1)
in	<i>integer(i4), optional :: mm</i>	Month in year of Julian day (default: 1)
in	<i>integer(i4), optional :: yy</i>	Year of Julian day (default: 1)
in	<i>integer(i4), optional :: hh</i>	Hours of Julian day (default: 0)
in	<i>integer(i4), optional :: nn</i>	Minutes of hour of Julian day (default: 0)

Parameters

in	<i>integer(i4), optional :: ss</i>	Seconds of minute of hour of Julian day (default: 0)
----	------------------------------------	--

Returns

real(dp) :: date2dec360 ! Fractional Julian day

Author

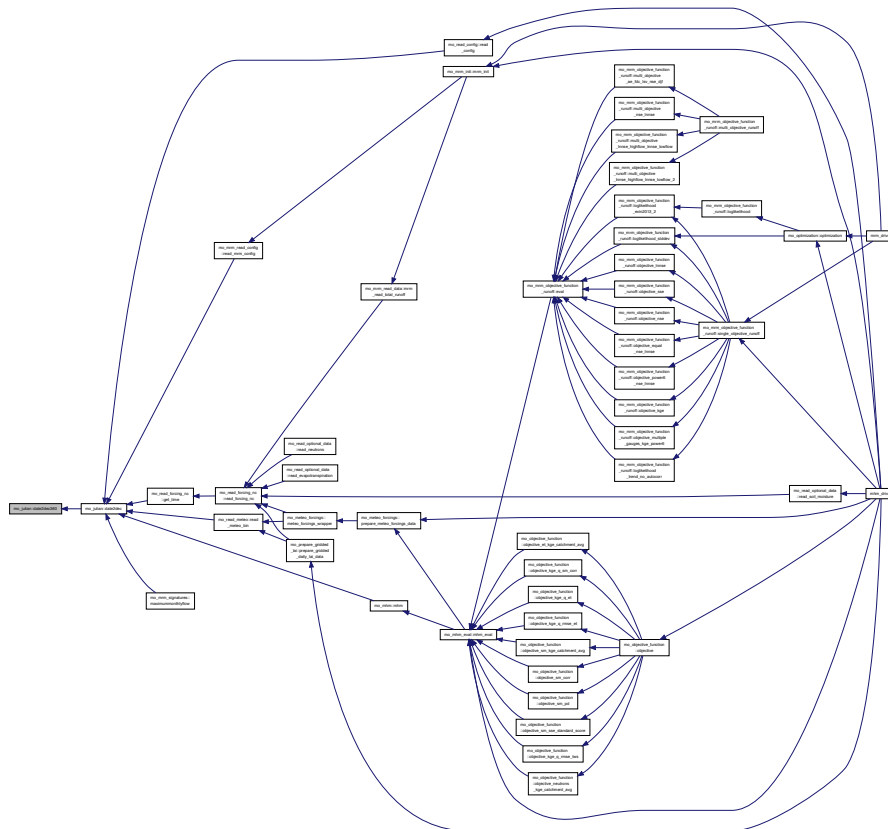
Written, David Schaefer

Date

Oct 2015

Referenced by `date2dec()`.

Here is the caller graph for this function:



15.14.2.7 date2dec365()

```

elemental real(dp) function mo_julian::date2dec365 (
    integer(i4), intent(in), optional dd,
    integer(i4), intent(in), optional mm,

```

```

integer(i4), intent(in), optional yy,
integer(i4), intent(in), optional hh,
integer(i4), intent(in), optional nn,
integer(i4), intent(in), optional ss ) [private]

```

Fractional Julian day from day, month, year, hour, minute, second in 365 day calendar.

In this routine `date2dec365` returns the fractional Julian Day that begins at noon of the calendar date specified by month `mm`, day `dd`, and year `yy`, all integer variables. The zeroth Julian Day is 01.01.0000 at noon.

Parameters

in	<i>integer(i4), optional :: dd</i>	Day in month of Julian day (default: 1)
in	<i>integer(i4), optional :: mm</i>	Month in year of Julian day (default: 1)
in	<i>integer(i4), optional :: yy</i>	Year of Julian day (default: 1)
in	<i>integer(i4), optional :: hh</i>	Hours of Julian day (default: 0)
in	<i>integer(i4), optional :: nn</i>	Minutes of hour of Julian day (default: 0)
in	<i>integer(i4), optional :: ss</i>	Seconds of minute of hour of Julian day (default: 0)

Returns

`real(dp) :: date2dec365 ! Fractional Julian day`

Author

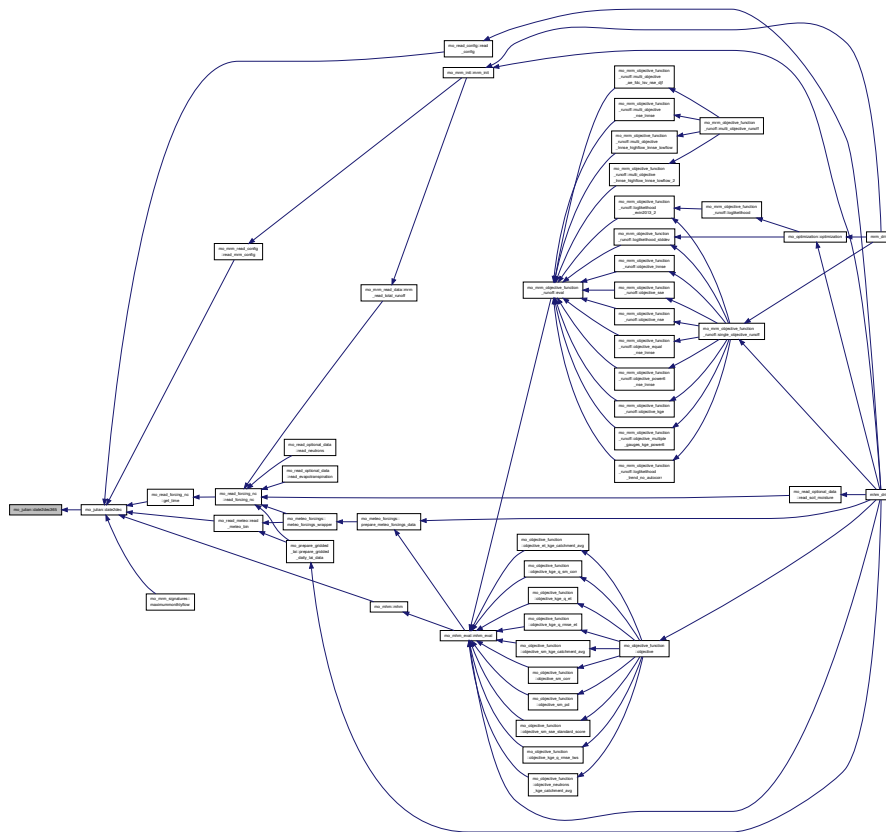
Written, David Schaefer

Date

Dec 2015

Referenced by `date2dec()`.

Here is the caller graph for this function:



15.14.2.8 date2decjulian()

```
elemental real(dp) function mo_julian::date2decjulian (
    integer(i4), intent(in), optional dd,
    integer(i4), intent(in), optional mm,
    integer(i4), intent(in), optional yy,
    integer(i4), intent(in), optional hh,
    integer(i4), intent(in), optional nn,
    integer(i4), intent(in), optional ss ) [private]
```

Fractional Julian day from day, month, year, hour, minute, second.

In this routine date2decJulian returns the fractional Julian Day that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day is 01.01.-4712 at noon, i.e. the 1st January 4713 BC 12:00:00 h. Julian day definition starts at noon of the 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc. This means that Julian day definition starts as 01.01.-4712 in astronomical units.

Parameters

in	<i>integer(i4), optional :: dd</i>	Day in month of Julian day (default: 1)
in	<i>integer(i4), optional :: mm</i>	Month in year of Julian day (default: 1)
in	<i>integer(i4), optional :: yy</i>	Year of Julian day (default: 1)
in	<i>integer(i4), optional :: hh</i>	Hours of Julian day (default: 0)

Parameters

in	<i>integer(i4), optional :: nn</i>	Minutes of hour of Julian day (default: 0)
in	<i>integer(i4), optional :: ss</i>	Seconds of minute of hour of Julian day (default: 0)

Returns

`real(dp) :: date2dec` — Fractional Julian day

Note

Julian day definition starts at noon of the 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc.

This means that Julian day definition starts as 01.01.-4712 in astronomical units.

`juldayJulian` and `caldatJulian` start at midnight of the 1st January 4713 BC. So `date2decJulian` and `juldayJulian` as well as `dec2dateJulian` and `caldatJulian` are shifted by half a day.

Use `date2decJulian` with `dec2dateJulian` together for fractional Julian dates and use `juldayJulian` with `caldatJulian` together for integer Julian days.

Author

Written, Matthias Cuntz

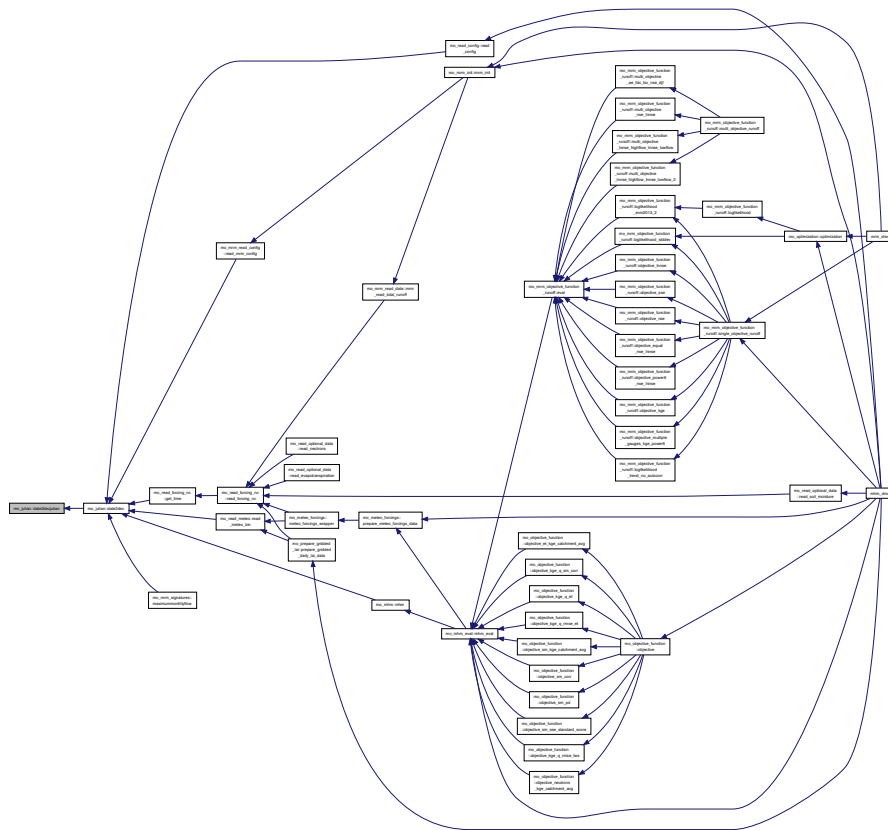
Date

Jan 2013 Modified Matthias Cuntz, May 2014 - changed to new algorithm with astronomical units removed
numerical recipes David Schaefer, Jan 2016 - renamed procedure

References `mo_kind::i8`.

Referenced by `date2dec()`.

Here is the caller graph for this function:



15.14.2.9 dec2date()

```

elemental subroutine, public mo_julian::dec2date (
    real(dp), intent(in) julian,
    integer(i4), intent(out), optional dd,
    integer(i4), intent(out), optional mm,
    integer(i4), intent(out), optional yy,
    integer(i4), intent(out), optional hh,
    integer(i4), intent(out), optional nn,
    integer(i4), intent(out), optional ss,
    integer(i4), intent(in), optional calendar )

```

Day, month, year, hour, minute, and second from fractional Julian day in the current or given calendar.

Wrapper around the calendar specific dec2date procedures. Inverse of the function date2dec. Here dec2date is input as a fractional Julian Day. The routine outputs dd, mm, yy, hh, nn, ss as the day, month, year, hour, minute, and second on which the specified Julian Day started at noon. The zeroth Julian Day depends on the called procedure. See their documentation for details.

Parameters

in	<i>real(dp) :: fJulian</i>	fractional Julian day
in	<i>integer(i4) :: calendar</i>	The calendar to use, the global calendar will be used by default
out	<i>integer(i4), optional :: dd</i>	Day in month of Julian day
out	<i>integer(i4), optional :: mm</i>	Month in year of Julian day

Parameters

out	<i>integer(i4), optional :: yy</i>	Year of Julian day
out	<i>integer(i4), optional :: hh</i>	Hour of Julian day
out	<i>integer(i4), optional :: nn</i>	Minute in hour of Julian day
out	<i>integer(i4), optional :: ss</i>	Second in minute of hour of Julian day

Author

Written, David Schaefer

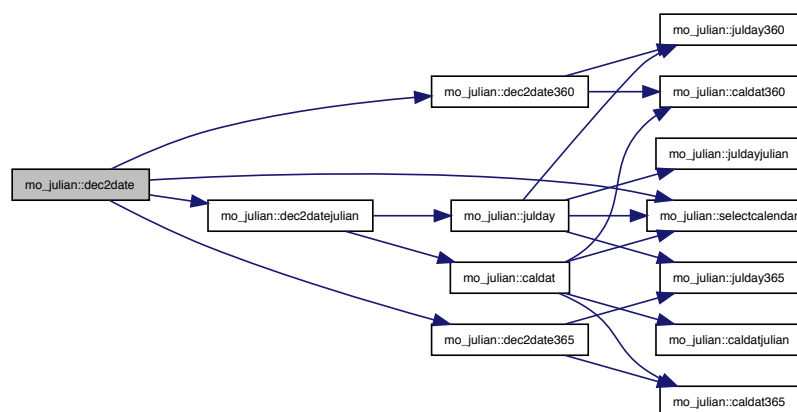
Date

Jan 2015

References `dec2date360()`, `dec2date365()`, `dec2datejulian()`, and `selectcalendar()`.

Referenced by `mo_mrm_write_fluxes_states::createoutputfile()`, `mo_write_fluxes_states::createoutputfile()`, `mo↔_temporal_aggregation::day2mon_average_dp()`, `mo_mrm_signatures::maximummonthlyflow()`, `mo_mhm::mhm()`, `mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf()`, `mo_read_config::read_config()`, `mo↔mrm_read_config::read_mrm_config()`, and `mo_mrm_write::write_daily_obs_sim_discharge()`.

Here is the call graph for this function:



```

elemental subroutine mo_julian::dec2date360 (
    real(dp), intent(in) julian,
    integer(i4), intent(out), optional dd,
    integer(i4), intent(out), optional mm,
    integer(i4), intent(out), optional yy,
    integer(i4), intent(out), optional hh,
    integer(i4), intent(out), optional nn,
    integer(i4), intent(out), optional ss ) [private]

```

Inverse of the function date2dec360. Here dec2date360 is input as a fractional Julian Day. The routine outputs dd, mm, yy, hh, nn, ss as the day, month, year, hour, minute, and second on which the specified Julian Day started at noon. The zeroth Julian Day is 01.01.0000 at noon.

in	<i>real(dp) :: fJulian</i>	fractional Julian day
out	<i>integer(i4), optional :: dd</i>	Day in month of Julian day

Parameters

out	<i>integer(i4), optional :: mm</i>	Month in year of Julian day
out	<i>integer(i4), optional :: yy</i>	Year of Julian day
out	<i>integer(i4), optional :: hh</i>	Hour of Julian day
out	<i>integer(i4), optional :: nn</i>	Minute in hour of Julian day
out	<i>integer(i4), optional :: ss</i>	Second in minute of hour of Julian day

Author

Written, David Schaefer

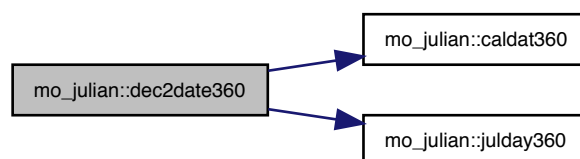
Date

Oct 2015

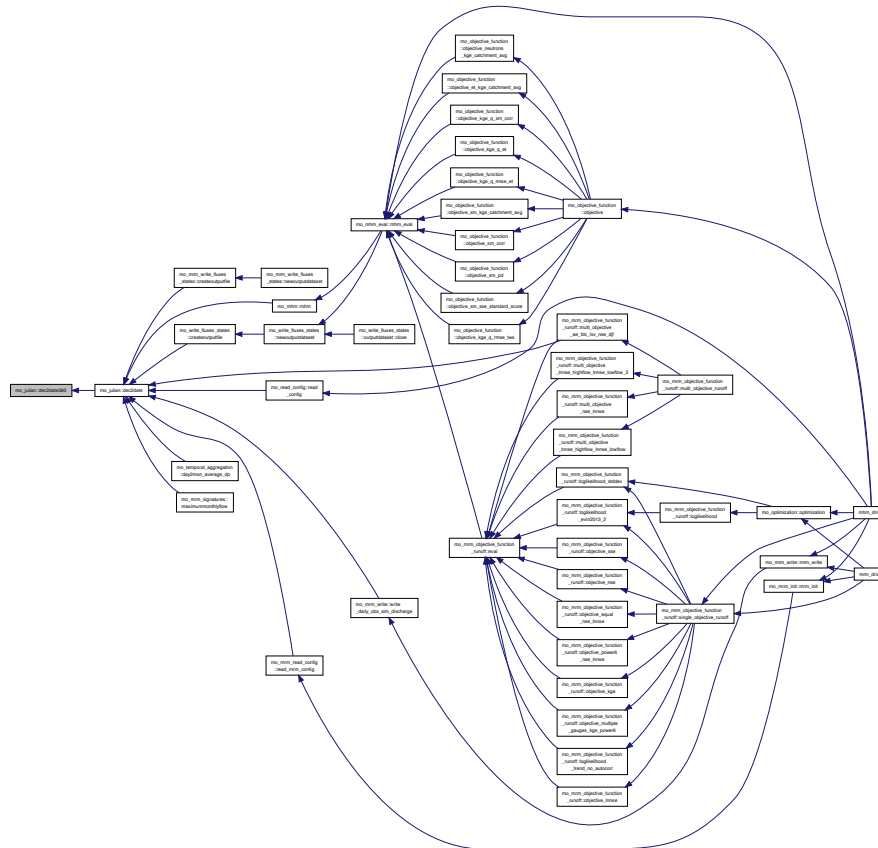
References `caldat360()`, `mo_kind::i4`, and `julday360()`.

Referenced by `dec2date()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.2.11 dec2date365()

```

elemental subroutine mo_julian::dec2date365 (
    real(dp), intent(in) julian,
    integer(i4), intent(out), optional dd,
    integer(i4), intent(out), optional mm,
    integer(i4), intent(out), optional yy,
    integer(i4), intent(out), optional hh,
    integer(i4), intent(out), optional nn,
    integer(i4), intent(out), optional ss ) [private]

```

Day, month, year, hour, minute, and second from fractional Julian day in a 365_day calendar.

Inverse of the function date2dec. Here dec2date365 is input as a fractional Julian Day. The routine outputs dd, mm, yy, hh, nn, ss as the day, month, year, hour, minute, and second on which the specified Julian Day started at noon. The zeroth Julian Day is 01.01.0000 at noon.

Parameters

in	<i>real(dp) :: fJulian</i>	fractional Julian day
out	<i>integer(i4), optional :: dd</i>	Day in month of Julian day
out	<i>integer(i4), optional :: mm</i>	Month in year of Julian day
out	<i>integer(i4), optional :: yy</i>	Year of Julian day
out	<i>integer(i4), optional :: hh</i>	Hour of Julian day

Parameters

out	<i>integer(i4), optional :: nn</i>	Minute in hour of Julian day
out	<i>integer(i4), optional :: ss</i>	Second in minute of hour of Julian day

Author

Written, David Schaefer

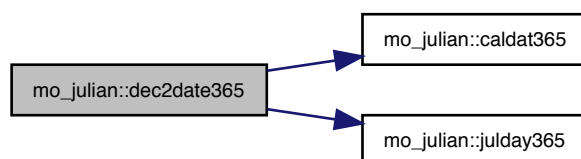
Date

Dec 2015

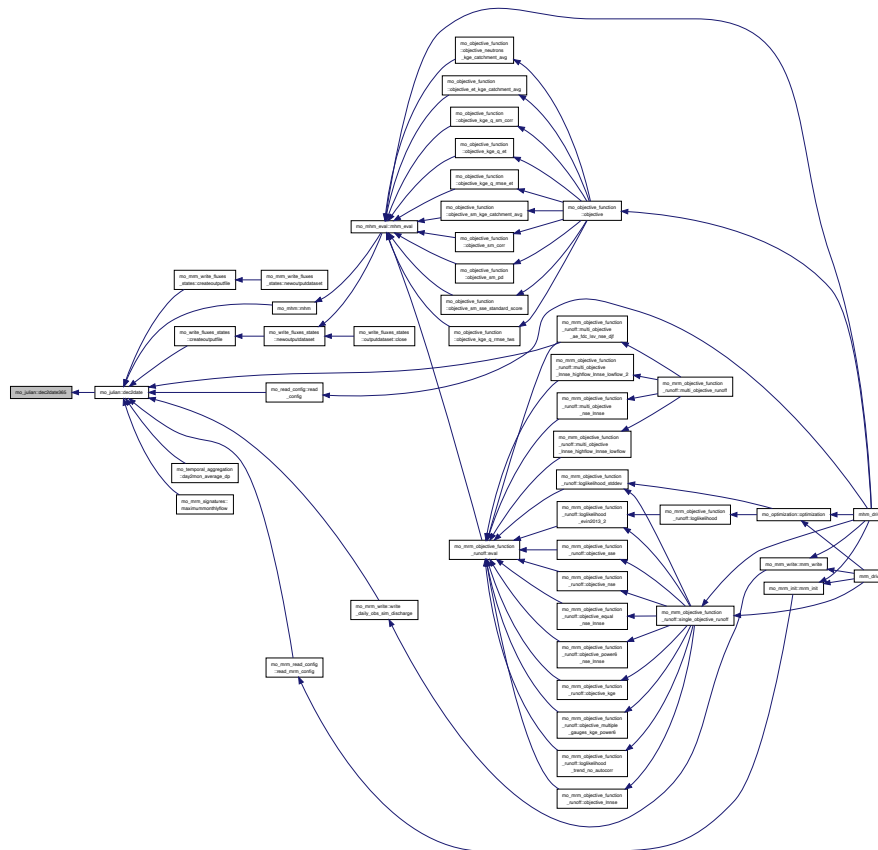
References `caldat365()`, `mo_kind::i4`, and `julday365()`.

Referenced by `dec2date()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.14.2.12 dec2datejulian()

```

elemental subroutine mo_julian::dec2datejulian (
    real(dp), intent(in) julian,
    integer(i4), intent(out), optional dd,
    integer(i4), intent(out), optional mm,
    integer(i4), intent(out), optional yy,
    integer(i4), intent(out), optional hh,
    integer(i4), intent(out), optional nn,
    integer(i4), intent(out), optional ss ) [private]

```

Day, month, year, hour, minute, and second from fractional Julian day.

Inverse of the function date2decJulian. Here dec2dateJulian is input as a fractional Julian Day, which starts at noon of the 1st January 4713 BC, i.e. 01.01.-4712. The routine outputs dd, mm, yy, hh, nn, ss as the day, month, year, hour, minute, and second on which the specified Julian Day started at noon. The zeroth Julian Day is 01.01.-4712 at noon, i.e. the 1st January 4713 BC at noon. Julian day definition starts at 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc. This means that Julian day definition starts as 01.01.-4712 in astronomical units.

Parameters

in	<i>real(dp) :: julian</i>	fractional Julian day
----	---------------------------	-----------------------

Parameters

out	<i>integer(i4), optional :: dd</i>	Day in month of Julian day
out	<i>integer(i4), optional :: mm</i>	Month in year of Julian day
out	<i>integer(i4), optional :: yy</i>	Year of Julian day
out	<i>integer(i4), optional :: hh</i>	Hour of Julian day
out	<i>integer(i4), optional :: nn</i>	Minute in hour of Julian day
out	<i>integer(i4), optional :: ss</i>	Second in minute of hour of Julian day

Note

Julian day definition starts at noon of the 1st January 4713 BC.

Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc.

This means that Julian day definition starts as 01.01.-4712 in astronomical units.

juldayJulian and caldatJulian start at midnight of the 1st January 4713 BC. So date2decJulian and juldayJulian as well as dec2dateJulian and caldatJulian are shifted by half a day.

Use date2decJulian with dec2dateJulian together for fractional Julian dates and use juldayJulian with caldatJulian together for integer Julian days.

Author

Written, Matthias Cuntz

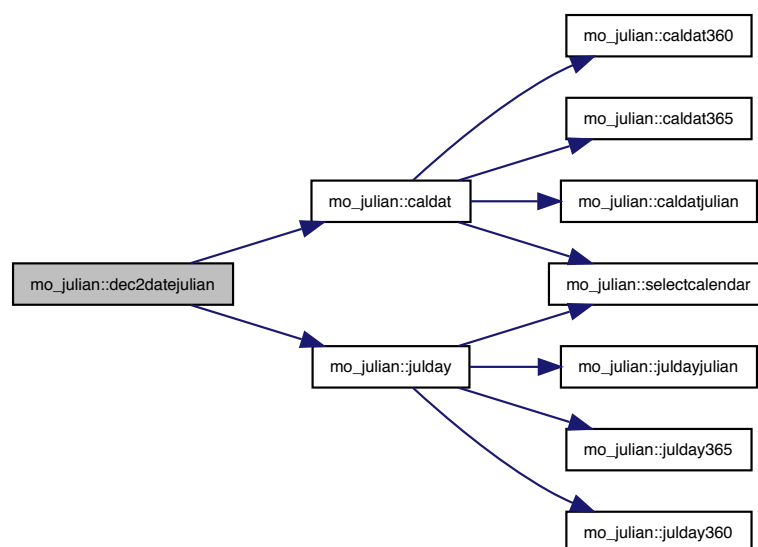
Date

Jan 2013 Modified Matthias Cuntz, May 2014 - changed to new algorithm with astronomical units removed
numerical recipes David Schaefer, Jan 2016 - renamed procedure

References caldat(), mo_kind::i4, mo_kind::i8, and julday().

Referenced by dec2date().

Here is the call graph for this function:



[illegible]

```

elemental integer(i4) function, public mo_julian::julday (
    integer(i4), intent(in) dd,
    integer(i4), intent(in) mm,
    integer(i4), intent(in) yy,
    integer(i4), intent(in), optional calendar )

```

Wrapper around the calendar specific julday procedures. In this routine julday returns the Julian Day Number that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day depends on the called procedure. See their documentation for details.

<code>in</code>	<code>integer(i4) :: dd</code>	Day in month of Julian day
<code>in</code>	<code>integer(i4) :: mm</code>	Month in year of Julian day
<code>in</code>	<code>integer(i4) :: yy</code>	Year of Julian day
<code>in</code>	<code>integer(i4), optional :: calendar</code>	The calendar to use, the global calendar will be used by default

Returns

integer(i4) :: julian ! Julian day

Author

Written, David Schaefer

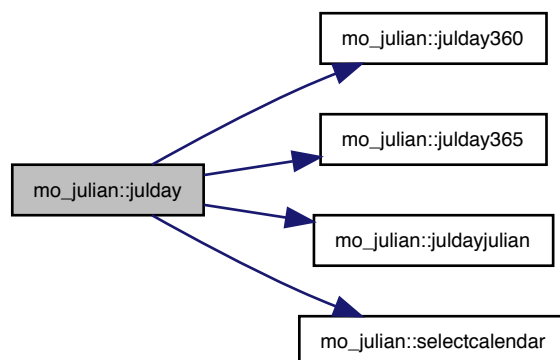
Date

Jan 2015

References `julday360()`, `julday365()`, `juldayjulian()`, and `selectcalendar()`.

Referenced by `mo_meteo_forcings::chunk_size()`, `mo_temporal_aggregation::day2mon_average_dp()`, `dec2datejulian()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_eval::mrm_eval()`, `ndays()`, `mo_read_forcing_nc::read_forcing_nc()`, and `mo_read_timeseries::read_timeseries()`.

Here is the call graph for this function:



The diagram illustrates the dependency structure of the 'mo' module. It features a central node 'mo_julian_data' which is the primary data source for many other components. To its right, there are several configuration and utility nodes, including 'mo_read_config/read_config', 'mo_read_optional_data/read_optional_data', and 'mo_read_neutrons'. These nodes are further connected to a large number of specialized function nodes, such as 'mo_read_neutrons/read_neutrons', 'mo_read_optional_data/read_optional_data', and 'mo_read_neutrons/read_neutrons'. The graph is organized into layers, with 'mo_julian_data' at the top and 'mo_read_neutrons' at the bottom. Arrows indicate the flow of dependencies from right to left.

```

elemental integer(i4) function mo_julian::julday360 (
    integer(i4), intent(in) dd,
    integer(i4), intent(in) mm,
    integer(i4), intent(in) yy ) [private]

```

In this routine julday360 returns the Julian Day Number that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day is 01.01.0000

in	<i>integer(i4) :: dd</i>	Day in month of Julian day
in	<i>integer(i4) :: mm</i>	Month in year of Julian day
in	<i>integer(i4) :: yy</i>	Year of Julian day

Returns

integer(i4) :: julian ! Julian day

Author

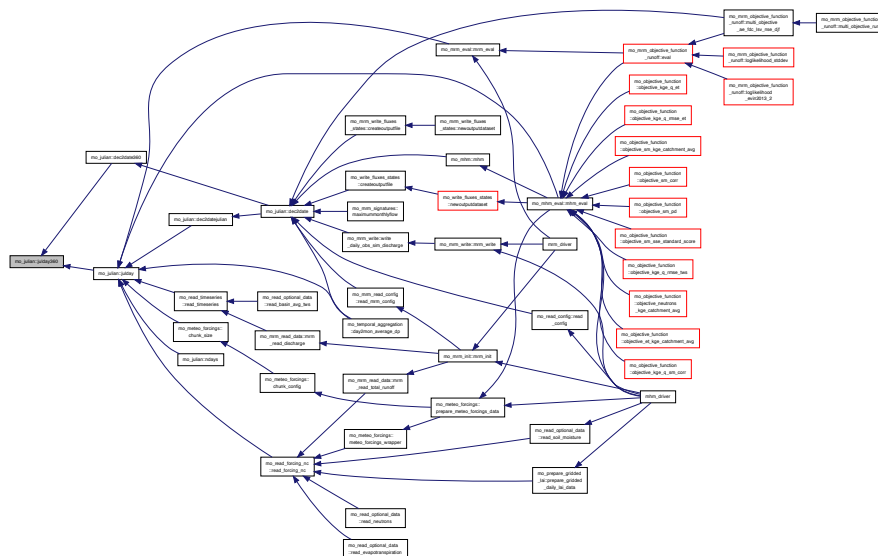
Written, David Schaefer

Date

Oct 2015

Referenced by dec2date360(), and julday().

Here is the caller graph for this function:

**15.14.2.15 julday365()**

```
elemental integer(i4) function mo_julian::julday365 (
    integer(i4), intent(in) dd,
    integer(i4), intent(in) mm,
    integer(i4), intent(in) yy ) [private]
```

Julian day from day, month and year in a 365_day calendar.

In this routine julday365 returns the Julian Day Number that begins at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. The zeroth Julian Day is 01.01.0000

Parameters

in	<i>integer(i4) :: dd</i>	Day in month of Julian day
in	<i>integer(i4) :: mm</i>	Month in year of Julian day
in	<i>integer(i4) :: yy</i>	Year of Julian day

Returns

```
integer(i4) :: julian ! Julian day
```

Author

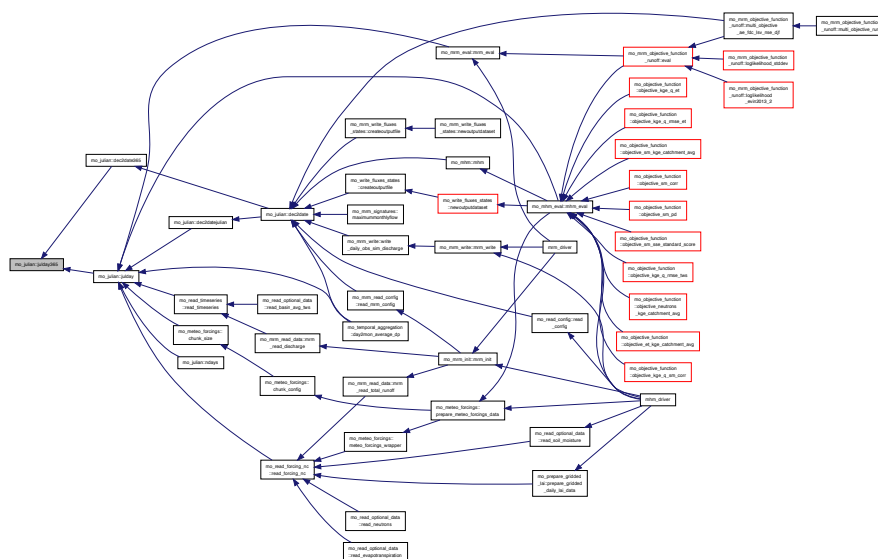
Written, David Schaefer

Date _____

Dec 2015

Referenced by `dec2date365()`, and `julday()`.

Here is the caller graph for this function:



15.14.2.16 juldayjulian()

```

elemental integer(i4) function mo_julian::juldayjulian (
    integer(i4), intent(in) dd,
    integer(i4), intent(in) mm,
    integer(i4), intent(in) yy ) [private]

```

Julian day from day, month and year.

In this routine `juldayJulian` returns the Julian Day Number that begins at noon of the calendar date specified by month `mm`, day `dd`, and year `yy`, all integer variables. The zeroth Julian Day is 01.01.-4712 at noon, i.e. the 1st January 4713 BC 12:00:00 h. Julian day definition starts at noon of the 1st January 4713 BC. Here, the astronomical definition is used, i.e. the year 1 BC (historic) is counted as 0 (astronomic), 2 BC is -1, etc. This means that Julian day definition starts as 01.01.-4712 in astronomical units.

Parameters

<code>in</code>	<code>integer(i4) :: dd</code>	Day in month of Julian day
<code>in</code>	<code>integer(i4) :: mm</code>	Month in year of Julian day
<code>in</code>	<code>integer(i4) :: yy</code>	Year of Julian day

15.14.2.17 ndays()

```

elemental integer(i4) function, public mo_julian::ndays (
    integer(i4), intent(in) dd,
    integer(i4), intent(in) mm,
    integer(i4), intent(in) yy )

```

IMSL Julian day from day, month and year.

In this routine ndays returns the IMSL Julian Day Number. Julian days begin at noon of the calendar date specified by month mm, day dd, and year yy, all integer variables. IMSL treats 01.01.1900 as a reference and assigns a Julian day 0 to it. $ndays = julday(dd,mm,yy) - julday(01,01,1900)$

Parameters

in	<i>integer(i4) :: dd</i>	Day in month of IMSL Julian day
in	<i>integer(i4) :: mm</i>	Month in year of IMSL Julian day
in	<i>integer(i4) :: yy</i>	Year of IMSL Julian day

Returns

integer(i4) :: julian — IMSL Julian day, i.e. days before or after 01.01.1900

Author

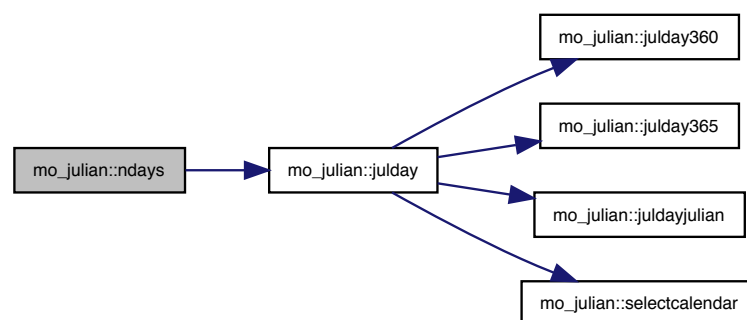
Written, Matthias Cuntz

Date

Dec 2011

References `julday()`.

Here is the call graph for this function:



15.14.2.18 ndyin()

```

elemental subroutine, public mo_julian::ndyin (
    integer(i4), intent(in) julian,

```

```
integer(i4), intent(out) dd,
integer(i4), intent(out) mm,
integer(i4), intent(out) yy )
```

Day, month and year from IMSL Julian day.

Inverse of the function ndys. Here ISML Julian is input as a Julian Day Number minus the Julian Day Number of 01.01.1900, and the routine outputs id, mm, and yy as the day, month, and year on which the specified Julian Day started at noon. ndyin is caldat(IMSLJulian + 2415021, dd, mm, yy)

Parameters

in	<i>integer(i4) :: julian</i>	IMSL Julian day, i.e. days before or after 01.01.1900
out	<i>integer(i4) :: dd</i>	Day in month of IMSL Julian day
out	<i>integer(i4) :: mm</i>	Month in year of IMSL Julian day
out	<i>integer(i4) :: yy</i>	Year of IMSL Julian day

Author

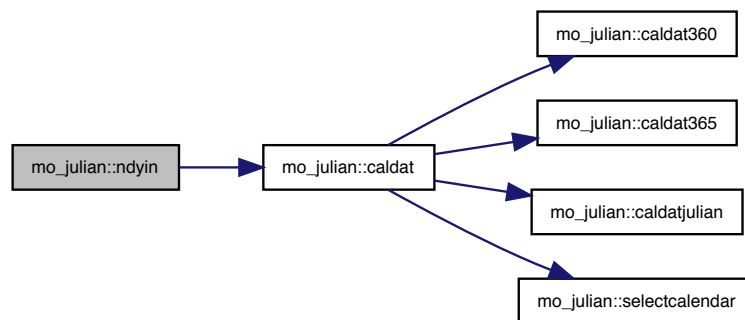
Written, Matthias Cuntz

Date

Dec 2011

References caldat().

Here is the call graph for this function:



15.14.2.19 selectcalendar()

```
pure integer(i4) function mo_julian::selectcalendar (
    integer(i4), intent(in), optional selector ) [private]
```

Select a calendar.

Returns a valid calendar index, based on the given optional argument and/or the module global private variable calendar. If an invalid selector is passed, its value is ignored and the global calendar value returned instead.

Parameters

in	<i>integer(i4), optional :: selector</i>	Calendar selector {1 2 3}
----	--	---------------------------

Author

Written, David Schaefer

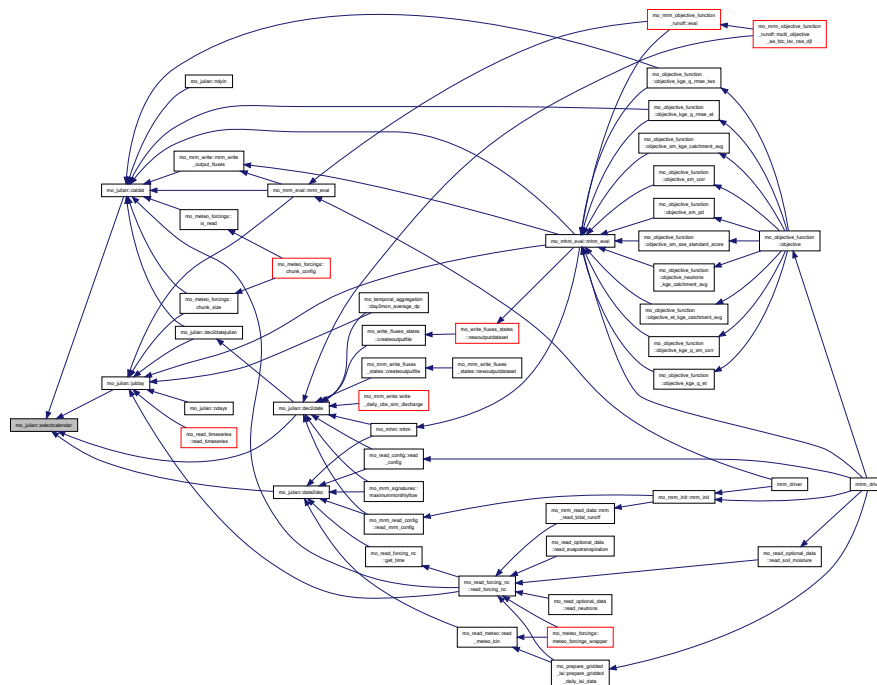
Date

Jan 2015

References calendar.

Referenced by caldat(), date2dec(), dec2date(), and julday().

Here is the caller graph for this function:



15.14.2.20 setcalendarinteger()

```
subroutine mo_julian::setcalendarinteger (
    integer(i4), intent(in) selector ) [private]
```

Set module private variable calendar.

Parameters

in	<i>integer(i4) :: selector</i>	{1 2 3}
----	--------------------------------	---------

Author

Written, David Schaefer

Date

Jan 2015

References calendar.

Referenced by setcalendarstring().

Here is the caller graph for this function:

**15.14.2.21 setcalendarstring()**

```

subroutine mo_julian::setcalendarstring (
    character(*), intent(in) selector ) [private]
  
```

Set module private variable calendar.

Parameters

in	<i>character(len=*) :: selector</i>	{"julian" "365day" "360day"}
----	-------------------------------------	------------------------------

Author

Written, David Schaefer

Date

Jan 2015

References setcalendarinteger().

Here is the call graph for this function:



15.14.3 Variable Documentation

15.14.3.1 calendar

```
integer(i4), save, private mo_julian::calendar = 1 [private]
```

Referenced by selectcalendar(), and setcalendarinteger().

15.15 mo_kind Module Reference

Define number representations.

Data Types

- type `sprs2_dp`
Double Precision Numerical Recipes types for sparse arrays.
- type `sprs2_sp`
Single Precision Numerical Recipes types for sparse arrays.

Variables

- integer, parameter `i1` = `SELECTED_INT_KIND(2)`
1 Byte Integer Kind
- integer, parameter `i2` = `c_short`
2 Byte Integer Kind
- integer, parameter `i4` = `c_int`
4 Byte Integer Kind
- integer, parameter `i8` = `c_long_long`
8 Byte Integer Kind
- integer, parameter `sp` = `c_float`
Single Precision Real Kind.
- integer, parameter `dp` = `c_double`
Double Precision Real Kind.
- integer, parameter `spc` = `c_float_complex`
Single Precision Complex Kind.
- integer, parameter `dpc` = `c_double_complex`
Double Precision Complex Kind.
- integer, parameter `lgt` = `KIND(.true.)`
Logical Kind.

15.15.1 Detailed Description

Define number representations.

This module declares the desired ranges and precisions of the number representations, such as single precision or double precision, 32-bit or 46-bit integer, etc. It confirms mostly with the `nrtype` module of Numerical Recipes in f90.

Authors

Juliane Mai, Matthias Cuntz, Nov 2011

Date

2011-2014

Copyright

GNU Lesser General Public License <http://www.gnu.org/licenses/>

15.15.2 Variable Documentation**15.15.2.1 dp**

```
integer, parameter mo_kind::dp = c_double
```

Double Precision Real Kind.

Referenced by mo_mrm_routing::add_inflow(), mo_mrm_tools::calculate_grid_properties(), mo_sce::cce(), mo_sce::chkcst(), mo_sce::comp(), mo_anneal::dchange_dp(), mo_dds::dds(), mo_mrm_tools::get_basin_info_mrm(), mo_read_forcing_nc::get_time(), mo_sce::getpnt(), mo_anneal::gettemperature_dp(), mo_dds::mdds(), mhm_driver(), mrm_driver(), mo_mrm_eval::mrm_eval(), mo_mrm_init::mrm_init(), mo_mrm_init::mrm_init_param(), mo_mrm_restart::mrm_read_restart_config(), mo_mrm_init::mrm_update_param(), mo_mrm_write::mrm_write_output_fluxes(), mo_dds::neigh_value(), mo_optimization::optimization(), mo_anneal::pargen_anneal_dp(), mo_mcmc::pargen_dp(), mo_mcmc::pargennorm_dp(), mo_sce::parstt(), mo_spatialsimilarity::pd_dp(), mo_read_forcing_nc::read_forcing_nc(), mo_read_meteo::read_meteo_bin(), mo_restart::read_restart_config(), mo_restart::read_restart_states(), mo_soil_database::read_soil_lut(), mo_read_forcing_nc::read_weights_nc(), mo_sce::sce(), mo_sce::sort_matrix(), mo_mrm_init::variables_alloc_routing(), mo_write_ascii::write_configfile(), mo_mrm_write::write_configfile(), and mo_restart::write_restart_files().

15.15.2.2 dpc

```
integer, parameter mo_kind::dpc = c_double_complex
```

Double Precision Complex Kind.

Referenced by mo_corr::four1_dp(), mo_corr::four1_sp(), mo_corr::fourrow_dp(), mo_corr::fourrow_sp(), mo_corr::realft_dp(), and mo_corr::zroots_unity_dp().

15.15.2.3 i1

```
integer, parameter mo_kind::i1 = SELECTED_INT_KIND(2)
```

1 Byte Integer Kind

15.15.2.4 i2

```
integer, parameter mo_kind::i2 = c_short
```

2 Byte Integer Kind

15.15.2.5 i4

```
integer, parameter mo_kind::i4 = c_int
```

4 Byte Integer Kind

Referenced by mo_mrm_routing::add_inflow(), mo_mrm_tools::calculate_grid_properties(), mo_julian::caldatjulian(), mo_sce::cce(), mo_sce::chkcst(), mo_meteo_forcings::chunk_config(), mo_meteo_forcings::chunk_size(), mo_sce::comp(), mo_string_utils::compress(), mo_mrm_init::config_output(), mo_anneal::dchange_dp(), mo_dds::dds(), mo_julian::dec2date360(), mo_julian::dec2date365(), mo_julian::dec2datejulian(), mo_mrm_tools::get_basin_info_mrm(), mo_read_forcing_nc::get_time(), mo_sce::getpnt(), mo_anneal::gettemperature_dp(), mo_startup::initialise(), mo_meteo_forcings::is_read(), mo_multi_param_reg::karstic_layer(), mo_mrm_init::l0_check_input_routing(), mo_mrm_net_startup::l11_flow_direction(), mo_mrm_net_startup::l11_variable_init(), mo_startup::l1_variable_init(), mo_neutrons::lookupintegral(), mo_dds::mdds(), mhm_driver(), mo_mhm_eval::mhm_eval(), mo_multi_param_reg::mpr(), mo_mpr_runoff::mpr_runoff(), mo_mpr_soilmoist::mpr_sm(), mrm_driver(), mo_mrm_eval::mrm_eval(), mo_mrm_init::mrm_init(), mo_mrm_init::mrm_init_param(), mo_mrm_read_data::mrm_l1_variable_init(), mo_mrm_restart::mrm_read_restart_config(), mo_mrm_init::mrm_update_param(), mo_mrm_write::mrm_write_output_fluxes(), mo_optimization::optimization(), mo_anneal::pargen_anneal_dp(), mo_sce::parstt(), mo_percentile::percentile_0d_dp(), mo_percentile::percentile_0d_sp(), mo_percentile::percentile_1d_dp(), mo_percentile::percentile_1d_sp(), mo_mpr_pet::pet_correctbyasp(), mo_mrm_init::print_startup_message(), mo_read_config::read_config(), mo_read_forcing_nc::read_forcing_nc(), mo_read_lut::read_lai_lut(), mo_read_meteo::read_meteo_bin(), mo_mrm_read_config::read_mrm_config(), mo_restart::read_restart_config(), mo_restart::read_restart_states(), mo_read_spatial_data::read_spatial_data_ascii_i4(), mo_read_forcing_nc::read_weights_nc(), mo_sce::sce(), mo_sce::sort_matrix(), mo_upscaling_operators::upscale_arithmetic_mean(), mo_upscaling_operators::upscale_harmonic_mean(), mo_mrm_init::variables_alloc_routing(), mo_write_ascii::write_configfile(), mo_mrm_write::write_configfile(), and mo_restart::write_restart_files().

15.15.2.6 i8

```
integer, parameter mo_kind::i8 = c_long_long
```

8 Byte Integer Kind

Referenced by mo_julian::caldatjulian(), mo_sce::cce(), mo_julian::date2decjulian(), mo_anneal::dchange_dp(), mo_dds::dds(), mo_julian::dec2datejulian(), mo_xor4096::get_timeseed_i8_0d(), mo_xor4096::get_timeseed_i8_1d(), mo_sce::getpnt(), mo_anneal::gettemperature_dp(), mo_julian::juldayjulian(), mo_dds::mdds(), mo_dds::neigh_value(), mo_optimization::optimization(), and mo_sce::sce().

15.15.2.7 lgt

```
integer, parameter mo_kind::lgt = KIND(.true.)
```

Logical Kind.

15.15.2.8 sp

```
integer, parameter mo_kind::sp = c_float
```

Single Precision Real Kind.

Referenced by mo_spatialsimilarity::pd_sp(), and mo_read_meteo::read_meteo_bin().

15.15.2.9 spc

integer, parameter mo_kind::spc = c_float_complex

Single Precision Complex Kind.

Referenced by mo_corr::four1_sp(), mo_corr::fourrow_sp(), mo_corr::realft_sp(), and mo_corr::zroots_unity_sp().

15.16 mo_linfit Module Reference

Fitting a straight line.

Data Types

- interface [linfit](#)
Fits a straight line to input data by minimizing χ^2 .

Functions/Subroutines

- real(dp) function, dimension(:), allocatable [linfit_dp](#) (x, y, a, b, siga, sigb, chi2, model2)
- real(sp) function, dimension(:), allocatable [linfit_sp](#) (x, y, a, b, siga, sigb, chi2, model2)

15.16.1 Detailed Description

Fitting a straight line.

This module provides a routine to fit a straight line with model I or model II regression.

Authors

Matthias Cuntz

Date

Mar 2011

15.16.2 Function/Subroutine Documentation

15.16.2.1 linfit_dp()

```
real(dp) function, dimension(:), allocatable mo_linfit::linfit_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    real(dp), intent(out), optional a,
    real(dp), intent(out), optional b,
    real(dp), intent(out), optional siga,
    real(dp), intent(out), optional sigb,
    real(dp), intent(out), optional chi2,
    logical, intent(in), optional model2 ) [private]
```

15.16.2.2 linfit_sp()

```

real(sp) function, dimension(:), allocatable mo_linfit::linfit_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    real(sp), intent(out), optional a,
    real(sp), intent(out), optional b,
    real(sp), intent(out), optional siga,
    real(sp), intent(out), optional sigb,
    real(sp), intent(out), optional chi2,
    logical, intent(in), optional model2 ) [private]

```

15.17 mo_mcmc Module Reference

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution.

Data Types

- interface [mcmc](#)

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are either known or modeled).

- interface [mcmc_stddev](#)

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are neither known nor modeled).

Functions/Subroutines

- subroutine [mcmc_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, restart, restart_file, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)
- subroutine [mcmc_stddev_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)
- real(dp) function [pargen_dp](#) (old, dMax, oMin, oMax, RN, inbound)
- real(dp) function [pargennorm_dp](#) (old, dMax, oMin, oMax, RN, inbound)
- recursive subroutine [generatenewparameterset_dp](#) (ParaSelectMode, paraold, truepara, rangePar, stepsize, save_state_2, save_state_3, paranew, ChangePara)

15.17.1 Detailed Description

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution.

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution.

Authors

Maren Goehler, Juliane Mai

Date

Aug 2012

15.17.2 Function/Subroutine Documentation

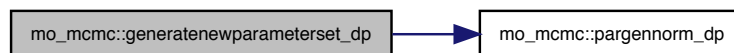
15.17.2.1 generatenewparameterset_dp()

```
recursive subroutine mo_mcmc::generatenewparameterset_dp (
    integer(i4), intent(in) ParaSelectMode,
    real(dp), dimension(:), intent(in) paraold,
    integer(i4), dimension(:), intent(in) truepara,
    real(dp), dimension(:, :), intent(in) rangePar,
    real(dp), dimension(:), intent(in) stepsize,
    integer(i8), dimension(n_save_state), intent(inout) save_state_2,
    integer(i8), dimension(n_save_state), intent(inout) save_state_3,
    real(dp), dimension(size(paraold)), intent(out) paranew,
    logical, dimension(size(paraold)), intent(out) ChangePara )
```

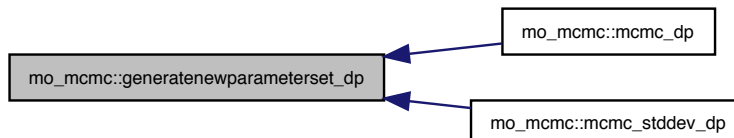
References pargennorm_dp().

Referenced by mcmc_dp(), and mcmc_stddev_dp().

Here is the call graph for this function:



Here is the caller graph for this function:



15.17.2.2 mcmc_dp()

```
subroutine mo_mcmc::mcmc_dp (
    likelihood,
    real(dp), dimension(:), intent(in) para,
    real(dp), dimension(:, :), intent(in) rangePar,
    real(dp), dimension(:, :), intent(out), allocatable mcmc_paras,
    real(dp), dimension(:, :), intent(out), allocatable burnin_paras,
    integer(i8), intent(in), optional seed_in,
    logical, intent(in), optional printflag_in,
    logical, dimension(size(para,1)), intent(in), optional maskpara_in,
    logical, intent(in), optional restart,
    character(len=*), intent(in), optional restart_file,
```

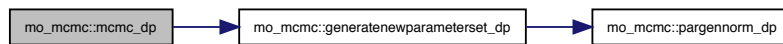
```

character(len=*), intent(in), optional tmp_file,
logical, intent(in), optional loglike_in,
integer(i4), intent(in), optional ParaSelectMode_in,
integer(i4), intent(in), optional iter_burnin_in,
integer(i4), intent(in), optional iter_mcmc_in,
integer(i4), intent(in), optional chains_in,
real(dp), dimension(size(para,1)), intent(in), optional stepsize_in ) [private]

```

References `generatenewparameterset_dp()`, and `mo_xor4096::n_save_state`.

Here is the call graph for this function:



15.17.2.3 mcmc_stddev_dp()

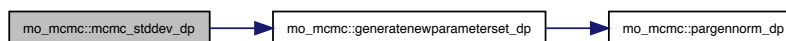
```

subroutine mo_mcmc::mcmc_stddev_dp (
    likelihood,
    real(dp), dimension(:), intent(in) para,
    real(dp), dimension(:, :), intent(in) rangePar,
    real(dp), dimension(:, :), intent(out), allocatable mcmc_paras,
    real(dp), dimension(:, :), intent(out), allocatable burnin_paras,
    integer(i8), intent(in), optional seed_in,
    logical, intent(in), optional printflag_in,
    logical, dimension(size(para,1)), intent(in), optional maskpara_in,
    character(len=*), intent(in), optional tmp_file,
    logical, intent(in), optional loglike_in,
    integer(i4), intent(in), optional ParaSelectMode_in,
    integer(i4), intent(in), optional iter_burnin_in,
    integer(i4), intent(in), optional iter_mcmc_in,
    integer(i4), intent(in), optional chains_in,
    real(dp), dimension(size(para,1)), intent(in), optional stepsize_in )

```

References `generatenewparameterset_dp()`, and `mo_xor4096::n_save_state`.

Here is the call graph for this function:



15.17.2.4 pargen_dp()

```

real(dp) function mo_mcmc::pargen_dp (
    real(dp), intent(in) old,

```

```

real(dp), intent(in) dMax,
real(dp), intent(in) oMin,
real(dp), intent(in) oMax,
real(dp), intent(in) RN,
logical, intent(out), optional inbound )

```

References mo_kind::dp.

15.17.2.5 pargennorm_dp()

```

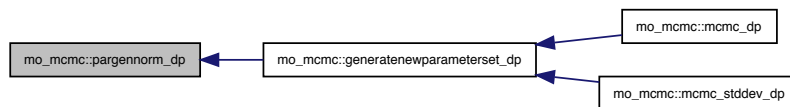
real(dp) function mo_mcmc::pargennorm_dp (
    real(dp), intent(in) old,
    real(dp), intent(in) dMax,
    real(dp), intent(in) oMin,
    real(dp), intent(in) oMax,
    real(dp), intent(in) RN,
    logical, intent(out), optional inbound )

```

References mo_kind::dp.

Referenced by generatenewparameterset_dp().

Here is the caller graph for this function:



15.18 mo_message Module Reference

Write out concatenated strings.

Functions/Subroutines

- subroutine, public [message](#) (t01, t02, t03, t04, t05, t06, t07, t08, t09, t10, uni, advance)
Write out several string concatenated either on screen or in a file.

Variables

- character(len=1024), public [message_text](#) = "

15.18.1 Detailed Description

Write out concatenated strings.

Write out several strings concatenated on standard out or a given unit, either advancing or not.

Author

Matthias Cuntz

Date

Jul 2011

15.18.2 Function/Subroutine Documentation**15.18.2.1 message()**

```

subroutine, public mo_message::message (
    character(len=*), intent(in), optional t01,
    character(len=*), intent(in), optional t02,
    character(len=*), intent(in), optional t03,
    character(len=*), intent(in), optional t04,
    character(len=*), intent(in), optional t05,
    character(len=*), intent(in), optional t06,
    character(len=*), intent(in), optional t07,
    character(len=*), intent(in), optional t08,
    character(len=*), intent(in), optional t09,
    character(len=*), intent(in), optional t10,
    integer, intent(in), optional uni,
    character(len=*), intent(in), optional advance )

```

Write out several string concatenated either on screen or in a file.

Parameters

in	<i>character(len=*), optional :: t01</i>	1st string
in	<i>character(len=*), optional :: t02</i>	2nd string
in	<i>character(len=*), optional :: t03</i>	3rd string
in	<i>character(len=*), optional :: t04</i>	4th string
in	<i>character(len=*), optional :: t05</i>	5th string
in	<i>character(len=*), optional :: t06</i>	6th string
in	<i>character(len=*), optional :: t07</i>	7th string
in	<i>character(len=*), optional :: t08</i>	8th string
in	<i>character(len=*), optional :: t09</i>	9th string
in	<i>character(len=*), optional :: t10</i>	10th string
in	<i>integer , optional :: unit</i>	Unit to write to (default: nout)
in	<i>character(len=*), optional :: advance</i>	WRITE advance keyword (default: 'yes') yes: newline will be written after message no: no newline at end of message

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

Date

Dec 2011

References `mo_constants::nout`.

Referenced by `mo_mrm_tools::calculate_grid_properties()`, `mo_init_states::calculate_grid_properties()`, `mo_multi_param_reg::canopy_intercept_param()`, `mo_read_wrapper::check_consistency_lut_map()`, `mo_meteo_forcings::chunk_size()`, `mo_mrm_write_fluxes_states::close()`, `mo_write_fluxes_states::close()`, `mo_mrm_init::config_output()`, `mo_startup::constants_init()`, `mo_mrm_objective_function_runoff::eval()`, `mo_objective_function::extract_basin_avg_tws()`, `mo_mrm_objective_function_runoff::extract_runoff()`, `mo_soil_database::generate_soil_database()`, `mo_init_states::get_basin_info()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_read_forcing_nc::get_time()`, `mo_meteo_forcings::is_read()`, `mo_startup::l0_check_input()`, `mo_mrm_init::l0_check_input_routing()`, `mo_mrm_net_startup::l11_flow_direction()`, `mo_startup::l2_variable_init()`, `mo_mrm_signatures::limb_densities()`, `mo_mrm_signatures::maximummonthlyflow()`, `mo_mhm::mhm()`, `mhm_driver()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_signatures::moments()`, `mo_multi_param_reg::mpr()`, `mo_mpr_soilmoist::mpr_sm()`, `mo_mpr_smhorizons::mpr_smhorizons()`, `mrm_driver()`, `mo_mrm_init::mrm_init()`, `mo_mrm_init::mrm_init_param()`, `mo_mrm_read_data::mrm_read_discharge()`, `mo_mrm_read_data::mrm_read_l0_data()`, `mo_mrm_restart::mrm_read_restart_config()`, `mo_mrm_write::mrm_write_optfile()`, `mo_mrm_write::mrm_write_optinamelist()`, `mo_mrm_restart::mrm_write_restart()`, `mo_objective_function::objective_et_kge_catchment_avg()`, `mo_objective_function::objective_kge_q_et()`, `mo_objective_function::objective_kge_q_rmse_et()`, `mo_objective_function::objective_kge_q_rmse_tws()`, `mo_objective_function::objective_kge_q_sm_corr()`, `mo_objective_function::objective_neutrons_kge_catchment_avg()`, `mo_objective_function::objective_sm_corr()`, `mo_objective_function::objective_sm_kge_catchment_avg()`, `mo_objective_function::objective_sm_pd()`, `mo_objective_function::objective_sse_standard_score()`, `mo_nml::open_nml()`, `mo_optimization::optimization()`, `mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data()`, `mo_meteo_forcings::prepare_meteo_forcings_data()`, `mo_mrm_init::print_startup_message()`, `mo_read_optional_data::read_basin_avg_tws()`, `mo_read_config::read_config()`, `mo_read_wrapper::read_data()`, `mo_read_optional_data::read_evapotranspiration()`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_mrm_read_latlon::read_latlon()`, `mo_read_latlon::read_latlon()`, `mo_read_meteo::read_meteo_bin()`, `mo_mrm_read_config::read_mrm_config()`, `mo_mrm_read_config::read_mrm_config_coupling()`, `mo_mrm_read_config::read_mrm_routing_params()`, `mo_read_optional_data::read_neutrons()`, `mo_restart::read_restart_config()`, `mo_soil_database::read_soil_lut()`, `mo_read_optional_data::read_soil_moisture()`, `mo_read_timeseries::read_timeseries()`, `mo_read_forcing_nc::read_weights_nc()`, `mo_mrm_signatures::runoffratio()`, `mo_write_ascii::write_configfile()`, `mo_mrm_write::write_configfile()`, `mo_mrm_write::write_daily_obs_sim_discharge()`, `mo_write_ascii::write_optfile()`, `mo_write_ascii::write_optinamelist()`, `mo_restart::write_restart_files()`, and `mo_mrm_signatures::zeroflowratio()`.

15.18.3 Variable Documentation

15.18.3.1 message_text

```
character(len=1024), public mo_message::message_text = ''
```

Referenced by `mo_startup::l0_check_input()`, `mo_mrm_init::l0_check_input_routing()`, `mhm_driver()`, `mo_nml::open_nml()`, `mo_nml::position_nml()`, and `mo_mrm_init::print_startup_message()`.

15.19 mo_meteo_forcings Module Reference

Prepare meteorological forcings data for mHM.

Functions/Subroutines

- subroutine, public [prepare_meteo_forcings_data](#) (iBasin, tt)
Prepare meteorological forcings data for a given variable.

- subroutine [meteo_forcings_wrapper](#) (iBasin, dataPath, inputFormat, dataOut1, lower, upper, ncvarName)

Prepare meteorological forcings data for mHM at Level-1.

- subroutine [meteo_weights_wrapper](#) (iBasin, read_meteo_weights, dataPath, dataOut1, lower, upper, ncvarName)

Prepare weights for meteorological forcings data for mHM at Level-1.

- subroutine [chunk_config](#) (iBasin, tt, read_flag, readPer)

- logical function [is_read](#) (iBasin, tt)

evaluate whether new chunk should be read at this timestep

- subroutine [chunk_size](#) (iBasin, tt, readPer)

calculate beginning and end of read Period, i.e. that is length of current chunk to read

15.19.1 Detailed Description

Prepare meteorological forcings data for mHM.

Prepare meteorological forcings data for mHM.

Authors

Rohini Kumar

Date

Jan 2012

15.19.2 Function/Subroutine Documentation

15.19.2.1 [chunk_config\(\)](#)

```
subroutine mo_meteo_forcings::chunk_config (
    integer(i4), intent(in) iBasin,
    integer(i4), intent(in) tt,
    logical, intent(out) read_flag,
    type(period), intent(out) readPer )
```

References [chunk_size\(\)](#), [mo_kind::i4](#), [is_read\(\)](#), and [mo_mhm_constants::nodata_dp](#).

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


```

integer(i4), intent(in) iBasin,
integer(i4), intent(in) tt,
type(period), intent(out) readPer )

```

calculate beginning and end of read Period, i.e. that is length of current chunk to read

Parameters

in	<i>integer(i4) :: iBasin</i>	current Basin to process
in	<i>integer(i4) :: tt</i>	current time step
in	<i>type(period) :: readPer</i>	start and end dates of read Period

Author

Stephan Thober

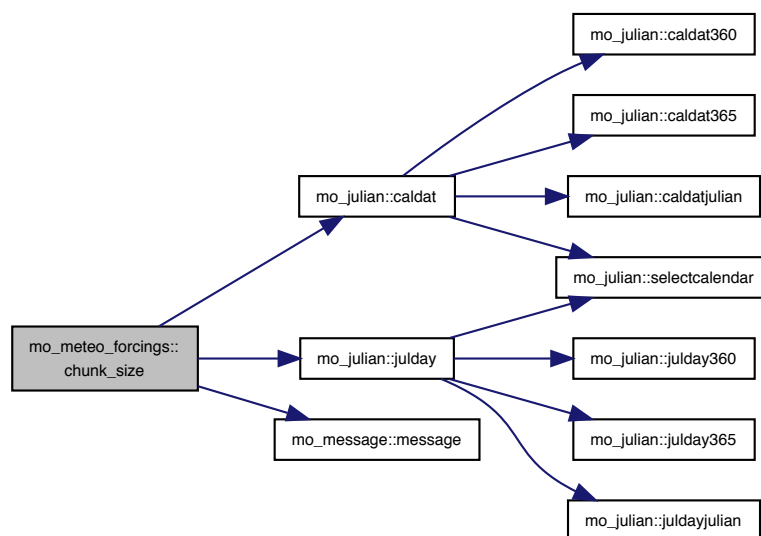
Date

Jun 2014

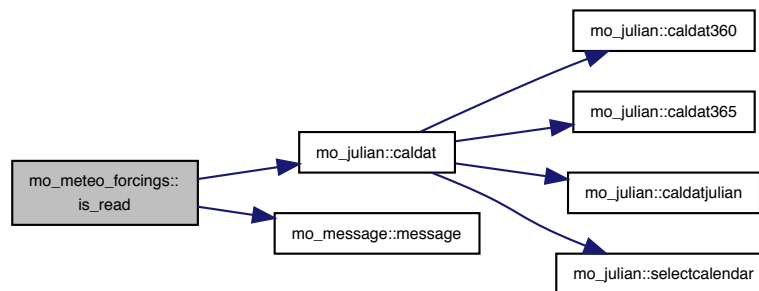
References `mo_julian::caldat()`, `mo_kind::i4`, `mo_julian::julday()`, `mo_message::message()`, and `mo_global_variables::simper`.

Referenced by `chunk_config()`.

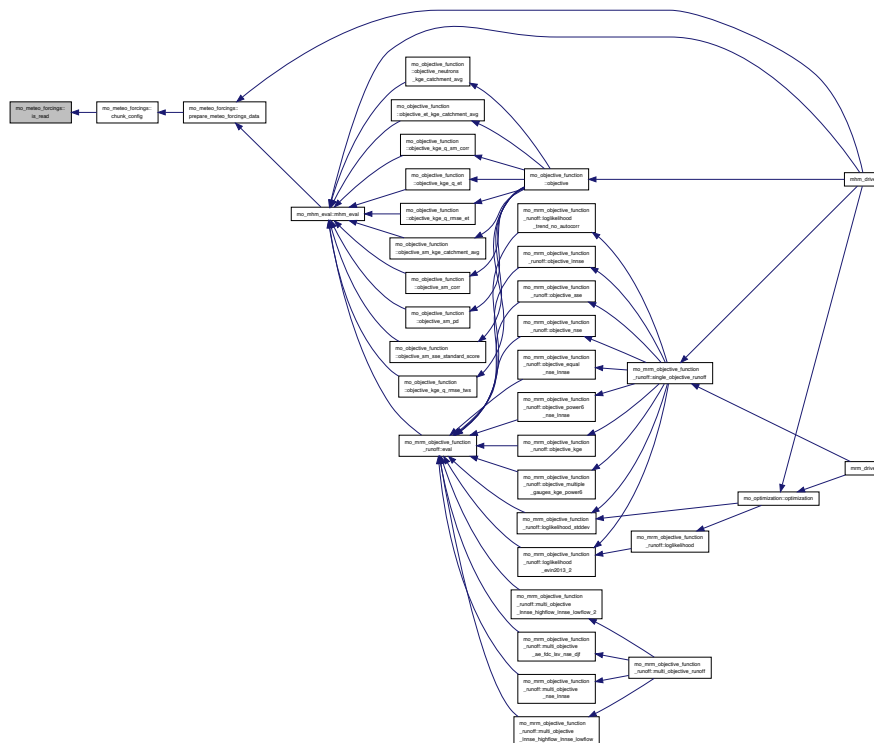
Here is the call graph for this function:



Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.4 meteo_forcings_wrapper()

```

subroutine mo_meteo_forcings::meteo_forcings_wrapper (
  integer(i4), intent(in) iBasin,
  character(len=*), intent(in) dataPath,
  character(len=*), intent(in) inputFormat,
  real(dp), dimension(:, :), intent(inout), allocatable dataOut1,
  real(dp), intent(in), optional lower,

```

```

real(dp), intent(in), optional upper,
character(len=*), intent(in), optional ncvarName )

```

Prepare meteorological forcings data for mHM at Level-1.

Prepare meteorological forcings data for mHM, which include

- 1) Reading meteo. datasets at their native resolution for every basin
- 2) Perform aggregation or disaggregation of meteo. datasets from their native resolution (level-2) to the required hydrologic resolution (level-1)
- 3) Pad the above datasets of every basin to their respective global ones

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
in	<i>character(len=*) :: dataPath</i>	Data path where a given meteo. variable is stored
in	<i>real(dp), dimension(:, :) :: dataOut1</i>	Packed meteorological variable for the whole simulation period
in	<i>real(dp), optional :: lower</i>	Lower bound for check of validity of data values
in	<i>real(dp), optional :: upper</i>	Upper bound for check of validity of data values
in	<i>type(period), optional :: readPer</i>	reading Period
in	<i>character(len=*), optional :: ncvarName</i>	name of the variable (for .nc files)

Author

Rohini Kumar

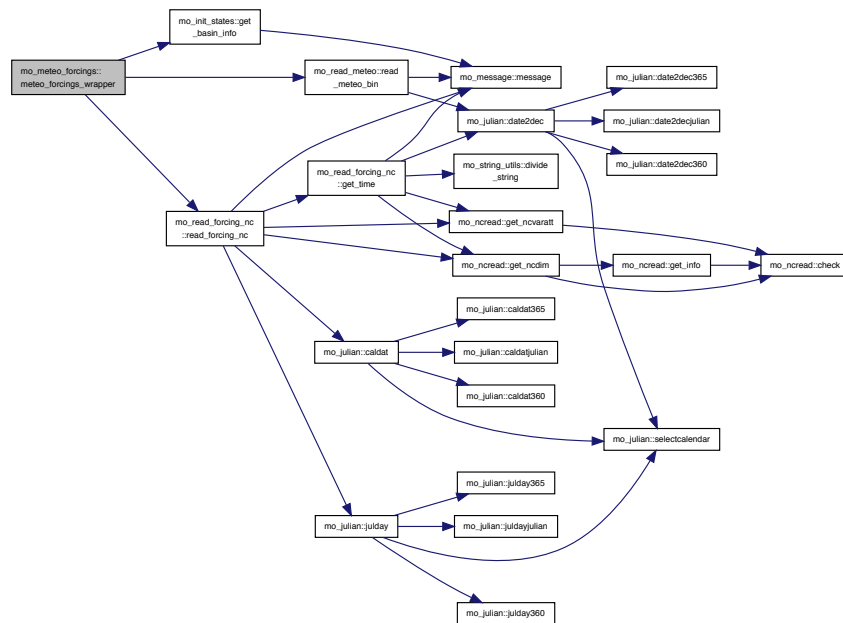
Date

Jan 2013

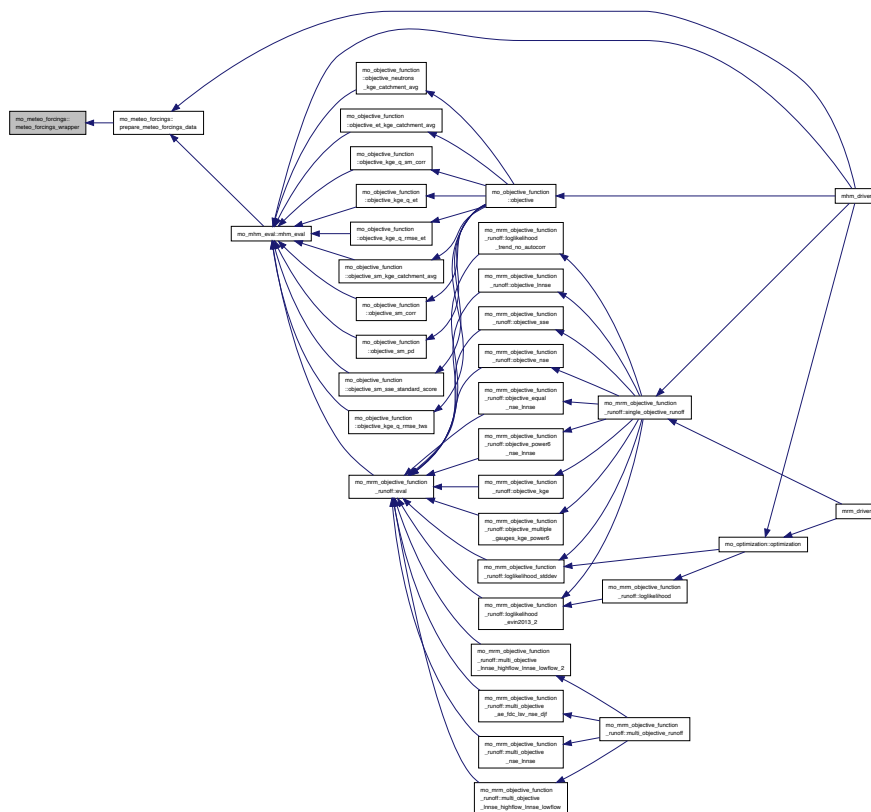
References `mo_init_states::get_basin_info()`, `mo_global_variables::level1`, `mo_global_variables::level2`, `mo_mhm↵_constants::nodata_dp`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_read_meteo::read_meteo_bin()`, and `mo_↵global_variables::readper`.

Referenced by `prepare_meteo_forcings_data()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.19.2.5 meteo_weights_wrapper()

```
subroutine mo_meteo_forcings::meteo_weights_wrapper (
    integer(i4), intent(in) iBasin,
    logical, intent(in) read_meteo_weights,
    character(len=*), intent(in) dataPath,
    real(dp), dimension(:,:,:), intent(inout), allocatable dataOut1,
    real(dp), intent(in), optional lower,
    real(dp), intent(in), optional upper,
    character(len=*), intent(in), optional ncvarName )
```

Prepare weights for meteorological forcings data for mHM at Level-1.

Prepare meteorological weights data for mHM, which include

- 1) Reading meteo. weights datasets at their native resolution for every basin
- 2) Perform aggregation or disaggregation of meteo. weights datasets from their native resolution (level-2) to the required hydrologic resolution (level-1)
- 3) Pad the above datasets of every basin to their respective global ones

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
in	<i>logical :: read_meteo_weights</i>	Flag for reading meteo weights
in	<i>character(len=*) :: dataPath</i>	Data path where a given meteo. variable is stored
in	<i>real(dp), dimension(:,:,:) :: dataOut1</i>	Packed meteorological variable for the whole simulation period
in	<i>real(dp), optional :: lower</i>	Lower bound for check of validity of data values
in	<i>real(dp), optional :: upper</i>	Upper bound for check of validity of data values
in	<i>character(len=*), optional :: ncvarName</i>	name of the variable (for .nc files)

Author

Stephan Thober & Rohini Kumar

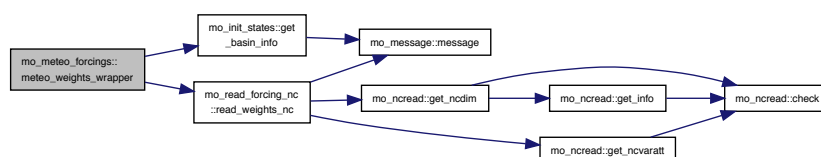
Date

Jan 2017

References `mo_init_states::get_basin_info()`, `mo_global_variables::level1`, `mo_global_variables::level2`, `mo_mhm_constants::nodata_dp`, and `mo_read_forcing_nc::read_weights_nc()`.

Referenced by `prepare_meteo_forcings_data()`.

Here is the call graph for this function:



[illegible]

```

subroutine, public mo_meteo_forcings::prepare_meteo_forcings_data (
    integer(i4), intent(in)  iBasin,
    integer(i4), intent(in)  tt )

```

Prepare meteorological forcings data for a given variable. Internally this subroutine calls another routine `meteo_` wrapper for different meterological variables

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Rohini Kumar

Call all main processes of mHM.

- subroutine, public **mhm** (perform_mpr, read_states, fSealedInCity, timeStep_LAI_input, counter_year, counter_month, counter_day, tt, time, processMatrix, c2TSTu, horizon_depth, nCells1, nHorizons_mHM, nimesteps_day, mask0, neutron_integral_AFast, iflag_soil_option, global_parameters, LCyearId, GeoUnit←List, GeoUnitKar, LAIUnitList, LAI_LUT, slope_emp0, L0_Latitude, cellId0, soilId0, L0_LCover_LAI, LCover0, Asp0, LAI0, geoUnit0, SDB_is_present, SDB_nHorizons, SDB_nTillHorizons, SDB_sand, SDB_clay, SDB←_DbM, SDB_Wd, SDB_RZdepth, nTCells0_inL1, L0upBound_inL1, L0downBound_inL1, L0leftBound_inL1, L0rightBound_inL1, latitude, evap_coeff, fday_prec, fnight_prec, fday_pet, fnight_pet, fday_temp, fnight←_temp, temp_weights, pet_weights, pre_weights, read_meteo_weights, pet_in, tmin_in, tmax_in, netrad←_in, absvappres_in, windspeed_in, prec_in, temp_in, yld, fForest1, fPerm1, fSealed1, interc, snowpack, sealedStorage, soilMoisture, unsatStorage, satStorage, neutrons, pet_calc, aet_soil, aet_canopy, aet←sealed, baseflow, infiltration, fast_interflow, melt, perc, prec_effect, rain, runoff_sealed, slow_interflow, snow, throughfall, total_runoff, alpha, deg_day_incr, deg_day_max, deg_day_noprec, deg_day, fAsp, petLAI←corFactorL1, HarSamCoeff, PrieTayAlpha, aeroResist, surfResist, frac_roots, interc_max, karst_loss, k0, k1, k2, kp, soil_moist_FC, soil_moist_sat, soil_moist_exponen, jarvis_thresh_c1, temp_thresh, unsat_thresh, water_thresh_sealed, wilting_point)

Pure mHM calculations.

15.20.1 Detailed Description

Call all main processes of mHM.

This module calls all processes of mHM for a given configuration. The configuration of the model is stored in the a process matrix. This configuration is specified in the namelist mhm.nml.

The processes are executed in ascending order. At the moment only process 5 and 8 have options.

The MPR technique is only called either if the land cover has been changed or for very first time step.

Currently the following processes are implemented:

Process	Name	Flag	Description
1	interception	1	Maximum interception
2	snow and melting	1	Degree-day
3	soil moisture	1	Feddes equation for ET reduction, Brooks-Corey like
3	soil moisture	2	Jarvis equation for ET reduction, Brooks-Corey like
3	soil moisture	3	Jarvis eq. for ET red. + FC dependency on root frac. coef.
4	direct runoff	1	Linear reservoir exceedance
5	PET	-1	PET is input, LAI based correction, dynamic scaling func.
5	PET	0	PET is input, Aspect based correction
5	PET	1	Hargreaves-Samani
5	PET	2	Priestley-Taylor
5	PET	3	Penman-Monteith
6	interflow	1	Nonlinear reservoir with saturation excess
7	percolation and base flow	1	GW linear reservoir
8	routing	0	no routing
8	routing	1	use mRM i.e. Muskingum
8	routing	2	use mRM i.e. adaptive timestep

Author

Luis Samaniego

Date

Dec 2012

15.20.2 Function/Subroutine Documentation

15.20.2.1 mhm()

```
subroutine, public mo_mhm::mhm (
    logical, intent(in) perform_mpr,
    logical, intent(in) read_states,
    real(dp), intent(in) fSealedInCity,
    integer(i4), intent(in) timeStep_LAI_input,
    integer(i4), intent(in) counter_year,
    integer(i4), intent(in) counter_month,
```

```

integer(i4), intent(in) counter_day,
integer(i4), intent(in) tt,
real(dp), intent(in) time,
integer(i4), dimension(:,:), intent(in) processMatrix,
real(dp), intent(in) c2TSTu,
real(dp), dimension(:), intent(in) horizon_depth,
integer(i4), intent(in) nCells1,
integer(i4), intent(in) nHorizons_mHM,
real(dp), intent(in) ntimesteps_day,
logical, dimension(:,:), intent(in) mask0,
real(dp), dimension(:), intent(in) neutron_integral_AFast,
integer(i4), intent(in) iflag_soil_option,
real(dp), dimension(:), intent(in) global_parameters,
integer(i4), intent(in) LCyearId,
integer(i4), dimension(:), intent(in) GeoUnitList,
integer(i4), dimension(:), intent(in) GeoUnitKar,
integer(i4), dimension(:), intent(in) LAIUnitList,
real(dp), dimension(:,:), intent(in) LAILUT,
real(dp), dimension(:), intent(in) slope_emp0,
real(dp), dimension(:), intent(in) L0_Latitude,
integer(i4), dimension(:), intent(in) cellId0,
integer(i4), dimension(:,:), intent(in) soilId0,
integer(i4), dimension(:), intent(in) L0_LCover_LAI,
integer(i4), dimension(:), intent(in) LCover0,
real(dp), dimension(:), intent(in) Asp0,
real(dp), dimension(:), intent(in) LAI0,
integer(i4), dimension(:), intent(in) geoUnit0,
integer(i4), dimension(:), intent(in) SDB_is_present,
integer(i4), dimension(:), intent(in) SDB_nHorizons,
integer(i4), dimension(:), intent(in) SDB_nTillHorizons,
real(dp), dimension(:,:), intent(in) SDB_sand,
real(dp), dimension(:,:), intent(in) SDB_clay,
real(dp), dimension(:,:), intent(in) SDB_DbM,
real(dp), dimension(:,:,:), intent(in) SDB_Wd,
real(dp), dimension(:), intent(in) SDB_RZdepth,
integer(i4), dimension(:), intent(in) nTCells0_inL1,
integer(i4), dimension(:), intent(in) L0upBound_inL1,
integer(i4), dimension(:), intent(in) L0downBound_inL1,
integer(i4), dimension(:), intent(in) L0leftBound_inL1,
integer(i4), dimension(:), intent(in) L0rightBound_inL1,
real(dp), dimension(:), intent(in) latitude,
real(dp), dimension(:), intent(in) evap_coeff,
real(dp), dimension(:), intent(in) fday_prec,
real(dp), dimension(:), intent(in) fnight_prec,
real(dp), dimension(:), intent(in) fday_pet,
real(dp), dimension(:), intent(in) fnight_pet,
real(dp), dimension(:), intent(in) fday_temp,
real(dp), dimension(:), intent(in) fnight_temp,
real(dp), dimension(:,:,:), intent(in) temp_weights,
real(dp), dimension(:,:,:), intent(in) pet_weights,
real(dp), dimension(:,:,:), intent(in) pre_weights,
logical, intent(in) read_meteo_weights,
real(dp), dimension(:), intent(in) pet_in,
real(dp), dimension(:), intent(in) tmin_in,
real(dp), dimension(:), intent(in) tmax_in,
real(dp), dimension(:), intent(in) netrad_in,
real(dp), dimension(:), intent(in) absvappres_in,
real(dp), dimension(:), intent(in) windspeed_in,

```

```

real(dp), dimension(:), intent(in) prec_in,
real(dp), dimension(:), intent(in) temp_in,
integer(i4), intent(inout) yId,
real(dp), dimension(:), intent(inout) fForest1,
real(dp), dimension(:), intent(inout) fPerm1,
real(dp), dimension(:), intent(inout) fSealed1,
real(dp), dimension(:), intent(inout) interc,
real(dp), dimension(:), intent(inout) snowpack,
real(dp), dimension(:), intent(inout) sealedStorage,
real(dp), dimension(:, :), intent(inout) soilMoisture,
real(dp), dimension(:), intent(inout) unsatStorage,
real(dp), dimension(:), intent(inout) satStorage,
real(dp), dimension(:), intent(inout) neutrons,
real(dp), dimension(:), intent(inout) pet_calc,
real(dp), dimension(:, :), intent(inout) aet_soil,
real(dp), dimension(:), intent(inout) aet_canopy,
real(dp), dimension(:), intent(inout) aet_sealed,
real(dp), dimension(:), intent(inout) baseflow,
real(dp), dimension(:, :), intent(inout) infiltration,
real(dp), dimension(:), intent(inout) fast_interflow,
real(dp), dimension(:), intent(inout) melt,
real(dp), dimension(:), intent(inout) perc,
real(dp), dimension(:), intent(inout) prec_effect,
real(dp), dimension(:), intent(inout) rain,
real(dp), dimension(:), intent(inout) runoff_sealed,
real(dp), dimension(:), intent(inout) slow_interflow,
real(dp), dimension(:), intent(inout) snow,
real(dp), dimension(:), intent(inout) throughfall,
real(dp), dimension(:), intent(inout) total_runoff,
real(dp), dimension(:), intent(inout) alpha,
real(dp), dimension(:), intent(inout) deg_day_incr,
real(dp), dimension(:), intent(inout) deg_day_max,
real(dp), dimension(:), intent(inout) deg_day_noprec,
real(dp), dimension(:), intent(inout) deg_day,
real(dp), dimension(:), intent(inout) fAsp,
real(dp), dimension(:), intent(inout) petLAIcorFactorL1,
real(dp), dimension(:), intent(inout) HarSamCoeff,
real(dp), dimension(:, :), intent(inout) PrieTayAlpha,
real(dp), dimension(:, :), intent(inout) aeroResist,
real(dp), dimension(:, :), intent(inout) surfResist,
real(dp), dimension(:, :), intent(inout) frac_roots,
real(dp), dimension(:), intent(inout) interc_max,
real(dp), dimension(:), intent(inout) karst_loss,
real(dp), dimension(:), intent(inout) k0,
real(dp), dimension(:), intent(inout) k1,
real(dp), dimension(:), intent(inout) k2,
real(dp), dimension(:), intent(inout) kp,
real(dp), dimension(:, :), intent(inout) soil_moist_FC,
real(dp), dimension(:, :), intent(inout) soil_moist_sat,
real(dp), dimension(:, :), intent(inout) soil_moist_exponen,
real(dp), dimension(:), intent(inout) jarvis_thresh_c1,
real(dp), dimension(:), intent(inout) temp_thresh,
real(dp), dimension(:), intent(inout) unsat_thresh,
real(dp), dimension(:), intent(inout) water_thresh_sealed,
real(dp), dimension(:, :), intent(inout) wilting_point )

```

Pure mHM calculations.

Pure mHM calculations. All variables are allocated and initialized.

They will be local variables within this call.

Note

Fields must be consistent to DEM.

Author

Luis Samaniego & Rohini Kumar

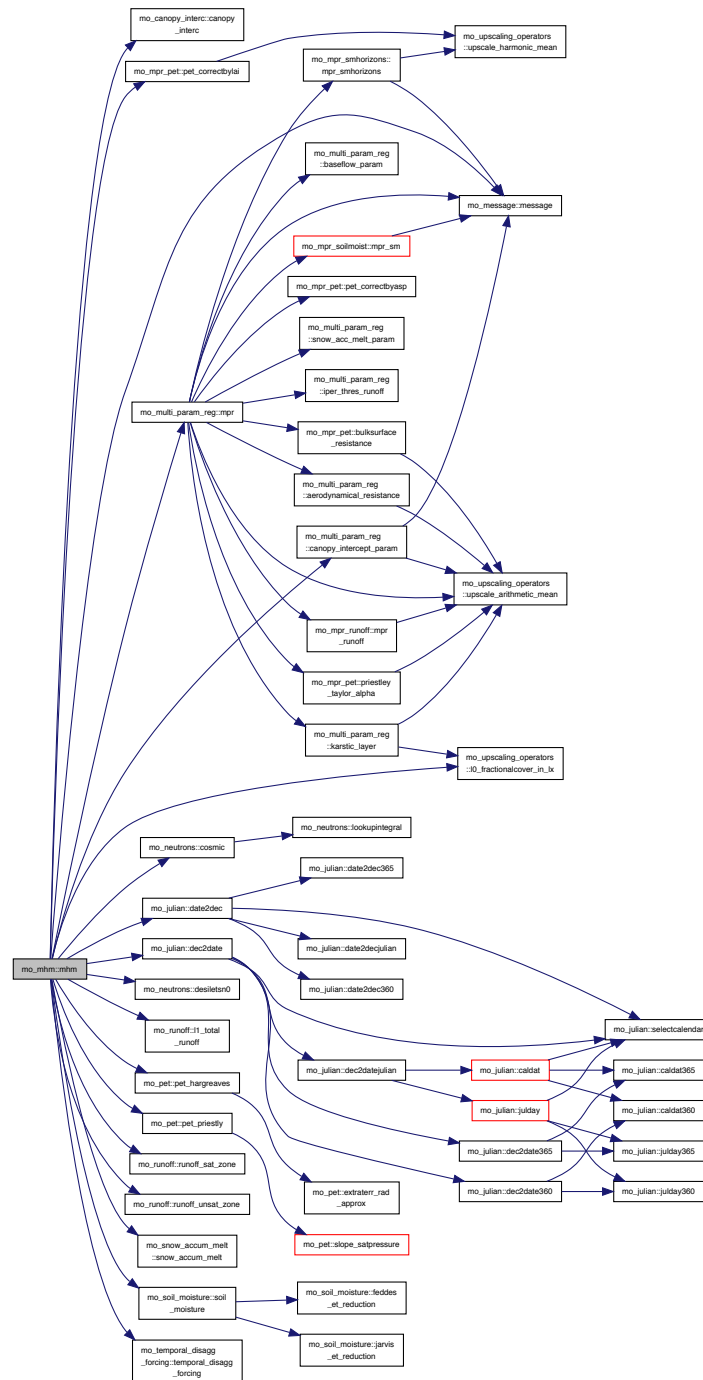
Date

Dec 2012

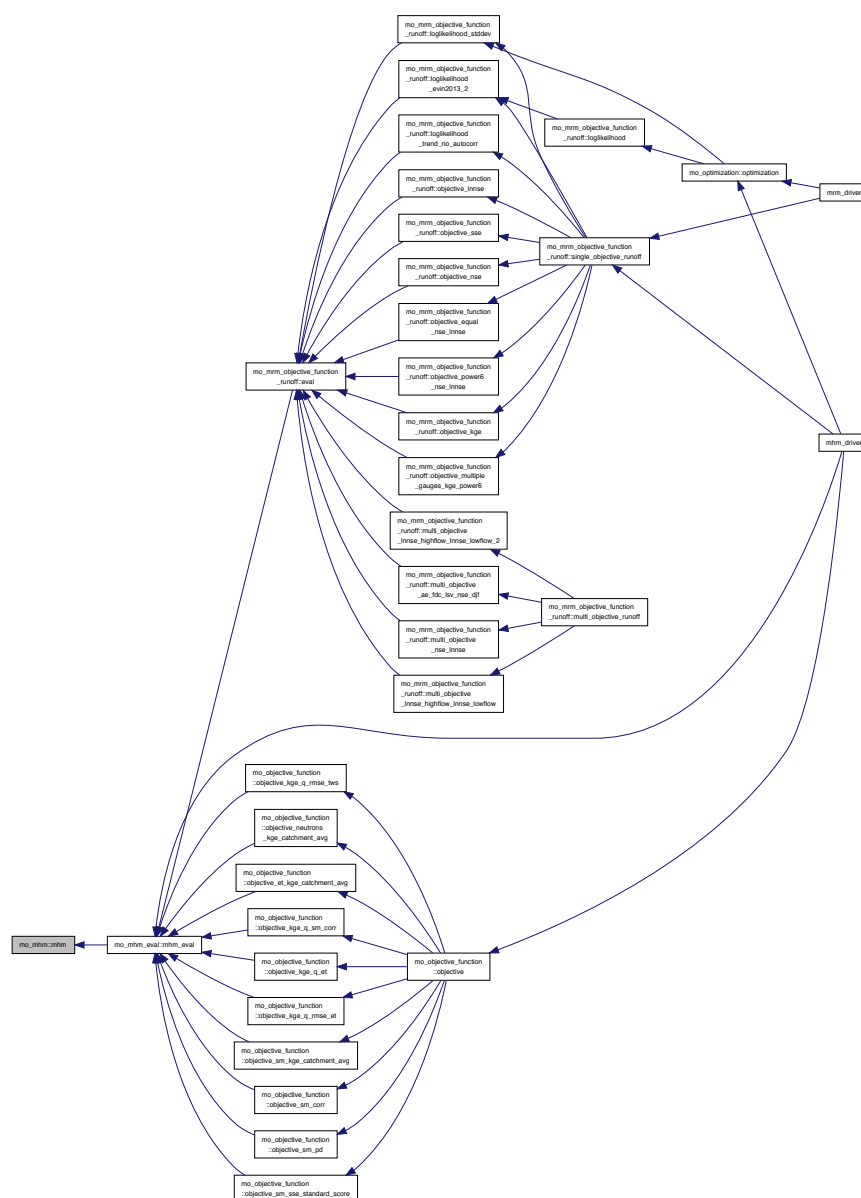
References mo_canopy_interc::canopy_interc(), mo_multi_param_reg::canopy_intercept_param(), mo_neutrons↵
::cosmic(), mo_julian::date2dec(), mo_julian::dec2date(), mo_neutrons::desiletsn0(), mo_mhm_constants↵
::harsamconst, mo_upscaling_operators::l0_fractionalcover_in_lx(), mo_runoff::l1_total_runoff(), mo_message↵
::message(), mo_multi_param_reg::mpr(), mo_mpr_pet::pet_correctbylai(), mo_pet::pet_hargreaves(), mo_pet↵
::pet_priestly(), mo_runoff::runoff_sat_zone(), mo_runoff::runoff_unsat_zone(), mo_snow_accum_melt::snow_↵
accum_melt(), mo_soil_moisture::soil_moisture(), and mo_temporal_disagg_forcing::temporal_disagg_forcing().

Referenced by mo_mhm_eval::mhm_eval().

Here is the call graph for this function:



Here is the caller graph for this function:



15.21 mo_mhm_constants Module Reference

Provides mHM specific constants.

Variables

- real(dp), parameter, public `h2odens = 1000.0_dp`
- integer(i4), parameter, public `nodata_i4 = -9999_i4`
- real(dp), parameter, public `nodata_dp = -9999._dp`
- integer(i4), parameter, public `nlcover_class = 3_i4`
- integer(i4), parameter, public `ncolpars = 5_i4`
- integer(i4), parameter, public `maxnosoilhorizons = 10_i4`

- integer(i4), parameter, public `maxnobasins` = 50_i4
- integer(i4), parameter, public `maxnlcovers` = 50_i4
- integer(i4), parameter, public `maxgeounit` = 25_i4
- real(dp), parameter, public `p1_initstatefluxes` = 0.00_dp
- real(dp), parameter, public `p2_initstatefluxes` = 15.00_dp
- real(dp), parameter, public `p3_initstatefluxes` = 10.00_dp
- real(dp), parameter, public `p4_initstatefluxes` = 75.00_dp
- real(dp), parameter, public `p5_initstatefluxes` = 1500.00_dp
- real(dp), parameter, public `c1_initstatesm` = 0.25_dp
- integer(i4), parameter, public `noutflxstate` = 20_i4
- real(dp), parameter, public `dayhours` = 24.0_dp
- real(dp), parameter, public `yearmonths` = 12.0_dp
- integer(i4), parameter, public `yearmonths_i4` = 12
- real(dp), parameter, public `yeardays` = 365.0_dp
- real(dp), parameter, public `daysecs` = 86400.0_dp
- real(dp), parameter, public `bulkdens_orgmatter` = 0.224_dp
- real(dp), parameter, public `field_cap_c1` = -0.60_dp
- real(dp), parameter, public `field_cap_c2` = 2.0_dp
- real(dp), parameter, public `vgenuchten_sandtresh` = 66.5_dp
- real(dp), parameter, public `vgenuchtenn_c1` = 1.392_dp
- real(dp), parameter, public `vgenuchtenn_c2` = 0.418_dp
- real(dp), parameter, public `vgenuchtenn_c3` = -0.024_dp
- real(dp), parameter, public `vgenuchtenn_c4` = 1.212_dp
- real(dp), parameter, public `vgenuchtenn_c5` = -0.704_dp
- real(dp), parameter, public `vgenuchtenn_c6` = -0.648_dp
- real(dp), parameter, public `vgenuchtenn_c7` = 0.023_dp
- real(dp), parameter, public `vgenuchtenn_c8` = 0.044_dp
- real(dp), parameter, public `vgenuchtenn_c9` = 3.168_dp
- real(dp), parameter, public `vgenuchtenn_c10` = -2.562_dp
- real(dp), parameter, public `vgenuchtenn_c11` = 7.0E-9_dp
- real(dp), parameter, public `vgenuchtenn_c12` = 4.004_dp
- real(dp), parameter, public `vgenuchtenn_c13` = 3.750_dp
- real(dp), parameter, public `vgenuchtenn_c14` = -0.016_dp
- real(dp), parameter, public `vgenuchtenn_c15` = -4.197_dp
- real(dp), parameter, public `vgenuchtenn_c16` = 0.013_dp
- real(dp), parameter, public `vgenuchtenn_c17` = 0.076_dp
- real(dp), parameter, public `vgenuchtenn_c18` = 0.276_dp
- real(dp), parameter, public `ks_c` = 10.0_dp
- real(dp), parameter, public `pwp_c` = 1.0_dp
- real(dp), parameter, public `pwp_matpot_thetar` = 15000.0_dp
- real(dp), parameter, public `stboltzmann` = 5.67e-08_dp
Stefan-Boltzmann constant [$W\ m^{-2}\ K^{-4}$].
- real(dp), parameter, public `harsamconst` = 17.800_dp
Hargreaves-Samani ref. ET formula [deg C].
- real(dp), parameter, public `windmeasheight` = 10.0_dp
assumed meteorol. measurement hight for estimation of aeroResist and surfResist
- real(dp), parameter, public `karman` = 0.41_dp
von karman constant
- real(dp), parameter, public `lai_factor_surfresi` = 0.3_dp
LAI factor for bulk surface resistance formulation.
- real(dp), parameter, public `lai_offset_surfresi` = 1.2_dp
LAI offset for bulk surface resistance formulation.
- real(dp), parameter, public `max_surfresist` = 250.0_dp

maximum bulk surface resistance

- real(dp), parameter, public `duffiedr` = 0.0330_dp
- real(dp), parameter, public `duffiedelta1` = 0.4090_dp
- real(dp), parameter, public `duffiedelta2` = 1.3900_dp
- real(dp), parameter, public `tetens_c1` = 0.6108_dp

Tetens's formula to calculate saturated vapour pressure.

- real(dp), parameter, public `tetens_c2` = 17.270_dp
- real(dp), parameter, public `tetens_c3` = 237.30_dp
- real(dp), parameter, public `satpressureslope1` = 4098.0_dp

calculation of the slope of the saturation vapour pressure curve following Tetens

- real(dp), parameter, public `desilets_a0` = 0.0808_dp

Neutrons and moisture: N0 formula, Desilets et al. 2010.

- real(dp), parameter, public `desilets_a1` = 0.372_dp
- real(dp), parameter, public `desilets_a2` = 0.115_dp
- real(dp), parameter, public `cosmic_bd` = 1.4020_dp

Neutrons and moisture: COSMIC, Shuttleworth et al. 2013.

- real(dp), parameter, public `cosmic_vwclat` = 0.0753_dp
- real(dp), parameter, public `cosmic_n` = 348.33_dp
- real(dp), parameter, public `cosmic_alpha` = 0.2392421548_dp
- real(dp), parameter, public `cosmic_l1` = 161.98621864_dp
- real(dp), parameter, public `cosmic_l2` = 129.14558985_dp
- real(dp), parameter, public `cosmic_l3` = 107.82204562_dp
- real(dp), parameter, public `cosmic_l4` = 3.1627190566_dp

15.21.1 Detailed Description

Provides mHM specific constants.

Provides mHM specific constants such as flood plain elevation.

Author

Matthias Cuntz

Date

Nov 2011

15.21.2 Variable Documentation

15.21.2.1 bulkdens_orgmatter

```
real(dp), parameter, public mo_mhm_constants::bulkdens_orgmatter = 0.224_dp
```

Referenced by `mo_mpr_soilmoist::mpr_sm()`.

15.21.2.2 c1_initstatesm

```
real(dp), parameter, public mo_mhm_constants::c1_initstatesm = 0.25_dp
```

Referenced by `mo_init_states::variables_default_init()`.

15.21.2.3 cosmic_alpha

```
real(dp), parameter, public mo_mhm_constants::cosmic_alpha = 0.2392421548_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.4 cosmic_bd

```
real(dp), parameter, public mo_mhm_constants::cosmic_bd = 1.4020_dp
```

Neutrons and moisture: COSMIC, Shuttleworth et al. 2013.

Referenced by mo_neutrons::cosmic().

15.21.2.5 cosmic_l1

```
real(dp), parameter, public mo_mhm_constants::cosmic_l1 = 161.98621864_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.6 cosmic_l2

```
real(dp), parameter, public mo_mhm_constants::cosmic_l2 = 129.14558985_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.7 cosmic_l3

```
real(dp), parameter, public mo_mhm_constants::cosmic_l3 = 107.82204562_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.8 cosmic_l4

```
real(dp), parameter, public mo_mhm_constants::cosmic_l4 = 3.1627190566_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.9 cosmic_n

```
real(dp), parameter, public mo_mhm_constants::cosmic_n = 348.33_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.10 cosmic_vwclat

```
real(dp), parameter, public mo_mhm_constants::cosmic_vwclat = 0.0753_dp
```

Referenced by mo_neutrons::cosmic().

15.21.2.11 dayhours

```
real(dp), parameter, public mo_mhm_constants::dayhours = 24.0_dp
```

15.21.2.12 daysecs

```
real(dp), parameter, public mo_mhm_constants::daysecs = 86400.0_dp
```

Referenced by mo_pet::extraterr_rad_approx(), mo_pet::pet_penman(), and mo_pet::pet_priestly().

15.21.2.13 desilets_a0

```
real(dp), parameter, public mo_mhm_constants::desilets_a0 = 0.0808_dp
```

Neutrons and moisture: N0 formula, Desilets et al. 2010.

Referenced by mo_neutrons::desiletsn0().

15.21.2.14 desilets_a1

```
real(dp), parameter, public mo_mhm_constants::desilets_a1 = 0.372_dp
```

Referenced by mo_neutrons::desiletsn0().

15.21.2.15 desilets_a2

```
real(dp), parameter, public mo_mhm_constants::desilets_a2 = 0.115_dp
```

Referenced by mo_neutrons::desiletsn0().

15.21.2.16 duffiedelta1

```
real(dp), parameter, public mo_mhm_constants::duffiedelta1 = 0.4090_dp
```

Referenced by mo_pet::extraterr_rad_approx().

15.21.2.17 duffiedelta2

```
real(dp), parameter, public mo_mhm_constants::duffiedelta2 = 1.3900_dp
```

Referenced by mo_pet::extraterr_rad_approx().

15.21.2.18 duffiedr

```
real(dp), parameter, public mo_mhm_constants::duffiedr = 0.0330_dp
```

Referenced by `mo_pet::extraterr_rad_approx()`.

15.21.2.19 field_cap_c1

```
real(dp), parameter, public mo_mhm_constants::field_cap_c1 = -0.60_dp
```

Referenced by `mo_mpr_soilmoist::field_cap()`.

15.21.2.20 field_cap_c2

```
real(dp), parameter, public mo_mhm_constants::field_cap_c2 = 2.0_dp
```

Referenced by `mo_mpr_soilmoist::field_cap()`.

15.21.2.21 h2odens

```
real(dp), parameter, public mo_mhm_constants::h2odens = 1000.0_dp
```

Referenced by `mo_neutrons::cosmic()`.

15.21.2.22 harsamconst

```
real(dp), parameter, public mo_mhm_constants::harsamconst = 17.800_dp
```

Hargreaves-Samani ref. ET formula [deg C].

Referenced by `mo_mhm::mhm()`.

15.21.2.23 karman

```
real(dp), parameter, public mo_mhm_constants::karman = 0.41_dp
```

von karman constant

Referenced by `mo_multi_param_reg::aerodynamical_resistance()`.

15.21.2.24 ks_c

```
real(dp), parameter, public mo_mhm_constants::ks_c = 10.0_dp
```

Referenced by `mo_mpr_soilmoist::hydro_cond()`.

15.21.2.25 lai_factor_surfresi

```
real(dp), parameter, public mo_mhm_constants::lai_factor_surfresi = 0.3_dp
```

LAI factor for bulk surface resistance formulation.

Referenced by mo_mpr_pet::bulksurface_resistance().

15.21.2.26 lai_offset_surfresi

```
real(dp), parameter, public mo_mhm_constants::lai_offset_surfresi = 1.2_dp
```

LAI offset for bulk surface resistance formulation.

Referenced by mo_mpr_pet::bulksurface_resistance().

15.21.2.27 max_surfresist

```
real(dp), parameter, public mo_mhm_constants::max_surfresist = 250.0_dp
```

maximum bulk surface resistance

Referenced by mo_mpr_pet::bulksurface_resistance().

15.21.2.28 maxgeounit

```
integer(i4), parameter, public mo_mhm_constants::maxgeounit = 25_i4
```

15.21.2.29 maxnlcovers

```
integer(i4), parameter, public mo_mhm_constants::maxnlcovers = 50_i4
```

15.21.2.30 maxnobasins

```
integer(i4), parameter, public mo_mhm_constants::maxnobasins = 50_i4
```

15.21.2.31 maxnosoilhorizons

```
integer(i4), parameter, public mo_mhm_constants::maxnosoilhorizons = 10_i4
```

15.21.2.32 ncolpars

```
integer(i4), parameter, public mo_mhm_constants::ncolpars = 5_i4
```

15.21.2.33 nlcover_class

```
integer(i4), parameter, public mo_mhm_constants::nlcover_class = 3_i4
```

Referenced by `mo_soil_database::read_soil_lut()`.

15.21.2.34 nodata_dp

```
real(dp), parameter, public mo_mhm_constants::nodata_dp = -9999._dp
```

Referenced by `mo_meteo_forcings::chunk_config()`, `mo_soil_database::generate_soil_database()`, `mo_startup::l0_check_input()`, `mo_startup::l0_variable_init()`, `mo_startup::l1_variable_init()`, `mo_startup::l2_variable_init()`, `mo_meteo_forcings::meteo_forcings_wrapper()`, `mo_meteo_forcings::meteo_weights_wrapper()`, `mo_mhm_eval::mhm_eval()`, `mo_objective_function::objective_et_kge_catchment_avg()`, `mo_objective_function::objective_neutrons_kge_catchment_avg()`, `mo_objective_function::objective_sm_kge_catchment_avg()`, `mo_objective_function::objective_sm_pd()`, `mo_read_optional_data::read_basin_avg_tws()`, `mo_read_config::read_config()`, `mo_read_wrapper::read_data()`, `mo_read_optional_data::read_evapotranspiration()`, `mo_read_optional_data::read_neutrons()`, `mo_restart::read_restart_config()`, `mo_soil_database::read_soil_lut()`, `mo_read_optional_data::read_soil_moisture()`, `mo_spatial_agg_disagg_forcing::spatial_aggregation_3d()`, `mo_spatial_agg_disagg_forcing::spatial_aggregation_4d()`, `mo_spatial_agg_disagg_forcing::spatial_disaggregation_3d()`, `mo_spatial_agg_disagg_forcing::spatial_disaggregation_4d()`, `mo_write_ascii::write_configfile()`, `mo_restart::write_restart_files()`, `mo_write_fluxes_states::writevariableattributes()`, and `mo_write_fluxes_states::writevariabletimestep()`.

15.21.2.35 nodata_i4

```
integer(i4), parameter, public mo_mhm_constants::nodata_i4 = -9999_i4
```

Referenced by `mo_soil_database::generate_soil_database()`, `mo_startup::l0_check_input()`, `mo_upscaling_operators::l0_fractionalcover_in_lx()`, `mo_startup::l0_variable_init()`, `mo_read_config::read_config()`, `mo_read_wrapper::read_data()`, `mo_soil_database::read_soil_lut()`, and `mo_restart::write_restart_files()`.

15.21.2.36 noutflxstate

```
integer(i4), parameter, public mo_mhm_constants::noutflxstate = 20_i4
```

15.21.2.37 p1_initstatefluxes

```
real(dp), parameter, public mo_mhm_constants::p1_initstatefluxes = 0.00_dp
```

Referenced by `mo_init_states::variables_default_init()`.

15.21.2.38 p2_initstatefluxes

```
real(dp), parameter, public mo_mhm_constants::p2_initstatefluxes = 15.00_dp
```

Referenced by `mo_init_states::variables_default_init()`.

15.21.2.39 p3_initstatefluxes

```
real(dp), parameter, public mo_mhm_constants::p3_initstatefluxes = 10.00_dp
```

Referenced by mo_init_states::variables_default_init().

15.21.2.40 p4_initstatefluxes

```
real(dp), parameter, public mo_mhm_constants::p4_initstatefluxes = 75.00_dp
```

Referenced by mo_init_states::variables_default_init().

15.21.2.41 p5_initstatefluxes

```
real(dp), parameter, public mo_mhm_constants::p5_initstatefluxes = 1500.00_dp
```

Referenced by mo_init_states::variables_default_init().

15.21.2.42 pwp_c

```
real(dp), parameter, public mo_mhm_constants::pwp_c = 1.0_dp
```

Referenced by mo_mpr_soilmoist::pwp().

15.21.2.43 pwp_matpot_thetar

```
real(dp), parameter, public mo_mhm_constants::pwp_matpot_thetar = 15000.0_dp
```

Referenced by mo_mpr_soilmoist::pwp().

15.21.2.44 satpressureslope1

```
real(dp), parameter, public mo_mhm_constants::satpressureslope1 = 4098.0_dp
```

calculation of the slope of the saturation vapour pressure curve following Tetens

Referenced by mo_pet::slope_satpressure().

15.21.2.45 stboltzmann

```
real(dp), parameter, public mo_mhm_constants::stboltzmann = 5.67e-08_dp
```

Stefan-Boltzmann constant [$\text{W m}^{-2} \text{K}^{-4}$].

15.21.2.46 tetens_c1

```
real(dp), parameter, public mo_mhm_constants::tetens_c1 = 0.6108_dp
```

Tetens's formula to calculate saturated vapour pressure.

Referenced by `mo_pet::sat_vap_pressure()`.

15.21.2.47 `tetens_c2`

```
real(dp), parameter, public mo_mhm_constants::tetens_c2 = 17.270_dp
```

Referenced by `mo_pet::sat_vap_pressure()`.

15.21.2.48 `tetens_c3`

```
real(dp), parameter, public mo_mhm_constants::tetens_c3 = 237.30_dp
```

Referenced by `mo_pet::sat_vap_pressure()`, and `mo_pet::slope_satpressure()`.

15.21.2.49 `vgenuchten_sandtresh`

```
real(dp), parameter, public mo_mhm_constants::vgenuchten_sandtresh = 66.5_dp
```

Referenced by `mo_mpr_soilmoist::genuchten()`.

15.21.2.50 `vgenuchtenn_c1`

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c1 = 1.392_dp
```

15.21.2.51 `vgenuchtenn_c10`

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c10 = -2.562_dp
```

15.21.2.52 `vgenuchtenn_c11`

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c11 = 7.0E-9_dp
```

15.21.2.53 `vgenuchtenn_c12`

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c12 = 4.004_dp
```

15.21.2.54 `vgenuchtenn_c13`

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c13 = 3.750_dp
```

15.21.2.55 vgenuchtenn_c14

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c14 = -0.016_dp
```

15.21.2.56 vgenuchtenn_c15

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c15 = -4.197_dp
```

15.21.2.57 vgenuchtenn_c16

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c16 = 0.013_dp
```

15.21.2.58 vgenuchtenn_c17

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c17 = 0.076_dp
```

15.21.2.59 vgenuchtenn_c18

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c18 = 0.276_dp
```

15.21.2.60 vgenuchtenn_c2

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c2 = 0.418_dp
```

15.21.2.61 vgenuchtenn_c3

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c3 = -0.024_dp
```

15.21.2.62 vgenuchtenn_c4

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c4 = 1.212_dp
```

15.21.2.63 vgenuchtenn_c5

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c5 = -0.704_dp
```

15.21.2.64 vgenuchtenn_c6

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c6 = -0.648_dp
```

15.21.2.65 vgenuchtenn_c7

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c7 = 0.023_dp
```

15.21.2.66 vgenuchtenn_c8

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c8 = 0.044_dp
```

15.21.2.67 vgenuchtenn_c9

```
real(dp), parameter, public mo_mhm_constants::vgenuchtenn_c9 = 3.168_dp
```

15.21.2.68 windmeasheight

```
real(dp), parameter, public mo_mhm_constants::windmeasheight = 10.0_dp
```

assumed meteorol. measurement hight for estimation of aeroResist and surfResist

Referenced by mo_multi_param_reg::aerodynamical_resistance().

15.21.2.69 yeardays

```
real(dp), parameter, public mo_mhm_constants::yeardays = 365.0_dp
```

Referenced by mo_pet::extraterr_rad_approx().

15.21.2.70 yearmonths

```
real(dp), parameter, public mo_mhm_constants::yearmonths = 12.0_dp
```

Referenced by mo_read_lut::read_lai_lut().

15.21.2.71 yearmonths_i4

```
integer(i4), parameter, public mo_mhm_constants::yearmonths_i4 = 12
```

Referenced by mo_multi_param_reg::aerodynamical_resistance(), mo_mpr_pet::bulksurface_resistance(), mo_mpr_pet::priestley_taylor_alpha(), mo_restart::read_restart_states(), and mo_init_states::variables_alloc().

15.22 mo_mhm_eval Module Reference

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Functions/Subroutines

- subroutine, public [mhm_eval](#) (parameterset, runoff, sm_opti, basin_avg_tws, neutrons_opti, et_opti)

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

15.22.1 Detailed Description

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Authors

Juliane Mai, Rohini Kumar

Date

Feb 2013

15.22.2 Function/Subroutine Documentation

15.22.2.1 mhm_eval()

```
subroutine, public mo_mhm_eval::mhm_eval (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), dimension(:,:), intent(out), optional, allocatable runoff,
    real(dp), dimension(:,:), intent(out), optional, allocatable sm_opti,
    real(dp), dimension(:,:), intent(out), optional, allocatable basin_avg_tws,
    real(dp), dimension(:,:), intent(out), optional, allocatable neutrons_opti,
    real(dp), dimension(:,:), intent(out), optional, allocatable et_opti )
```

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Parameters

in	<i>real(dp), dimension(:) :: parameterset</i>	a set of global parameter (gamma) to run mHM, DIMENSION [no. of global_Parameters]
out	<i>real(dp), dimension(:, :), optional :: runoff</i>	returns runoff time series, DIMENSION [nTimeSteps, nGaugesTotal]
out	<i>real(dp), dimension(:, :), optional :: sm_opti</i>	returns soil moisture time series for all grid cells (of multiple basins concatenated), DIMENSION [nCells, nTimeSteps]
out	<i>real(dp), dimension(:, :), optional :: basin_avg_tws</i>	returns basin averaged total water storage time series, DIMENSION [nTimeSteps, nBasins]
out	<i>real(dp), dimension(:, :), optional :: neutron_opti</i>	returns neutron counts time series for all grid cells (of multiple basins concatenated), DIMENSION [nCells, nTimeSteps]

Parameters

out	<i>real(dp), dimension(:,,:), optional :: et_opti</i>	returns evapotranspiration time series for all grid cells (of multiple basins concatenated), DIMENSION [nCells, nTimeSteps]
-----	---	--

Author

Juliane Mai, Rohini Kumar

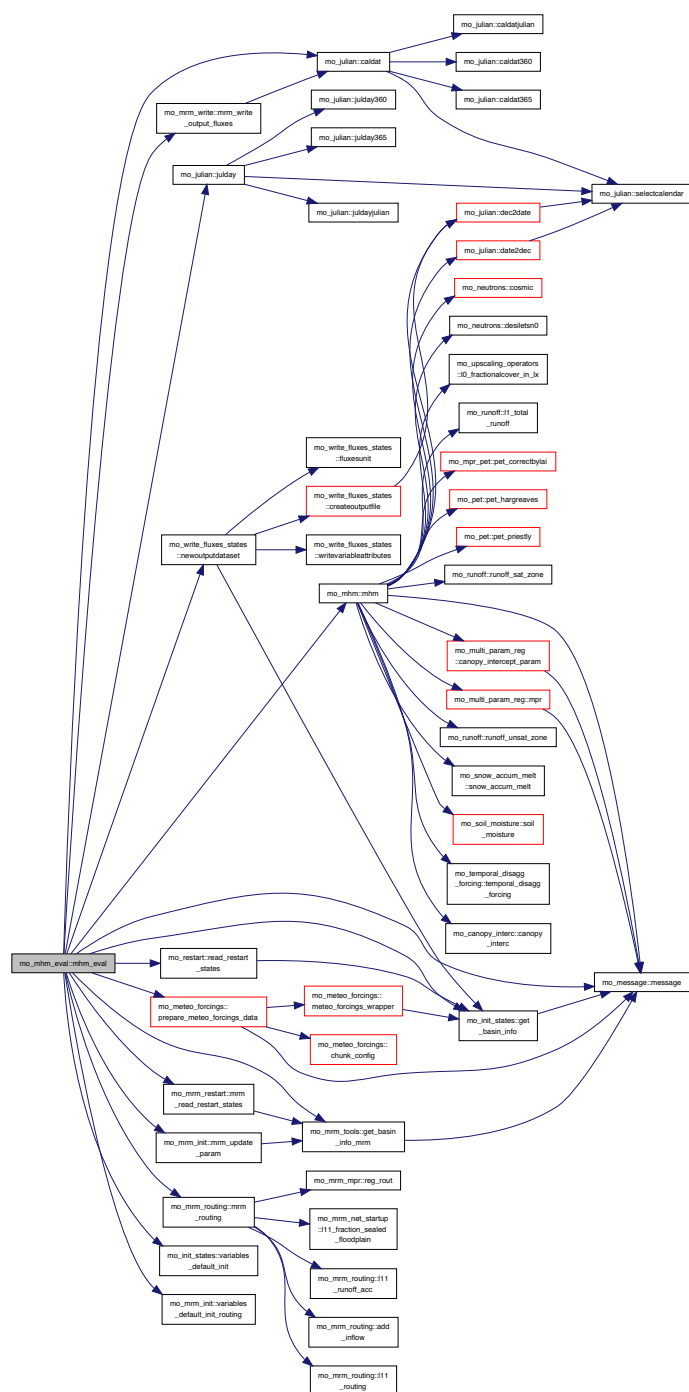
Date

Feb 2013

References mo_mrm_global_variables::basin_mrm, mo_julian::caldat(), mo_mrm_global_variables::dirrestartin, mo_mrm_global_variables::fracsealed_cityarea, mo_init_states::get_basin_info(), mo_mrm_tools::get_basin_info_mrm(), mo_mrm_constants::hoursecs, mo_kind::i4, mo_mrm_global_variables::inflowgauge, mo_julian::julday(), mo_mrm_global_variables::l0_areacell, mo_mrm_global_variables::l0_floodplain, mo_mrm_global_variables::l0_lcover_mrm, mo_mrm_global_variables::l11_afloodplain, mo_mrm_global_variables::l11_areacell, mo_mrm_global_variables::l11_c1, mo_mrm_global_variables::l11_c2, mo_mrm_global_variables::l11_fracpimp, mo_mrm_global_variables::l11_fromn, mo_mrm_global_variables::l11_l1_id, mo_mrm_global_variables::l11_length, mo_mrm_global_variables::l11_netperm, mo_mrm_global_variables::l11_noutlets, mo_mrm_global_variables::l11_qmod, mo_mrm_global_variables::l11_qout, mo_mrm_global_variables::l11_qtin, mo_mrm_global_variables::l11_qtr, mo_mrm_global_variables::l11_slope, mo_mrm_global_variables::l11_ton, mo_mrm_global_variables::l11_tsroun, mo_mrm_global_variables::l1_areacell, mo_mrm_global_variables::l1_l1_id, mo_message::message(), mo_mhm::mhm(), mo_mrm_restart::mrm_read_restart_states(), mo_mrm_routing::mrm_routing(), mo_mrm_global_variables::mrm_runoff, mo_mrm_init::mrm_update_param(), mo_mrm_write::mrm_write_output_fluxes(), mo_mrm_global_variables::nbasins, mo_write_fluxes_states::newoutputdataset(), mo_mhm_constants::nodata_dp, mo_global_variables::ntstepday, mo_common_variables::optimize, mo_global_variables::outputflxstate, mo_mrm_global_variables::outputflxstate_mrm, mo_mrm_global_variables::perform_mpr, mo_meteo_forcings::prepare_meteo_forcings_data(), mo_common_variables::processmatrix, mo_mrm_global_variables::read_restart, mo_restart::read_restart_states(), mo_mrm_global_variables::resolutionhydrology, mo_mrm_global_variables::resolutionrouting, mo_mrm_global_variables::simper, mo_mrm_global_variables::timestep, mo_mrm_global_variables::timestep_model_inputs, mo_global_variables::timestep_model_outputs, mo_mrm_global_variables::timestep_model_outputs_mrm, mo_init_states::variables_default_init(), mo_mrm_init::variables_default_init_routing(), and mo_mrm_global_variables::warmingdays_mrm.

Referenced by mo_mrm_objective_function_runoff::eval(), mhm_driver(), mo_objective_function::objective_et_kge_catchment_avg(), mo_objective_function::objective_kge_q_et(), mo_objective_function::objective_kge_q_rmse_et(), mo_objective_function::objective_kge_q_rmse_tws(), mo_objective_function::objective_kge_q_sm_corr(), mo_objective_function::objective_neutrons_kge_catchment_avg(), mo_objective_function::objective_sm_corr(), mo_objective_function::objective_sm_kge_catchment_avg(), mo_objective_function::objective_sm_pd(), and mo_objective_function::objective_sm_sse_standard_score().

Here is the call graph for this function:



Data Types

- Generated on December 1, 2017

- interface [correlation](#)
- interface [covariance](#)
- interface [kurtosis](#)
- interface [mean](#)
- interface [mixed_central_moment](#)
- interface [mixed_central_moment_var](#)
- interface [moment](#)
- interface [skewness](#)
- interface [stddev](#)
- interface [variance](#)

Functions/Subroutines

- real(dp) function [absdev_dp](#) (dat, mask)
- real(sp) function [absdev_sp](#) (dat, mask)
- real(dp) function [average_dp](#) (dat, mask)
- real(sp) function [average_sp](#) (dat, mask)
- real(dp) function [central_moment_dp](#) (x, r, mask)
- real(sp) function [central_moment_sp](#) (x, r, mask)
- real(dp) function [central_moment_var_dp](#) (x, r, mask)
- real(sp) function [central_moment_var_sp](#) (x, r, mask)
- real(dp) function [correlation_dp](#) (x, y, mask)
- real(sp) function [correlation_sp](#) (x, y, mask)
- real(dp) function [covariance_dp](#) (x, y, mask)
- real(sp) function [covariance_sp](#) (x, y, mask)
- real(dp) function [kurtosis_dp](#) (dat, mask)
- real(sp) function [kurtosis_sp](#) (dat, mask)
- real(dp) function [mean_dp](#) (dat, mask)
- real(sp) function [mean_sp](#) (dat, mask)
- real(dp) function [mixed_central_moment_dp](#) (x, y, r, s, mask)
- real(sp) function [mixed_central_moment_sp](#) (x, y, r, s, mask)
- real(dp) function [mixed_central_moment_var_dp](#) (x, y, r, s, mask)
- real(sp) function [mixed_central_moment_var_sp](#) (x, y, r, s, mask)
- subroutine [moment_dp](#) (dat, [average](#), [variance](#), [skewness](#), [kurtosis](#), [mean](#), [stddev](#), [absdev](#), mask)
- subroutine [moment_sp](#) (dat, [average](#), [variance](#), [skewness](#), [kurtosis](#), [mean](#), [stddev](#), [absdev](#), mask)
- real(dp) function [stddev_dp](#) (dat, mask)
- real(sp) function [stddev_sp](#) (dat, mask)
- real(dp) function [skewness_dp](#) (dat, mask)
- real(sp) function [skewness_sp](#) (dat, mask)
- real(dp) function [variance_dp](#) (dat, mask)
- real(sp) function [variance_sp](#) (dat, mask)

15.23.1 Function/Subroutine Documentation

15.23.1.1 [absdev_dp\(\)](#)

```
real(dp) function mo_moment::absdev_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.2 `absdev_sp()`

```
real(sp) function mo_moment::absdev_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.3 `average_dp()`

```
real(dp) function mo_moment::average_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.4 `average_sp()`

```
real(sp) function mo_moment::average_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.5 `central_moment_dp()`

```
real(dp) function mo_moment::central_moment_dp (  
    real(dp), dimension(:), intent(in) x,  
    integer(i4), intent(in) r,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.6 `central_moment_sp()`

```
real(sp) function mo_moment::central_moment_sp (  
    real(sp), dimension(:), intent(in) x,  
    integer(i4), intent(in) r,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.7 `central_moment_var_dp()`

```
real(dp) function mo_moment::central_moment_var_dp (  
    real(dp), dimension(:), intent(in) x,  
    integer(i4), intent(in) r,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.8 `central_moment_var_sp()`

```
real(sp) function mo_moment::central_moment_var_sp (  
    real(sp), dimension(:), intent(in) x,  
    integer(i4), intent(in) r,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.9 correlation_dp()

```
real(dp) function mo_moment::correlation_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.10 correlation_sp()

```
real(sp) function mo_moment::correlation_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.11 covariance_dp()

```
real(dp) function mo_moment::covariance_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.12 covariance_sp()

```
real(sp) function mo_moment::covariance_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.13 kurtosis_dp()

```
real(dp) function mo_moment::kurtosis_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.14 kurtosis_sp()

```
real(sp) function mo_moment::kurtosis_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.15 mean_dp()

```
real(dp) function mo_moment::mean_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.16 mean_sp()

```
real(sp) function mo_moment::mean_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.17 mixed_central_moment_dp()

```
real(dp) function mo_moment::mixed_central_moment_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    integer(i4), intent(in) r,  
    integer(i4), intent(in) s,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.18 mixed_central_moment_sp()

```
real(sp) function mo_moment::mixed_central_moment_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    integer(i4), intent(in) r,  
    integer(i4), intent(in) s,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.19 mixed_central_moment_var_dp()

```
real(dp) function mo_moment::mixed_central_moment_var_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    integer(i4), intent(in) r,  
    integer(i4), intent(in) s,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.20 mixed_central_moment_var_sp()

```
real(sp) function mo_moment::mixed_central_moment_var_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    integer(i4), intent(in) r,  
    integer(i4), intent(in) s,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.21 moment_dp()

```
subroutine mo_moment::moment_dp (  
    real(dp), dimension(:), intent(in) dat,  
    real(dp), intent(out), optional average,  
    real(dp), intent(out), optional variance,  
    real(dp), intent(out), optional skewness,  
    real(dp), intent(out), optional kurtosis,  
    real(dp), intent(out), optional mean,  
    real(dp), intent(out), optional stddev,  
    real(dp), intent(out), optional absdev,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.22 moment_sp()

```
subroutine mo_moment::moment_sp (  
    real(sp), dimension(:), intent(in) dat,  
    real(sp), intent(out), optional average,  
    real(sp), intent(out), optional variance,  
    real(sp), intent(out), optional skewness,  
    real(sp), intent(out), optional kurtosis,  
    real(sp), intent(out), optional mean,  
    real(sp), intent(out), optional stddev,  
    real(sp), intent(out), optional absdev,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.23 skewness_dp()

```
real(dp) function mo_moment::skewness_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.24 skewness_sp()

```
real(sp) function mo_moment::skewness_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.25 stddev_dp()

```
real(dp) function mo_moment::stddev_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.26 stddev_sp()

```
real(sp) function mo_moment::stddev_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.27 variance_dp()

```
real(dp) function mo_moment::variance_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.23.1.28 variance_sp()

```
real(sp) function mo_moment::variance_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.24 mo_mpr_pet Module Reference

scaling function for PET correction using LAI at level-0

Functions/Subroutines

- subroutine, public [pet_correctbylai](#) (param, nodata, LCOVER0, LAI0, mask0, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, nLO_in_L1, L1_petLAIcorFactor)
estimate PET correction factor based on LAI at L1
- subroutine, public [pet_correctbyasp](#) (ld0, latitude_l0, Asp0, param, nodata, fAsp0)
correction of PET
- subroutine, public [priestley_taylor_alpha](#) (LCover_LAI0, LAI_LUT, LAIUnitList, param, mask0, nodata, cell_id0, nLO_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, priestley_taylor_alpha1)
Regionalization of priestley taylor alpha.
- subroutine, public [bulksurface_resistance](#) (LCover_LAI0, LAI_LUT, LAIUnitList, param, mask0, nodata, cell_id0, nLO_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, bulksurface_resistance1)
Regionalization of bulk surface resistance.

15.24.1 Detailed Description

scaling function for PET correction using LAI at level-0

This module sets up pet correction factor at level-1 based on LAI

Author

Mehmet Cuneyd Demirel, Simon Stisen

Date

May 2017

15.24.2 Function/Subroutine Documentation

15.24.2.1 bulksurface_resistance()

```
subroutine, public mo_mpr_pet::bulksurface_resistance (
    integer(i4), dimension(:), intent(in) LCover_LAI0,
    real(dp), dimension(:,:), intent(in) LAI_LUT,
    integer(i4), dimension(:), intent(in) LAIUnitList,
    real(dp), intent(in) param,
    logical, dimension(:,:), intent(in) mask0,
    real(dp), intent(in) nodata,
    integer(i4), dimension(:), intent(in) cell_id0,
    integer(i4), dimension(:), intent(in) nL0_in_L1,
    integer(i4), dimension(:), intent(in) Upp_row_L1,
    integer(i4), dimension(:), intent(in) Low_row_L1,
    integer(i4), dimension(:), intent(in) Lef_col_L1,
    integer(i4), dimension(:), intent(in) Rig_col_L1,
    real(dp), dimension(:,:), intent(out) bulksurface_resistance1 )
```

Regionalization of bulk surface resistance.

estimation of bulk surface resistance Global parameters needed (see mhm_parameter.nml):

- param(1) = stomatal_resistance

Parameters

in	<i>integer(i4) :: LCover_LAI0(:)</i>	- land cover id for LAI at level 0
in	<i>real(dp) :: LAI_LUT(:)</i>	- LUT of LAI values
in	<i>integer(i4) :: LAIUnitList(:)</i>	- List of ids of each LAI class in LAI_LUT
in	<i>real(dp) :: param</i>	- global parameter
in	<i>logical :: mask0(:, :)</i>	- mask at level 0 field
in	<i>real(dp) :: nodata</i>	- nodata value
in	<i>integer(i4) :: cell_id0 (:)</i>	- Cell ids at level 0
in	<i>integer(i4) :: nL0_in_L1 (:)</i>	- Number of L0 cells within a L1 cell
in	<i>integer(i4) :: Upp_row_L1(:)</i>	- Upper row of high resolution block
in	<i>integer(i4) :: Low_row_L1(:)</i>	- Lower row of high resolution block
in	<i>integer(i4) :: Lef_col_L1(:)</i>	- Left column of high resolution block
in	<i>integer(i4) :: Rig_col_L1(:)</i>	- Right column of high resolution block
out	<i>real(dp) :: bulksurface_resistance1(:)</i>	- [s m ⁻¹] bulk surface resistance McMahon et al, 2013: Estimating actual, potential, reference crop and pan evaporation using standard meteorological data: a pragmatic synthesis , HESS

Author

Matthias Zink

15.24.2.2 pet_correctbyasp()

```

subroutine, public mo_mpr_pet::pet_correctbyasp (
    integer(i4), dimension(:), intent(in) Id0,
    real(dp), dimension(:), intent(in) latitude_l0,
    real(dp), dimension(:), intent(in) Asp0,
    real(dp), dimension(3), intent(in) param,
    real(dp), intent(in) nodata,
    real(dp), dimension(:), intent(out) fAsp0 )

```

correction of PET

Correction of PET based on L0 aspect data.

```

Global parameters needed (see mhm_parameter.nml):\n
- param(1) = minCorrectionFactorPET \n
- param(2) = maxCorrectionFactorPET \n
- param(3) = aspectTresholdPET      \n

```

Parameters

in	<i>integer(i4) :: id0(:,.)</i>	- cell id at Level 0
in	<i>real(dp) :: nodata</i>	- no data value
in	<i>real(dp) :: param(3)</i>	- the three process parameters
out	<i>real(dp) :: fAsp0(:,.)</i>	- [1] PET correction factor for aspect

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

References mo_kind::i4.

Referenced by mo_multi_param_reg::mpr().

[illegible]

estimate PET correction factor based on LAI at L1

estimate PET correction factor based on LAI at L1 for a given Leaf Area Index field.

Global parameters needed (see mhm_parameter.nml):

Process Case 5:

- param(1) = PET_a_forest
- param(2) = PET_a_impervious
- param(3) = PET_a_pervious
- param(4) = PET_b
- param(5) = PET_c
 Example $DSF = PET_a + PET_b * (1 - \exp(PET_c * LAI))$ Similar to the crop coefficient concept $Kc = a + b * (1 - \exp(c * LAI))$ by Allen, R. G., L. S. Pereira, D. Raes, and M. Smith (1998), Crop evapotranspiration - Guidelines for computing crop water requirements., FAO Irrigation and drainage paper 56. See Chapter 9, Equation 97
<http://www.fao.org/docrep/X0490E/x0490e0f.htm> Date: 17/5/2017

Parameters

in	<i>integer(i4) :: cell_id0(:)</i>	- Cell ids at level 0
in	<i>real(dp) :: LCOVER0(:)</i>	- Landcover at level-0
in	<i>real(dp) :: LAI0(:)</i>	- LAI at level-0
in	<i>real(dp) :: param(:)</i>	- array of global parameters
in	<i>real(dp) :: nodata</i>	- nodata value
in	<i>integer(i4) :: L0_cell_id(:, :)</i>	- cell ids of high resolution field, Number of rows times Number of columns of high resolution field
in	<i>integer(i4) :: upp_row_L1(:)</i>	- Upper row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: low_row_L1(:)</i>	- Lower row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: lef_col_L1(:)</i>	- Left column id in high resolution field (L0) of low resolution cell
in	<i>integer(i4) :: rig_col_L1(:)</i>	- Right column id in high resolution field (L0) of low resolution cell
in	<i>integer(i4) :: nL0_in_L1(:)</i>	- Number of high resolution cells (L0) in low resolution cell (L1 cell)
in, out	<i>real(dp) :: L1_petLAIcorFactor(:)</i>	- PET correction using LAI.

Author

M. Cuneyd Demirel and Simon Stisen from GEUS.dk

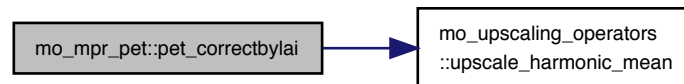
Date

May, 2017

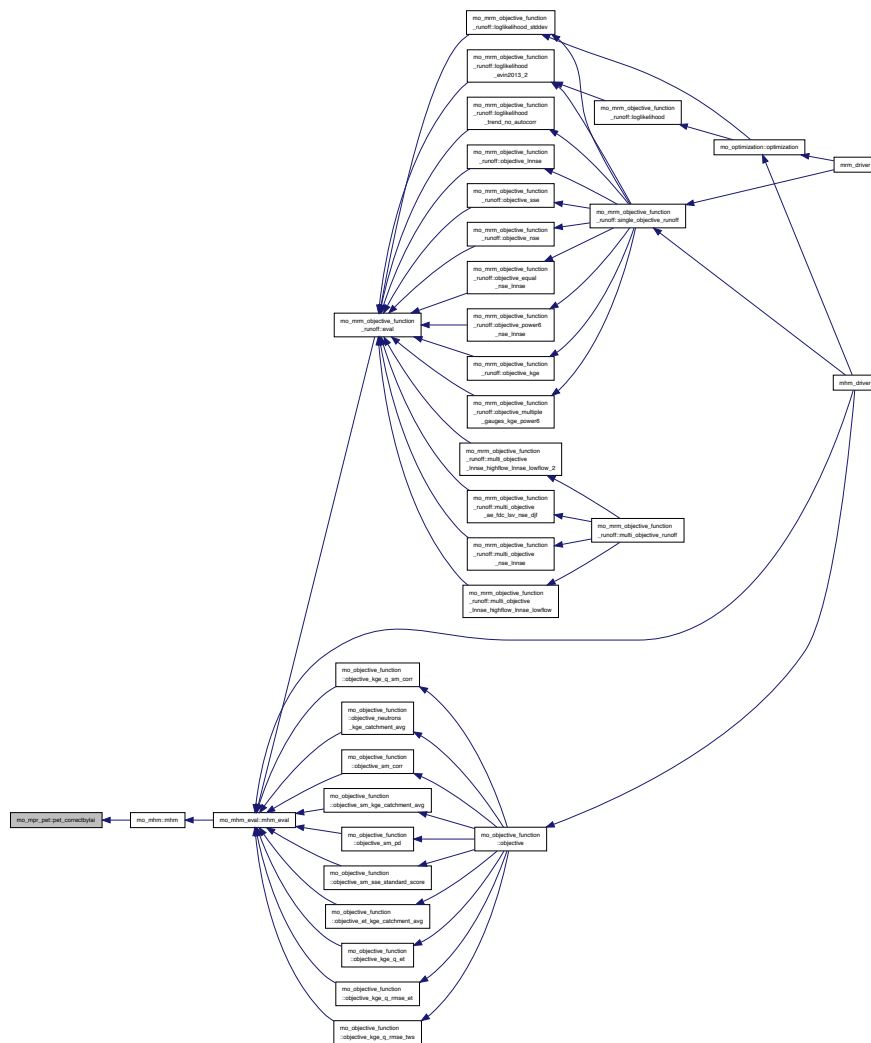
References `mo_upscaling_operators::upscale_harmonic_mean()`.

Referenced by `mo_mhm::mhm()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.24.2.4 priestley_taylor_alpha()

```

subroutine, public mo_mpr_pet::priestley_taylor_alpha (
    integer(i4), dimension(:), intent(in) LCover_LAI0,
    real(dp), dimension(:,:), intent(in) LAI_LUT,
    integer(i4), dimension(:), intent(in) LAI_UnitList,
    real(dp), dimension(:), intent(in) param,
    logical, dimension(:,:), intent(in) mask0,
    real(dp), intent(in) nodata,
    integer(i4), dimension(:), intent(in) cell_id0,
    integer(i4), dimension(:), intent(in) nL0_in_L1,
    integer(i4), dimension(:), intent(in) Upp_row_L1,
    integer(i4), dimension(:), intent(in) Low_row_L1,
    integer(i4), dimension(:), intent(in) Lef_col_L1,
    integer(i4), dimension(:), intent(in) Rig_col_L1,
    real(dp), dimension(:,:), intent(out) priestley_taylor_alpha1 )

```

Regionalization of priestley taylor alpha.

estimation of priestley taylor alpha Global parameters needed (see mhm_parameter.nml):

- param(1) = PriestleyTaylorCoeff
- param(2) = PriestleyTaylorLAIcorr

Parameters

in	<i>integer(i4) :: LCover_LAI0(:)</i>	- land cover id for LAI at level 0
in	<i>real(dp) :: LAI_LUT(:)</i>	- LUT of LAI values
in	<i>integer(i4) :: LAI_UnitList(:)</i>	- List of ids of each LAI class in LAI_LUT
in	<i>real(dp) :: param(:)</i>	- global parameter
in	<i>logical :: mask0(:, :)</i>	- mask at level 0 field
in	<i>real(dp) :: nodata</i>	- nodata value
in	<i>integer(i4) :: cell_id0 (:)</i>	- Cell ids at level 0
in	<i>integer(i4) :: nL0_in_L1 (:)</i>	- Number of L0 cells within a L1 cell
in	<i>integer(i4) :: Upp_row_L1(:)</i>	- Upper row of high resolution block
in	<i>integer(i4) :: Low_row_L1(:)</i>	- Lower row of high resolution block
in	<i>integer(i4) :: Lef_col_L1(:)</i>	- Left column of high resolution block
in	<i>integer(i4) :: Rig_col_L1(:)</i>	- Right column of high resolution block
out	<i>real(dp) :: priestley_taylor_alpha1(:)</i>	- [s m ⁻¹] bulk surface resistance

Author

Matthias Zink

Date

Apr 2013

References mo_constants::eps_dp, mo_upscaling_operators::upscale_arithmetic_mean(), and mo_mhm_constants::yearmonths_i4.

Functions/Subroutines

- subroutine, public [mpr_runoff](#) (LCOVER0, mask0, nodata, SMs_FC0, slope_emp0, KsVar_H0, param, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, nL0_in_L1, c2TSTu, L1_HL1, L1_K0, L1_K1, L1_alpha)

multiscale parameter regionalization for runoff parameters

15.25.1 Detailed Description

multiscale parameter regionalization for runoff generation

This contains the routine for multiscale parameter regionalization of the runoff parametrization.

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

15.25.2 Function/Subroutine Documentation

15.25.2.1 mpr_runoff()

```
subroutine, public mo_mpr_runoff::mpr_runoff (
    integer(i4), dimension(:), intent(in) LCOVER0,
    logical, dimension(:,:), intent(in) mask0,
    real(dp), intent(in) nodata,
    real(dp), dimension(:), intent(in) SMs_FC0,
    real(dp), dimension(:), intent(in) slope_emp0,
    real(dp), dimension(:), intent(in) KsVar_H0,
    real(dp), dimension(5), intent(in) param,
    integer(i4), dimension(:), intent(in) cell_id0,
    integer(i4), dimension(:), intent(in) upp_row_L1,
    integer(i4), dimension(:), intent(in) low_row_L1,
    integer(i4), dimension(:), intent(in) lef_col_L1,
    integer(i4), dimension(:), intent(in) rig_col_L1,
    integer(i4), dimension(:), intent(in) nL0_in_L1,
    real(dp), intent(in) c2TSTu,
    real(dp), dimension(:), intent(out) L1_HL1,
    real(dp), dimension(:), intent(out) L1_K0,
    real(dp), dimension(:), intent(out) L1_K1,
    real(dp), dimension(:), intent(out) L1_alpha )
```

multiscale parameter regionalization for runoff parameters

Perform the multiscale parameter regionalization for runoff global parameters (see mhm_parameter.nml). These are the following five parameters:

- param(1) = interflowStorageCapacityFactor
- param(2) = interflowRecession_slope

- param(3) = fastInterflowRecession_forest
- param(4) = slowInterflowRecession_Ks
- param(5) = exponentSlowInterflow

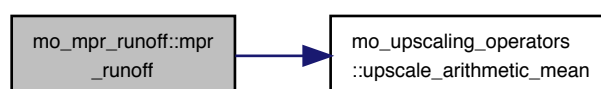
Parameters

in	<i>real(dp), dimension(5) :: param</i>	global parameters
in	<i>real(dp) :: nodata</i>	nodata value
in	<i>real(dp) :: SMs_FC0(:)</i>	[-] soil moisture deficit from field capacity w.r.t to saturation
in	<i>real(dp) :: slope_emp0(:)</i>	empirical quantile values F(slope)
in	<i>real(dp) :: KsVar_H0(:)</i>	[-] relative variability of saturated hydraulic conductivity for Horizontal flow at level 0
in	<i>integer(i4) :: LCOVER0(:)</i>	land cover at level 0
in	<i>logical :: mask0(:, :)</i>	mask at Level 0
in	<i>integer(i4) :: cell_id0(:)</i>	Cell ids of hi res field
in	<i>integer(i4) :: upp_row_L1(:)</i>	Upper row of hi res block
in	<i>integer(i4) :: low_row_L1(:)</i>	Lower row of hi res block
in	<i>integer(i4) :: lef_col_L1(:)</i>	Left column of hi res block
in	<i>integer(i4) :: rig_col_L1(:)</i>	Right column of hi res block
in	<i>integer(i4) :: nL0_in_L1(:)</i>	Number of L0 cells within a L1 cell
in	<i>real(dp) :: c2TSTu</i>	unit transformations
out	<i>real(dp) :: L1_HL1(:)</i>	[10 ⁻³ m] Threshold water depth in upper reservoir (for Runoff contribution)
out	<i>real(dp) :: L1_K0(:)</i>	[10 ⁻³ m] Recession coefficient of the upper reservoir, upper outlet
out	<i>real(dp) :: L1_K1(:)</i>	[10 ⁻³ m] Recession coefficient of the upper reservoir, lower outlet
out	<i>real(dp) :: L1_alpha(:)</i>	[1] Exponent for the upper reservoir

References mo_kind::i4, and mo_upscaling_operators::upscale_arithmetic_mean().

Referenced by mo_multi_param_reg::mpr().

Here is the call graph for this function:



[illegible]

Generated on December 1, 2017

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

15.26.2 Function/Subroutine Documentation**15.26.2.1 mpr_smhorizons()**

```

subroutine, public mo_mpr_smhorizons::mpr_smhorizons (
    real(dp), dimension(:), intent(in) param,
    integer(i4), dimension(:,:), intent(in) processMatrix,
    real(dp), intent(in) nodata,
    integer(i4), intent(in) iFlag_soil,
    integer(i4), intent(in) nHorizons_mHM,
    real(dp), dimension(:), intent(in) HorizonDepth,
    integer(i4), dimension(:), intent(in) LCOVER0,
    integer(i4), dimension(:,:), intent(in) soilID0,
    integer(i4), dimension(:), intent(in) nHorizons,
    integer(i4), dimension(:), intent(in) nTillHorizons,
    real(dp), dimension(:,:,:), intent(in) thetaS_till,
    real(dp), dimension(:,:,:), intent(in) thetaFC_till,
    real(dp), dimension(:,:,:), intent(in) thetaPW_till,
    real(dp), dimension(:,:), intent(in) thetaS,
    real(dp), dimension(:,:), intent(in) thetaFC,
    real(dp), dimension(:,:), intent(in) thetaPW,
    real(dp), dimension(:,:,:), intent(in) Wd,
    real(dp), dimension(:,:,:), intent(in) Db,
    real(dp), dimension(:,:), intent(in) DbM,
    real(dp), dimension(:), intent(in) RZdepth,
    logical, dimension(:,:), intent(in) mask0,
    integer(i4), dimension(:), intent(in) cell_id0,
    integer(i4), dimension(:), intent(in) upp_row_L1,
    integer(i4), dimension(:), intent(in) low_row_L1,
    integer(i4), dimension(:), intent(in) lef_col_L1,
    integer(i4), dimension(:), intent(in) rig_col_L1,
    integer(i4), dimension(:), intent(in) nL0_in_L1,
    real(dp), dimension(:,:), intent(inout) L1_beta,
    real(dp), dimension(:,:), intent(inout) L1_SMs,
    real(dp), dimension(:,:), intent(inout) L1_FC,
    real(dp), dimension(:,:), intent(inout) L1_PW,
    real(dp), dimension(:,:), intent(inout) L1_fRoots )

```

upscale soil moisture horizons

calculate soil properties at the level 1.

Global parameters needed (see mhm_parameter.nml):

- param(1) = rootFractionCoefficient_forest
- param(2) = rootFractionCoefficient_impervious

- param(3) = rootFractionCoefficient_pervious
- param(4) = infiltrationShapeFactor

Parameters

in	<i>real(dp) :: param(:)</i>	- four or six global parameters depending on SM process
in	<i>integer(i4) :: processMatrix</i>	- matrix specifying user defined processes
in	<i>real(dp) :: nodata</i>	- no data value
in	<i>integer(i4) :: iFlag_soil</i>	- flags for handling multiple soil databases
in	<i>integer(i4) :: nHorizons_mHM</i>	- number of horizons to model
in	<i>integer(i4) :: HorizonDepth(:)</i>	- [mm] horizon depth from surface, postive downwards
in	<i>integer(i4) :: L0_LUC(:, :)</i>	- land use cover at level 0
in	<i>integer(i4) :: L0_soilID(:, :)</i>	- soil IDs at level 0
in	<i>integer(i4) :: nHorizons(:)</i>	- horizons per soil type
in	<i>integer(i4) :: nTillHorizons(:)</i>	- Number of Tillage horizons
in	<i>real(dp) :: thetaS_till(:, :, :)</i>	- saturated water content of soil horizons upto tillage depth, f(OM, management)
in	<i>real(dp) :: thetaFC_till(:, :, :)</i>	- Field capacity of tillage layers; LUC dependent, f(OM, management)
in	<i>real(dp) :: thetaPW_till(:, :, :)</i>	- Permanent wilting point of tillage layers; LUC dependent, f(OM, management)
in	<i>real(dp) :: thetaS(:, :)</i>	- saturated water content of soil horizons after tillage depth
in	<i>real(dp) :: thetaFC(:, :)</i>	- Field capacity of deeper layers
in	<i>real(dp) :: thetaPW(:, :)</i>	- Permanent wilting point of deeper layers
in	<i>real(dp) :: Wd(:, :, :)</i>	- weights of mHM Horizons according to horizons provided in soil database
in	<i>real(dp) :: Db(:, :, :)</i>	- Bulk density
in	<i>real(dp) :: DbM(:, :)</i>	- mineral Bulk density
in	<i>real(dp) :: RZdepth(:)</i>	- [mm] Total soil depth
in	<i>integer(i4) :: L0_cellCoor(:, :)</i>	- cell coordinates at level 0
in	<i>integer(i4) :: L0_cell_id(:, :)</i>	- cell ids of high resolution field, Number of rows times Number of columns of high resolution field
in	<i>integer(i4) :: upp_row_L1(:)</i>	- Upper row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: low_row_L1(:)</i>	- Lower row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: lef_col_L1(:)</i>	- Left column id in high resolution field (L0) of low resolution cell
in	<i>integer(i4) :: rig_col_L1(:)</i>	- Right column id in high resolution field (L0) of low resolution cell
in	<i>integer(i4) :: nL0_in_L1(:)</i>	- Number of high resolution cells (L0) in low resolution cell (L1 cell)
in, out	<i>real(dp) :: L1_beta(:, :)</i>	- Parameter that determines the relative contribution to SM, upscaled Bulk density. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: L1_SMs(:, :)</i>	- [10 ⁻³ m] depth of saturated SM cont Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: L1_FC(:, :)</i>	- [10 ⁻³ m] field capacity. Number of cells at L1 times number of horizons in mHM

Parameters

in, out	<i>real(dp) :: L1_PW(:, :)</i>	- [10 ⁻³ m] permanent wilting point. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: L1_fRoots(:, :)</i>	- fraction of roots in soil horizons. Number of cells at L1 times number of horizons in mHM

Author

Luis Samaniego, Rohini Kumar, Stephan Thober

Date

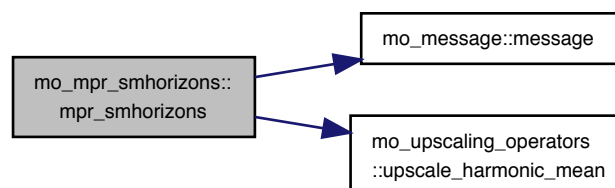
Dec 2012

in this case the second dimension of `soilId0 = 1`

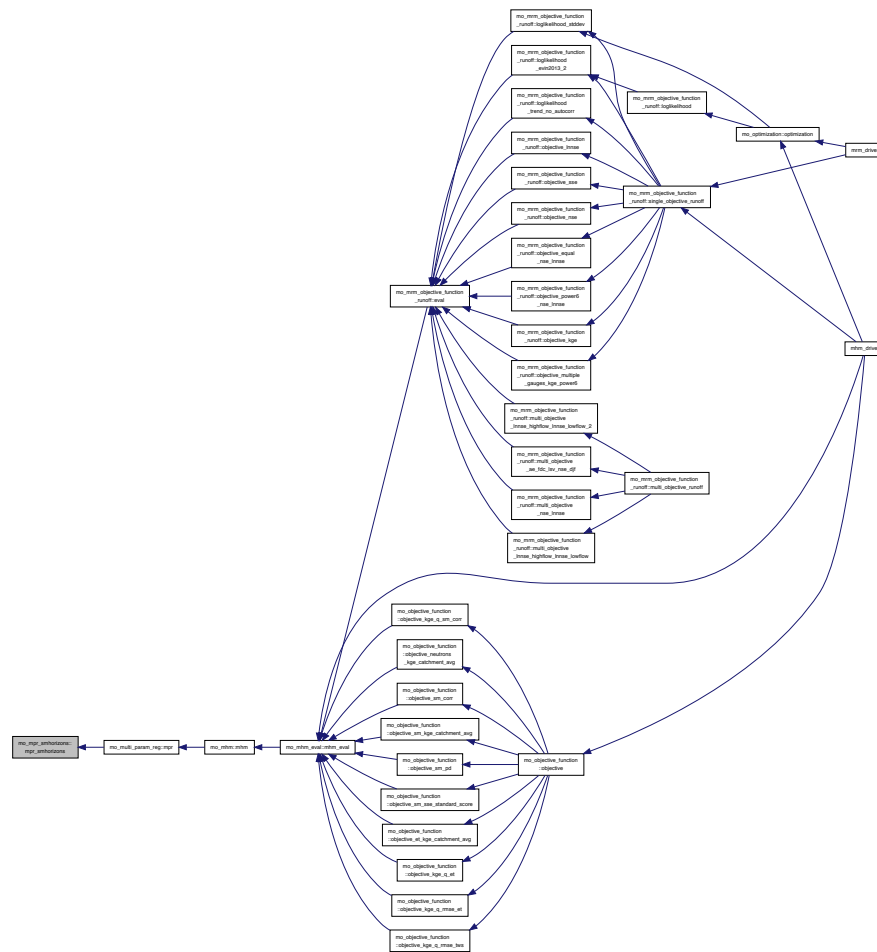
References `mo_message::message()`, and `mo_upscaling_operators::upscale_harmonic_mean()`.

Referenced by `mo_multi_param_reg::mpr()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.27 mo_mpr_soilmoist Module Reference

Multiscale parameter regionalization (MPR) for soil moisture.

Functions/Subroutines

- subroutine, public [mpr_sm](#) (param, nodata, iFlag_soil, is_present, nHorizons, nTillHorizons, sand, clay, DbM, ID0, soilld0, LCover0, thetaS_till, thetaFC_till, thetaPW_till, thetaS, thetaFC, thetaPW, Ks, Db, KsVar_H0, KsVar_V0, SMs_FC0)

multiscale parameter regionalization for soil moisture

- elemental pure subroutine [pwp](#) (Genu_Mual_n, Genu_Mual_alpha, thetaS, thetaPWP)

Permanent Wilting point.

- elemental pure subroutine [field_cap](#) (thetaFC, Ks, thetaS, Genu_Mual_n)

calculates the field capacity

- subroutine [genuchten](#) (thetaS, Genu_Mual_n, Genu_Mual_alpha, param, sand, clay, Db)

calculates the Genuchten shape parameter

- subroutine [hydro_cond](#) (KS, param, sand, clay)

calculates the hydraulic conductivity Ks

15.27.1 Detailed Description

Multiscale parameter regionalization (MPR) for soil moisture.

This module contains all routines required for parametrizing soil moisture processes.

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

15.27.2 Function/Subroutine Documentation

15.27.2.1 field_cap()

```
elemental pure subroutine mo_mpr_soilmoist::field_cap (
    real(dp), intent(out) thetaFC,
    real(dp), intent(in) Ks,
    real(dp), intent(in) thetaS,
    real(dp), intent(in) Genu_Mual_n )
```

calculates the field capacity

estimate Field capacity; FC – Flux based approach (Twarakavi, et. al. 2009, WRR)

According to the above reference FC is defined as the soil water content at which the drainage from a profile ceases under natural conditions. Since drainage from a soil profile in a simulation never becomes zero, we assume that drainage ceases when the bottom flux from the soil reaches a value that is equivalent to the minimum amount of precipitation that could be recorded (i.e. 0.01 cm/d == 1 mm/d). It is assumed that ThetaR = 0.0_dp

Parameters

in	<i>real(dp) :: Ks</i>	- saturated hydraulic conductivity
in	<i>real(dp) :: thetaS</i>	- saturated water content
in	<i>real(dp) :: Genu_Mual_n</i>	- Genuchten shape parameter
out	<i>real(dp) :: thetaFC</i>	- Field capacity

Author

Stephan Thober, Rohini Kumar

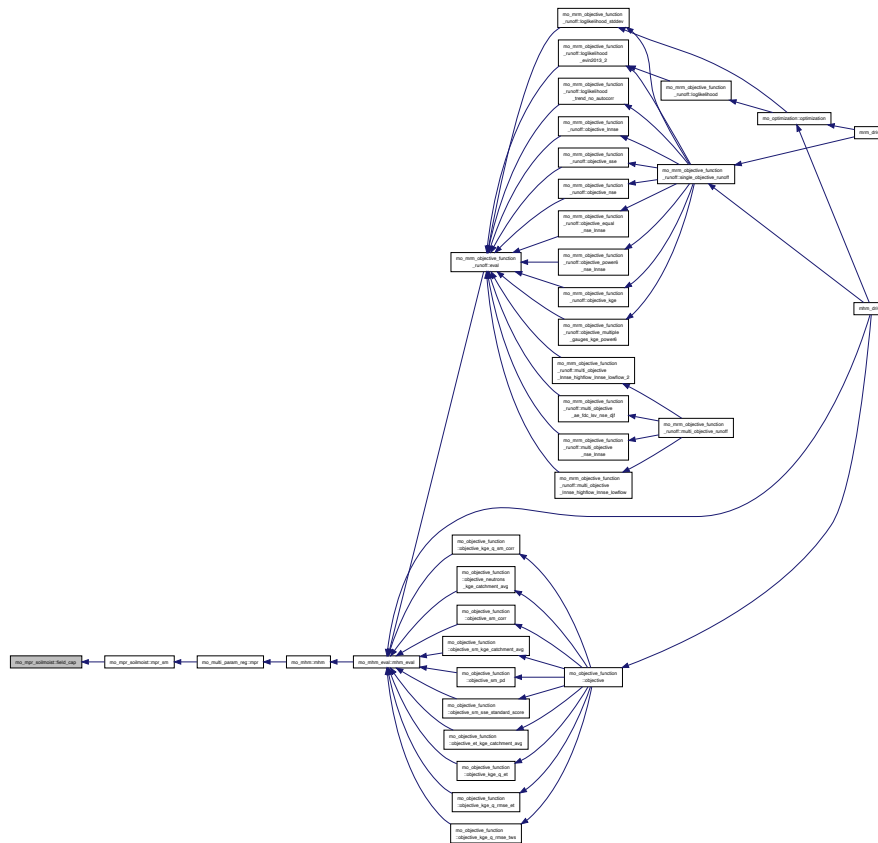
Date

Dec 2012

References mo_mhm_constants::field_cap_c1, and mo_mhm_constants::field_cap_c2.

Referenced by mpr_sm().

Here is the caller graph for this function:



15.27.2.2 genuchten()

```

subroutine mo_mpr_soilmoist::genuchten (
    real(dp), intent(out) thetaS,
    real(dp), intent(out) Genu_Mual_n,
    real(dp), intent(out) Genu_Mual_alpha,
    real(dp), dimension(6), intent(in) param,
    real(dp), intent(in) sand,
    real(dp), intent(in) clay,
    real(dp), intent(in) Db )

```

calculates the Genuchten shape parameter

estimate SMs_till & van Genuchten's shape parameter (n) (Zacharias et al, 2007, soil Phy.)

Global parameters needed (see mhm_parameter.nml):

- param(1) = PTF_lower66_5_constant

- `param(2) = PTF_lower66_5_clay`
- `param(3) = PTF_lower66_5_Db`
- `param(4) = PTF_higher66_5_constant`
- `param(5) = PTF_higher66_5_clay`
- `param(6) = PTF_higher66_5_Db`

Parameters

in	<i>real(dp) :: param(6)</i>	- given parameters
in	<i>real(dp) :: sand</i>	- [%] sand content
in	<i>real(dp) :: clay</i>	- [%] clay content
in	<i>real(dp) :: Db</i>	- [10^3 kg/m ³] bulk density
out	<i>real(dp) :: thetaS</i>	- saturated water content
out	<i>real(dp) :: Genu_Mual_n</i>	- van Genuchten shape parameter
out	<i>real(dp) :: Genu_Mual_alpha</i>	- van Genuchten shape parameter

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

References `mo_mhm_constants::vgenuchten_sandtresh`.

Referenced by `mpr_sm()`.

The graph illustrates the internal structure and dependencies of the 'ms' package. It features a central node 'ms' at the top, which is connected to numerous other nodes. These nodes are organized into several columns, each representing a different function or variable. The nodes are connected by directed edges, indicating the flow of information or dependencies. The graph is a visual representation of the internal structure and dependencies of the 'ms' package.

```

subroutine mo_mpr_soilmoist::hydro_cond (
    real(dp), intent(out) KS,
    real(dp), dimension(4), intent(in) param,
    real(dp), intent(in) sand,
    real(dp), intent(in) clay )

```

By default save this value of Ks, particularly for the deeper layers where OM content plays relatively low or no role

Global parameters needed (see mhm_parameter.nml):

- Generated on December 1, 2017


```

integer(i4), intent(in) iFlag_soil,
integer(i4), dimension(:), intent(in) is_present,
integer(i4), dimension(:), intent(in) nHorizons,
integer(i4), dimension(:), intent(in) nTillHorizons,
real(dp), dimension(:, :), intent(in) sand,
real(dp), dimension(:, :), intent(in) clay,
real(dp), dimension(:, :), intent(in) DbM,
integer(i4), dimension(:), intent(in) ID0,
integer(i4), dimension(:, :), intent(in) soilId0,
integer(i4), dimension(:), intent(in) LCover0,
real(dp), dimension(:, :, :), intent(out) thetaS_till,
real(dp), dimension(:, :, :), intent(out) thetaFC_till,
real(dp), dimension(:, :, :), intent(out) thetaPW_till,
real(dp), dimension(:, :), intent(out) thetaS,
real(dp), dimension(:, :), intent(out) thetaFC,
real(dp), dimension(:, :), intent(out) thetaPW,
real(dp), dimension(:, :, :), intent(out) Ks,
real(dp), dimension(:, :, :), intent(out) Db,
real(dp), dimension(:), intent(out) KsVar_H0,
real(dp), dimension(:), intent(out) KsVar_V0,
real(dp), dimension(:), intent(out) SMS_FC0 )

```

multiscale parameter regionalization for soil moisture

This subroutine is a wrapper around all soil moisture parameter routines. This subroutine requires 13 parameters. These parameters have to correspond to the parameters in the original parameter array at the following locations: 10-12, 13-18, 27-30.

Global parameters needed (see mhm_parameter.nml):

- param(1) = orgMatterContent_forest
- param(2) = orgMatterContent_impervious
- param(3) = orgMatterContent_pervious
- param(4) = PTF_lower66_5_constant
- param(5) = PTF_lower66_5_clay
- param(6) = PTF_lower66_5_Db
- param(7) = PTF_higher66_5_constant
- param(8) = PTF_higher66_5_clay
- param(9) = PTF_higher66_5_Db
- param(10) = PTF_Ks_constant
- param(11) = PTF_Ks_sand

- param(12) = PTF_Ks_clay
- param(13) = PTF_Ks_curveSlope

Parameters

in	<i>real(dp) :: param(13)</i>	- global parameters
in	<i>real(dp) :: nodata</i>	- no data value
in	<i>integer(i4) :: iFlag_soil</i>	- flags for handling multiple soil databases
in	<i>integer(i4) :: is_present(:)</i>	- indicates whether soiltype is present
in	<i>integer(i4) :: nHorizons(:)</i>	- Number of Horizons per soiltype2
in	<i>integer(i4) :: nTillHorizons(:)</i>	- Number of Tillage Horizons
in	<i>real(dp) :: sand(:,:)</i>	- sand content
in	<i>real(dp) :: clay(:,:)</i>	- clay content
in	<i>real(dp) :: DbM(:,:)</i>	- mineral Bulk density
in	<i>integer(i4) :: L0_ID(:,:)</i>	- cell ids at level 0
in	<i>integer(i4) :: L0_soilId(:,:)</i>	- soil ids at level 0
in	<i>integer(i4) :: L0_LUC(:,:)</i>	- land cover ids at level 0
out	<i>real(dp) :: thetaS_till(:,:,:)</i>	- saturated soil moisture tillage layer
out	<i>real(dp) :: thetaFC_till(:,:,:)</i>	- field capacity tillage layer
out	<i>real(dp) :: thetaPW_till(:,:,:)</i>	- permanent wilting point tillage layer
out	<i>real(dp) :: thetaS(:,:)</i>	- saturated soil moisture
out	<i>real(dp) :: thetaFC(:,:)</i>	- field capacity
out	<i>real(dp) :: thetaPW(:,:)</i>	- permanent wilting point
out	<i>real(dp) :: Ks(:,:,:)</i>	- saturated hydraulic conductivity
out	<i>real(dp) :: Db(:,:,:)</i>	- Bulk density
out	<i>real(dp) :: L0_KsVar_H(:,:)</i>	- relative variability of saturated hydraulic conductivity for Horizontal flow
out	<i>real(dp) :: L0_KsVar_V(:,:)</i>	- relative variability of saturated hydraulic conductivity for Vertical flow
out	<i>real(dp) :: L0_SMs_FC(:,:)</i>	- soil moisture deficit from field capacity w.r.t to saturation

Author

Stephan Thober, Rohini Kumar

Date

Dec 2012

here = ncells0

in this case the second dimension of soilId0 = 1

non-till

till layers

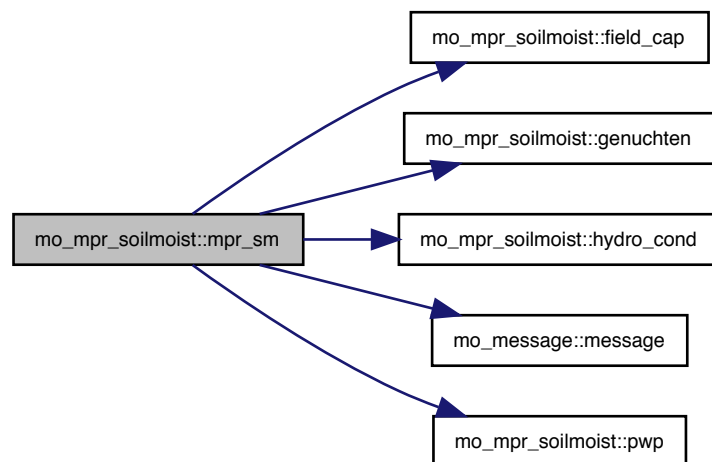
HORIZON

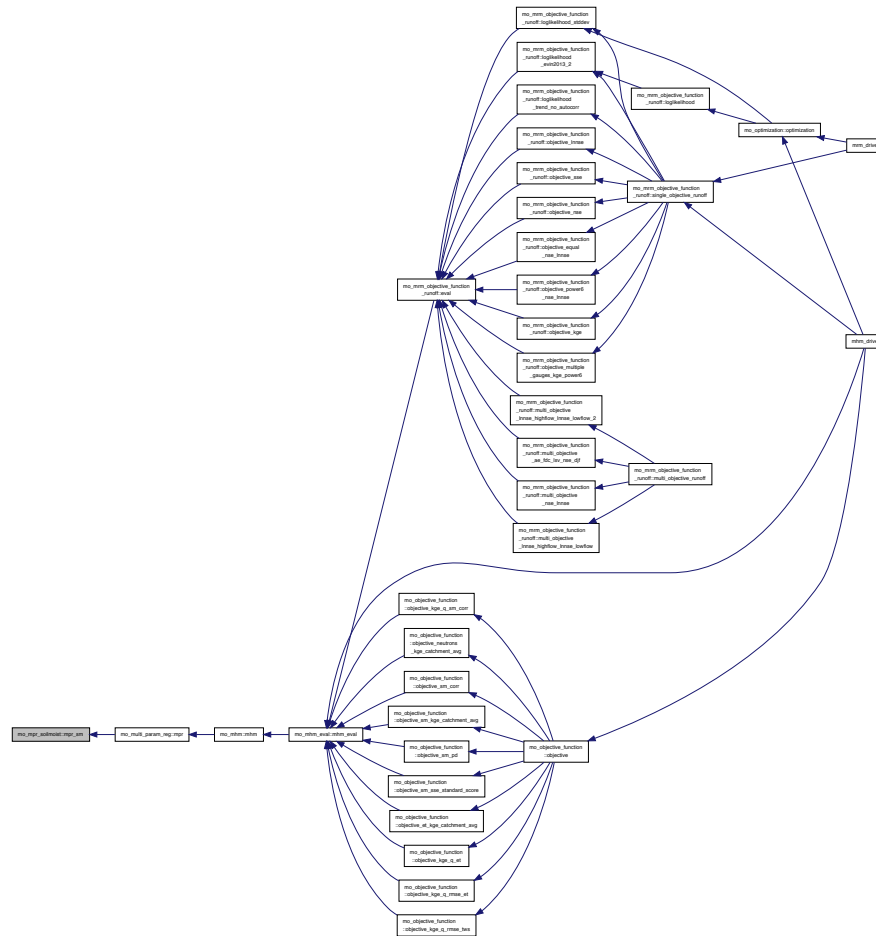
SOIL TYPE

References mo_mhm_constants::bulkdens_orgmatter, field_cap(), genuchten(), hydro_cond(), mo_kind::i4, mo_message::message(), and pwp().

Referenced by mo_multi_param_reg::mpr().

Here is the call graph for this function:





```

elemental pure subroutine mo_mpr_soilmoist::pwp (
    real(dp), intent(in)  Genu_Mual_n,
    real(dp), intent(in)  Genu_Mual_alpha,
    real(dp), intent(in)  thetaS,
    real(dp), intent(out) thetaPWP )

```

This subroutine calculates the permanent wilting point according to Zacharias et al. (2007, Soil Phy.) and using van Genuchten 1980's equation. For the water retention curve at a matrix potential of -1500 kPa, it is assumed that $\theta_R = 0$.

in	$real(dp) :: Genu_Mual_n$	- Genuchten shape parameter
in	$real(dp) :: Genu_Mual_alpha$	- Genuchten shape parameter
in	$real(dp) :: thetaS$	- saturated water content
out	$real(dp) :: thetaPWP$	- Permanent Wilting point

Author

Stephan Thober, Rohini Kumar

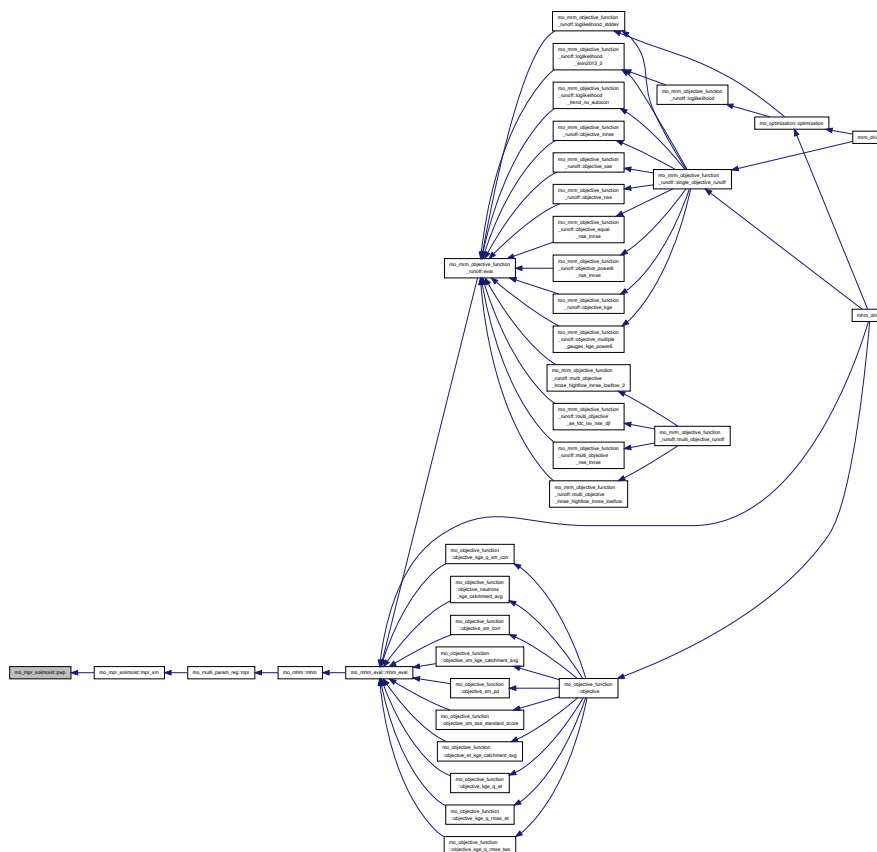
Date

Dec, 2012

References mo_mhm_constants::pwp_c, and mo_mhm_constants::pwp_matpot_thetar.

Referenced by mpr_sm().

Here is the caller graph for this function:



15.28 mo_mrm_constants Module Reference

Variables

- integer(i4), parameter, public `noutfluxstate` = 1_i4
- integer(i4), parameter, public `nodata_i4` = -9999_i4
- real(dp), parameter, public `nodata_dp` = -9999._dp
- integer(i4), parameter, public `nroutingstates` = 2
- integer(i4), parameter, public `ncolpars` = 5_i4
- integer(i4), parameter, public `maxnogauges` = 50_i4
- integer(i4), parameter, public `maxnobasins` = 50_i4
- integer(i4), parameter, public `maxnrcovers` = 50_i4
- real(dp), parameter, public `hoursecs` = 3600.0_dp

- `real(dp), parameter, public p1_initstatefluxes = 0.00_dp`
- `real(dp), parameter, public rout_space_weight = 0._dp`
- `real(dp), parameter, public deltah = 5.000_dp`
- `real(dp), dimension(19), parameter given_ts = (/ 60._dp, 120._dp, 180._dp, 240._dp, 300._dp, 360._dp, 600._dp, 720._dp, 900._dp, 1200._dp, 1800._dp, 3600._dp, 7200._dp, 10800._dp, 14400._dp, 21600._dp, 28800._dp, 43200._dp, 86400._dp/)`

15.28.1 Variable Documentation

15.28.1.1 deltah

`real(dp), parameter, public mo_mrm_constants::deltah = 5.000_dp`

Referenced by `mo_mrm_net_startup::moveup()`.

15.28.1.2 given_ts

`real(dp), dimension(19), parameter mo_mrm_constants::given_ts = (/ 60._dp, 120._dp, 180._dp, 240._dp, 300._dp, 360._dp, 600._dp, 720._dp, 900._dp, 1200._dp, 1800._dp, 3600._dp, 7200._dp, 10800._dp, 14400._dp, 21600._dp, 28800._dp, 43200._dp, 86400._dp/)`

Referenced by `mo_mrm_init::mrm_init_param()`, and `mo_mrm_init::mrm_update_param()`.

15.28.1.3 hoursecs

`real(dp), parameter, public mo_mrm_constants::hoursecs = 3600.0_dp`

Referenced by `mo_mrm_routing::l11_runoff_acc()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_eval::mrm_eval()`, `mo_mrm_init::mrm_init_param()`, and `mo_mrm_read_data::mrm_read_total_runoff()`.

15.28.1.4 maxnlcovers

`integer(i4), parameter, public mo_mrm_constants::maxnlcovers = 50_i4`

15.28.1.5 maxnobasins

`integer(i4), parameter, public mo_mrm_constants::maxnobasins = 50_i4`

15.28.1.6 maxnogauges

`integer(i4), parameter, public mo_mrm_constants::maxnogauges = 50_i4`

Referenced by `mo_mrm_read_config::read_mrm_config()`.

15.28.1.7 ncolpars

```
integer(i4), parameter, public mo_mrm_constants::ncolpars = 5_i4
```

Referenced by mo_mrm_read_config::read_mrm_routing_params().

15.28.1.8 nodata_dp

```
real(dp), parameter, public mo_mrm_constants::nodata_dp = -9999._dp
```

Referenced by mo_objective_function::extract_basin_avg_tws(), mo_mrm_net_startup::l11_fraction_sealed_floodplain(), mo_mrm_routing::l11_runoff_acc(), mo_mrm_net_startup::l11_stream_features(), mo_mrm_net_startup::l11_variable_init(), mo_mrm_read_data::mrm_l0_variable_init(), mo_mrm_read_data::mrm_l1_variable_init(), mo_mrm_read_data::mrm_read_discharge(), mo_mrm_read_data::mrm_read_l0_data(), mo_mrm_restart::mrm_read_restart_config(), mo_mrm_read_data::mrm_read_total_runoff(), mo_mrm_restart::mrm_write_restart(), mo_objective_function::objective_kge_q_rmse_et(), mo_objective_function::objective_kge_q_rmse_tws(), mo_mrm_write::write_configfile(), mo_mrm_write_fluxes_states::writevariableattributes(), and mo_mrm_write_fluxes_states::writevariabletimestep().

15.28.1.9 nodata_i4

```
integer(i4), parameter, public mo_mrm_constants::nodata_i4 = -9999_i4
```

Referenced by mo_mrm_init::l0_check_input_routing(), mo_mrm_net_startup::l11_flow_direction(), mo_mrm_net_startup::l11_link_location(), mo_mrm_net_startup::l11_routing_order(), mo_mrm_net_startup::l11_set_drain_outlet_gauges(), mo_mrm_net_startup::l11_set_network_topology(), mo_mrm_net_startup::l11_stream_features(), mo_mrm_net_startup::l11_variable_init(), mo_mrm_init::mrm_init(), mo_mrm_read_data::mrm_l0_variable_init(), mo_mrm_read_data::mrm_read_l0_data(), mo_mrm_restart::mrm_read_restart_config(), mo_mrm_restart::mrm_write_restart(), mo_mrm_read_config::read_mrm_config(), and mo_mrm_read_data::rotate_fdir_variable().

15.28.1.10 noutflxstate

```
integer(i4), parameter, public mo_mrm_constants::noutflxstate = 1_i4
```

15.28.1.11 nroutingstates

```
integer(i4), parameter, public mo_mrm_constants::nroutingstates = 2
```

Referenced by mo_mrm_restart::mrm_read_restart_states(), mo_mrm_restart::mrm_write_restart(), and mo_mrm_init::variables_alloc_routing().

15.28.1.12 p1_initstatefluxes

```
real(dp), parameter, public mo_mrm_constants::p1_initstatefluxes = 0.00_dp
```

Referenced by mo_mrm_init::variables_default_init_routing().

15.28.1.13 rout_space_weight

```
real(dp), parameter, public mo_mrm_constants::rout_space_weight = 0._dp
```

Referenced by mo_mrm_init::mrm_update_param().

15.29 mo_mrm_eval Module Reference

Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

Functions/Subroutines

- subroutine, public [mrm_eval](#) (parameterset, runoff)
Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

15.29.1 Detailed Description

Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

Authors

Stephan Thober

Date

Sep 2015

15.29.2 Function/Subroutine Documentation

15.29.2.1 mrm_eval()

```
subroutine, public mo_mrm_eval::mrm_eval (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), dimension(:, :), intent(out), optional, allocatable runoff )
```

Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

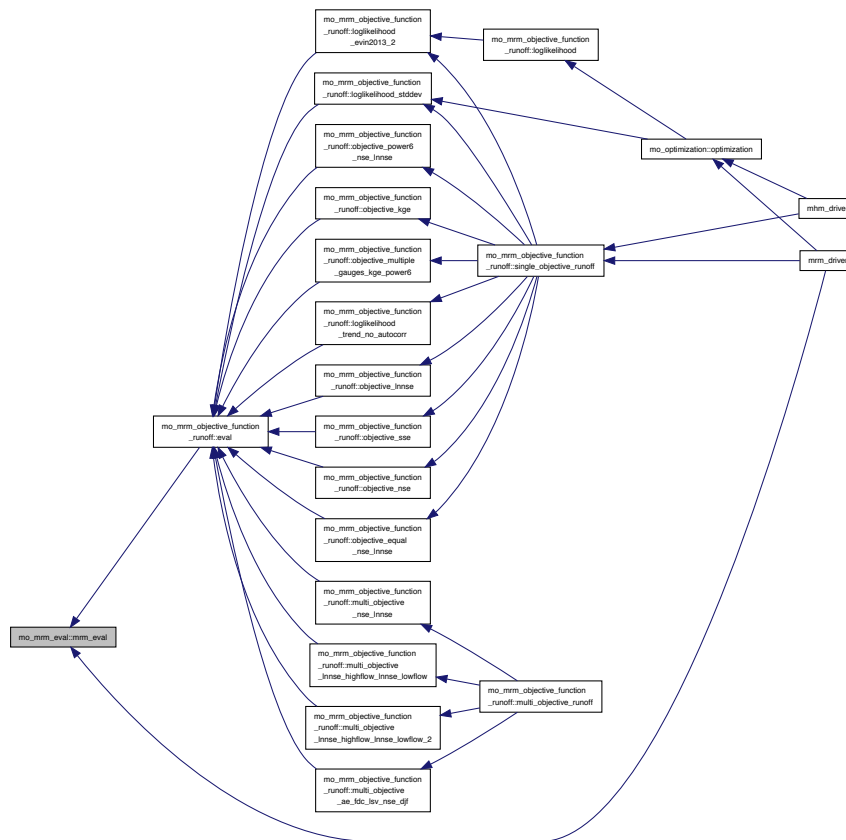
Parameters

in	<i>real(dp), dimension(:) :: parameterset</i>	a set of global parameter (gamma) to run mHM, DIMENSION [no. of global_Parameters]
out	<i>real(dp), dimension(:, :), optional :: runoff</i>	returns runoff time series, DIMENSION [nTimeSteps, nGaugesTotal]

Author

Stephan Thober

Here is the caller graph for this function:



15.30 mo_mrm_file Module Reference

Provides file names and units for mRM.

Variables

- character(len= *), parameter `version` = '1.1'
Current mHM model version.
- character(len= *), parameter `version_date` = 'Nov 2016'
Time of current mHM model version release.
- character(len= *), parameter `file_main` = 'mrm_driver.f90'
Driver file.
- character(len= *), parameter `file_namelist_mrm` = 'mhm.nml'
Namelist file name.
- integer, parameter `unamelist_mrm` = 40
Unit for namelist.
- character(len= *), parameter `file_namelist_param_mrm` = 'mhm_parameter.nml'
Parameter namelists file name.
- integer, parameter `unamelist_param` = 41
Unit for namelist.
- character(len= *), parameter `file_dem` = 'dem.asc'

- DEM input data file.*

 - integer, parameter `udem` = 53

Unit for DEM input data file.
- character(len= *), parameter `file_facc` = 'facc.asc'

flow accumulation input data file

 - integer, parameter `ufacc` = 56

Unit for flow accumulation input data file.
- character(len= *), parameter `file_fdir` = 'fdir.asc'

flow direction input data file

 - integer, parameter `ufdir` = 57

Unit for flow direction input data file.
- integer, parameter `ulcoverclass` = 61

Unit for LCover input data file.
- character(len= *), parameter `file_gaugeloc` = 'idgauges.asc'

gauge location input data file

 - integer, parameter `ugaugeloc` = 62

Unit for gauge location input data file.
- integer, parameter `udischarge` = 66

unit for discharge time series
- character(len= *), parameter `file_defoutput` = 'mrm_outputs.nml'

file defining mRM's outputs

 - integer, parameter `udefoutput` = 67

Unit for file defining mRM's outputs.
- character(len= *), parameter `file_config` = 'ConfigFile.log'

file defining mHM's outputs

 - integer, parameter `uconfig` = 68

Unit for file defining mHM's outputs.
- character(len= *), parameter `file_opti` = 'FinalParam.out'

file defining optimization outputs (objective and p arameter set)

 - integer, parameter `uopti` = 72

Unit for file optimization outputs (objective and p arameter set)
- character(len= *), parameter `file_opti_nml` = 'FinalParam.nml'

file defining optimization outputs in a namelist fo rmat (parameter set)

 - integer, parameter `uopti_nml` = 73

Unit for file optimization outputs in a namelist fo rmat (parameter set)
- character(len= *), parameter `file_daily_discharge` = 'daily_discharge.out'

file defining optimazation outputs

 - integer, parameter `udaily_discharge` = 74

Unit for file optimazation outputs.
- character(len= *), parameter `ncfile_discharge` = 'discharge.nc'

file defining optimazation outputs
- character(len= *), parameter `file_mrm_output` = 'mRM_Fluxes_States.nc'

file containing mrm output

15.30.1 Detailed Description

Provides file names and units for mRM.

Provides all filenames as well as all units used for the multiscale Routing Model mRM.

Author

Matthias Cuntz, Stephan Thober

Date

Aug 2015

15.30.2 Variable Documentation

15.30.2.1 file_config

```
character(len=*), parameter mo_mrm_file::file_config = 'ConfigFile.log'
```

file defining mHM's outputs

Referenced by mo_mrm_write::write_configfile().

15.30.2.2 file_daily_discharge

```
character(len=*), parameter mo_mrm_file::file_daily_discharge = 'daily_discharge.out'
```

file defining optimization outputs

Referenced by mo_mrm_write::write_daily_obs_sim_discharge().

15.30.2.3 file_defoutput

```
character(len=*), parameter mo_mrm_file::file_defoutput = 'mrm_outputs.nml'
```

file defining mRM's outputs

Referenced by mo_mrm_init::config_output(), mo_mrm_init::print_startup_message(), and mo_mrm_read_config↵
::read_mrm_config().

15.30.2.4 file_dem

```
character(len=*), parameter mo_mrm_file::file_dem = 'dem.asc'
```

DEM input data file.

15.30.2.5 file_facc

```
character(len=*), parameter mo_mrm_file::file_facc = 'facc.asc'
```

flow accumulation input data file

Referenced by mo_mrm_read_data::mrm_read_l0_data().

15.30.2.6 file_fdir

```
character(len=*), parameter mo_mrm_file::file_fdir = 'fdir.asc'
```

flow direction input data file

15.30.2.7 file_gaugeloc

```
character(len=*), parameter mo_mrm_file::file_gaugeloc = 'idgauges.asc'
```

gauge location input data file

15.30.2.8 file_main

```
character(len=*), parameter mo_mrm_file::file_main = 'mrm_driver.f90'
```

Driver file.

Referenced by mo_mrm_init::print_startup_message().

15.30.2.9 file_mrm_output

```
character(len=*), parameter mo_mrm_file::file_mrm_output = 'mRM_Fluxes_States.nc'
```

file containing mrm output

Referenced by mo_mrm_write_fluxes_states::createoutputfile().

15.30.2.10 file_namelist_mrm

```
character(len=*), parameter mo_mrm_file::file_namelist_mrm = 'mhm.nml'
```

Namelist file name.

Referenced by mo_mrm_init::config_output(), mo_mrm_init::print_startup_message(), mo_mrm_read_config↵
::read_mrm_config(), and mo_mrm_read_config::read_mrm_config_coupling().

15.30.2.11 file_namelist_param_mrm

```
character(len=*), parameter mo_mrm_file::file_namelist_param_mrm = 'mhm_parameter.nml'
```

Parameter namelists file name.

Referenced by mo_mrm_init::config_output(), mo_mrm_init::print_startup_message(), and mo_mrm_read_config↵
::read_mrm_config().

15.30.2.12 file_opti

```
character(len=*), parameter mo_mrm_file::file_opti = 'FinalParam.out'
```

file defining optimization outputs (objective and parameter set)

Referenced by mo_mrm_write::mrm_write_optifile().

15.30.2.13 file_opti_nml

```
character(len=*), parameter mo_mrm_file::file_opti_nml = 'FinalParam.nml'
```

file defining optimization outputs in a namelist for rmat (parameter set)

Referenced by mo_mrm_write::mrm_write_optinamelist().

15.30.2.14 ncfile_discharge

```
character(len=*), parameter mo_mrm_file::ncfile_discharge = 'discharge.nc'
```

file defining optimization outputs

Referenced by mo_mrm_write::write_daily_obs_sim_discharge().

15.30.2.15 uconfig

```
integer, parameter mo_mrm_file::uconfig = 68
```

Unit for file defining mHM's outputs.

Referenced by mo_mrm_write::write_configfile().

15.30.2.16 udaily_discharge

```
integer, parameter mo_mrm_file::udaily_discharge = 74
```

Unit for file optimization outputs.

Referenced by mo_mrm_write::write_daily_obs_sim_discharge().

15.30.2.17 ndefoutput

```
integer, parameter mo_mrm_file::ndefoutput = 67
```

Unit for file defining mRM's outputs.

Referenced by mo_mrm_read_config::read_mrm_config().

15.30.2.18 udem

```
integer, parameter mo_mrm_file::udem = 53
```

Unit for DEM input data file.

15.30.2.19 udischarge

```
integer, parameter mo_mrm_file::udischarge = 66
```

unit for discharge time series

Referenced by mo_mrm_read_data::mrm_read_discharge().

15.30.2.20 ufacc

```
integer, parameter mo_mrm_file::ufacc = 56
```

Unit for flow accumulation input data file.

Referenced by mo_mrm_read_data::mrm_read_l0_data().

15.30.2.21 ufdir

```
integer, parameter mo_mrm_file::ufdir = 57
```

Unit for flow direction input data file.

15.30.2.22 ugaugeloc

```
integer, parameter mo_mrm_file::ugaugeloc = 62
```

Unit for gauge location input data file.

15.30.2.23 ulcoverclass

```
integer, parameter mo_mrm_file::ulcoverclass = 61
```

Unit for LCover input data file.

15.30.2.24 unamelist_mrm

```
integer, parameter mo_mrm_file::unamelist_mrm = 40
```

Unit for namelist.

Referenced by mo_mrm_read_config::read_mrm_config(), and mo_mrm_read_config::read_mrm_config_↔ coupling().

15.30.2.25 unamelist_param

integer, parameter mo_mrm_file::unamelist_param = 41

Unit for namelist.

Referenced by mo_mrm_read_config::read_mrm_routing_params().

15.30.2.26 uopti

integer, parameter mo_mrm_file::uopti = 72

Unit for file optimization outputs (objective and p arameter set)

Referenced by mo_mrm_write::mrm_write_optifile().

15.30.2.27 uopti_nml

integer, parameter mo_mrm_file::uopti_nml = 73

Unit for file optimization outputs in a namelist fo rmat (parameter set)

Referenced by mo_mrm_write::mrm_write_optinamelist().

15.30.2.28 version

character(len=*), parameter mo_mrm_file::version = '1.1'

Current mHM model version.

Referenced by mo_mrm_write_fluxes_states::createoutputfile(), mo_mrm_init::print_startup_message(), and mo←
_mrm_write::write_configfile().

15.30.2.29 version_date

character(len=*), parameter mo_mrm_file::version_date = 'Nov 2016'

Time of current mHM model version release.

Referenced by mo_mrm_init::print_startup_message().

15.31 mo_mrm_global_variables Module Reference

Global variables for mRM only.

Data Types

- type [basininfo](#)
- type [gaugingstation](#)
- type [gridgeoref](#)

Variables

- integer(i4) [mrm_coupling_mode](#)
- logical [is_start](#)
- character(1024), public [project_details](#)
- character(1024), public [setup_description](#)
- character(1024), public [simulation_type](#)
- character(256), public [conventions](#)
- character(1024), public [contact](#)
- character(1024), public [mhm_details](#)
- character(1024), public [history](#)
- integer(i4), public [timestep](#)
- integer(i4), dimension(:), allocatable, public [timestep_model_inputs](#)
- integer(i4), public [iflag_cordinate_sys](#)
- integer(i4), dimension(:), allocatable, public [l0_basin](#)
- real(dp), dimension(:), allocatable, public [resolutionrouting](#)
- real(dp), dimension(:), allocatable, public [resolutionhydrology](#)
- logical, public [read_restart](#)
- logical, public [write_restart](#)
- logical, public [perform_mpr](#)
- character(256), public [dirconfigout](#)
- character(256), public [dircommonfiles](#)
- character(256), dimension(:), allocatable, public [dirmorpho](#)
- character(256), dimension(:), allocatable, public [dirlcover](#)
- character(256), dimension(:), allocatable, public [dirgauges](#)
- character(256), dimension(:), allocatable, public [dirtotalrunoff](#)
- character(256), dimension(:), allocatable, public [dirout](#)
- character(256), dimension(:), allocatable, public [dirrestartout](#)
- character(256), dimension(:), allocatable, public [dirrestartin](#)
- character(256), dimension(:), allocatable, public [filelatlon](#)
- integer(i4), public [ntstepday](#)
- type([gridgeoref](#)), public [level0](#)
- type([gridgeoref](#)), public [level1](#)
- type([gridgeoref](#)), public [level11](#)
- type([gridgeoref](#)), public [level110](#)
- real(dp), dimension(:), allocatable, public [l0_longitude](#)
- real(dp), dimension(:), allocatable, public [l0_latitude](#)
- real(dp), dimension(:), allocatable, public [l11_rect_longitude](#)
- real(dp), dimension(:), allocatable, public [l11_rect_latitude](#)
- real(dp), dimension(:,:), allocatable, public [mrm_runoff](#)
- integer(i4), public [ngaugestotal](#)
- integer(i4), public [ninflowgaugestotal](#)
- integer(i4), public [nmeasperday](#)
- type([gaugingstation](#)), public [gauge](#)
- type([gaugingstation](#)), public [inflowgauge](#)
- real(dp), public [fracsealed_cityarea](#)
- integer(i4), public [nlcoverscene](#)
- character(256), dimension(:), allocatable, public [lcfilename](#)
- integer(i4), dimension(:,:), allocatable, public [lcyearid](#)
- type([period](#)), dimension(:), allocatable, public [warmpers](#)
- type([period](#)), dimension(:), allocatable, public [evalpers](#)
- type([period](#)), dimension(:), allocatable, public [simpers](#)
- type([period](#)), dimension(:), allocatable, public [readpers](#)
- integer(i4), dimension(:), allocatable, public [warmingdays_mrm](#)
- integer(i4), public [nbasins](#)

- type([basininfo](#)), public [basin_mrm](#)
- logical, dimension(:), allocatable, target [l0_mask_mrm](#)
- real(dp), dimension(:), allocatable, target, public [l0_elev_read](#)
- real(dp), dimension(:), pointer, public [l0_elev_mrm](#)
- integer(i4), dimension(:), allocatable, public [l0_facc](#)
- integer(i4), dimension(:), allocatable, public [l0_fdir](#)
- integer(i4), dimension(:,:), pointer, public [l0_lcover_mrm](#)
- integer(i4), dimension(:,:), allocatable, target, public [l0_lcover_read](#)
- integer(i4), dimension(:,:), allocatable, public [l0_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [l0_id](#)
- integer(i4), dimension(:), allocatable, public [l0_gaugeloc](#)
- integer(i4), dimension(:), allocatable, public [l0_inflowgaugeloc](#)
- integer(i4), dimension(:), allocatable, public [l0_l1_id](#)
- real(dp), dimension(:), allocatable, public [l0_areacell](#)
- integer(i4), dimension(:), allocatable, public [l0_drasc](#)
- integer(i4), dimension(:), allocatable, public [l0_dracell](#)
- integer(i4), dimension(:), allocatable, public [l0_streamnet](#)
- integer(i4), dimension(:), allocatable, public [l0_floodplain](#)
- integer(i4) [l0_ncells](#)
- integer(i4), dimension(:), allocatable, public [l1_l1_id](#)
- real(dp), dimension(:), allocatable, public [l1_areacell](#)
- integer(i4), dimension(:), allocatable, public [l1_id](#)
- integer(i4) [l1_ncells](#)
- real(dp), dimension(:,:), allocatable, public [l1_total_runoff_in](#)
- integer(i4), public [l1_ncells](#)
- integer(i4), dimension(:,:), allocatable, public [l11_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [l1_l1_id](#)
- real(dp), dimension(:), allocatable, public [l11_areacell](#)
- integer(i4), dimension(:), allocatable, public [l11_id](#)
- integer(i4), dimension(:), allocatable, public [l11_fdir](#)
- integer(i4), dimension(:), allocatable, public [l11_noutlets](#)
- integer(i4), dimension(:), allocatable, public [l11_upbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l11_downbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l11_leftbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l11_rightbound_l0](#)
- integer(i4), dimension(:), allocatable, public [l11_upbound_l1](#)
- integer(i4), dimension(:), allocatable, public [l11_downbound_l1](#)
- integer(i4), dimension(:), allocatable, public [l11_leftbound_l1](#)
- integer(i4), dimension(:), allocatable, public [l11_rightbound_l1](#)
- integer(i4), dimension(:), allocatable, public [l11_rowout](#)
- integer(i4), dimension(:), allocatable, public [l11_colout](#)
- real(dp), dimension(:), allocatable, public [l11_qmod](#)
- real(dp), dimension(:), allocatable, public [l11_qout](#)
- real(dp), dimension(:,:), allocatable, public [l11_qtin](#)
- real(dp), dimension(:,:), allocatable, public [l11_qtr](#)
- real(dp), dimension(:), allocatable, public [l11_fracfpimp](#)
- integer(i4), dimension(:), allocatable, public [l11_fromn](#)
- integer(i4), dimension(:), allocatable, public [l11_ton](#)
- integer(i4), dimension(:), allocatable, public [l11_netperm](#)
- integer(i4), dimension(:), allocatable, public [l11_frow](#)
- integer(i4), dimension(:), allocatable, public [l11_fcol](#)
- integer(i4), dimension(:), allocatable, public [l11_trow](#)
- integer(i4), dimension(:), allocatable, public [l11_tcol](#)
- integer(i4), dimension(:), allocatable, public [l11_rorder](#)
- integer(i4), dimension(:), allocatable, public [l11_label](#)

- logical, dimension(:), allocatable, public [l11_sink](#)
- real(dp), dimension(:), allocatable, public [l11_length](#)
- real(dp), dimension(:), allocatable, public [l11_afloodplain](#)
- real(dp), dimension(:), allocatable, public [l11_slope](#)
- real(dp), dimension(:), allocatable, public [l11_k](#)
- real(dp), dimension(:), allocatable, public [l11_xi](#)
- real(dp), dimension(:), allocatable, public [l11_tsROUT](#)
- real(dp), dimension(:), allocatable, public [l11_c1](#)
- real(dp), dimension(:), allocatable, public [l11_c2](#)
- integer(i4) [timestep_model_outputs_mrm](#)
- logical, dimension(noutflxstate) [outputflxstate_mrm](#)

15.31.1 Detailed Description

Global variables for mRM only.

Authors

Luis Samaniego, Stephan Thober

Date

Aug 2015

15.31.2 Variable Documentation

15.31.2.1 basin_mrm

```
type(basininfo), public mo_mrm_global_variables::basin_mrm
```

Referenced by `mo_mrm_init::config_output()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_init::l0_check_↵
input_routing()`, `mo_mrm_net_startup::l11_flow_direction()`, `mo_mrm_net_startup::l11_link_location()`, `mo_mrm_↵
_net_startup::l11_set_drain_outlet_gauges()`, `mo_mrm_net_startup::l11_variable_init()`, `mo_mhm_eval::mhm_↵
_eval()`, `mo_mrm_init::mrm_init()`, `mo_mrm_init::mrm_init_param()`, `mo_mrm_read_data::mrm_l1_variable_↵
init()`, `mo_mrm_write::mrm_write()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_read_latlon::read_latlon()`,
`mo_write_ascii::write_configfile()`, `mo_mrm_write::write_configfile()`, and `mo_mrm_write::write_daily_obs_sim_↵
discharge()`.

15.31.2.2 contact

```
character(1024), public mo_mrm_global_variables::contact
```

15.31.2.3 conventions

```
character(256), public mo_mrm_global_variables::conventions
```

15.31.2.4 dircommonfiles

```
character(256), public mo_mrm_global_variables::dircommonfiles
```

15.31.2.5 dirconfigout

```
character(256), public mo_mrm_global_variables::dirconfigout
```

Referenced by `mrm_driver()`, `mo_mrm_write::mrm_write_optifile()`, `mo_mrm_write::mrm_write_optinamelist()`, and `mo_mrm_write::write_configfile()`.

15.31.2.6 dirgauges

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirgauges
```

Referenced by `mo_mrm_init::config_output()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.31.2.7 dirlcover

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirlcover
```

Referenced by `mo_mrm_init::config_output()`, and `mo_mrm_write::write_configfile()`.

15.31.2.8 dirmorpho

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirmorpho
```

Referenced by `mo_mrm_init::config_output()`, and `mo_mrm_write::write_configfile()`.

15.31.2.9 dirout

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirout
```

Referenced by `mo_mrm_write_fluxes_states::close()`, `mo_mrm_init::config_output()`, `mo_mrm_write_fluxes_states::createoutputfile()`, `mo_mrm_write::write_configfile()`, and `mo_mrm_write::write_daily_obs_sim_discharge()`.

15.31.2.10 dirrestartin

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirrestartin
```

Referenced by `mo_mhm_eval::mhm_eval()`, and `mo_mrm_init::mrm_init()`.

15.31.2.11 dirrestartout

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirrestartout
```

Referenced by mo_mrm_write::mrm_write(), and mo_mrm_write::write_configfile().

15.31.2.12 dirtotalrunoff

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::dirtotalrunoff
```

Referenced by mo_mrm_write::write_configfile().

15.31.2.13 evalper

```
type(period), dimension(:), allocatable, public mo_mrm_global_variables::evalper
```

Referenced by mo_mrm_write_fluxes_states::createoutputfile(), mo_mrm_objective_function_runoff::extract_runoff(), mo_mrm_write::mrm_write(), mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf(), mo_mrm_write::write_configfile(), and mo_mrm_write::write_daily_obs_sim_discharge().

15.31.2.14 filelatlon

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::filelatlon
```

Referenced by mo_mrm_read_latlon::read_latlon().

15.31.2.15 fracsealed_cityarea

```
real(dp), public mo_mrm_global_variables::fracsealed_cityarea
```

Referenced by mo_mhm_eval::mhm_eval().

15.31.2.16 gauge

```
type(gaugingstation), public mo_mrm_global_variables::gauge
```

Referenced by mo_mrm_objective_function_runoff::extract_runoff(), mo_mrm_write::mrm_write(), mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf(), mo_write_ascii::write_configfile(), mo_mrm_write::write_configfile(), and mo_mrm_write::write_daily_obs_sim_discharge().

15.31.2.17 history

```
character(1024), public mo_mrm_global_variables::history
```

15.31.2.18 iflag_cordinate_sys

```
integer(i4), public mo_mrm_global_variables::iflag_cordinate_sys
```

Referenced by mo_mrm_init::mrm_init_param(), mo_mrm_read_data::mrm_I0_variable_init(), and mo_mrm_init::mrm_update_param().

15.31.2.19 inflowgauge

```
type(gaugingstation), public mo_mrm_global_variables::inflowgauge
```

Referenced by mo_mhm_eval::mhm_eval(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.20 is_start

```
logical mo_mrm_global_variables::is_start
```

Referenced by mo_mrm_routing::mrm_routing().

15.31.2.21 l0_areacell

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l0_areacell
```

Referenced by mo_mrm_net_startup::l11_variable_init(), mo_mhm_eval::mhm_eval(), mo_mrm_read_data::mrm_l0_variable_init(), mo_mrm_read_data::mrm_l1_variable_init(), and mo_mrm_restart::mrm_write_restart().

15.31.2.22 l0_basin

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_basin
```

Referenced by mo_mrm_init::mrm_init(), mo_mrm_restart::mrm_read_restart_config(), and mo_mrm_read_latlon::read_latlon().

15.31.2.23 l0_cellcoor

```
integer(i4), dimension(:,:), allocatable, public mo_mrm_global_variables::l0_cellcoor
```

Referenced by mo_mrm_read_data::mrm_l0_variable_init(), and mo_mrm_restart::mrm_write_restart().

15.31.2.24 l0_dracell

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_dracell
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.25 l0_drasc

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_drasc
```

Referenced by mo_mrm_net_startup::l11_flow_direction(), and mo_mrm_restart::mrm_write_restart().

15.31.2.26 l0_elev_mrm

```
real(dp), dimension(:), pointer, public mo_mrm_global_variables::l0_elev_mrm
```

Referenced by mo_mrm_net_startup::l11_stream_features().

15.31.2.27 l0_elev_read

```
real(dp), dimension(:), allocatable, target, public mo_mrm_global_variables::l0_elev_read
```

15.31.2.28 l0_facc

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_facc
```

Referenced by mo_mrm_init::l0_check_input_routing(), and mo_mrm_net_startup::l11_flow_direction().

15.31.2.29 l0_fdir

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_fdir
```

Referenced by mo_mrm_init::l0_check_input_routing(), mo_mrm_net_startup::l11_flow_direction(), and mo_mrm↵_net_startup::l11_set_drain_outlet_gauges().

15.31.2.30 l0_floodplain

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_floodplain
```

Referenced by mo_mhm_eval::mhm_eval(), and mo_mrm_restart::mrm_write_restart().

15.31.2.31 l0_gaugeloc

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_gaugeloc
```

15.31.2.32 l0_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_id
```

Referenced by mo_mrm_read_data::mrm_l0_variable_init(), and mo_mrm_restart::mrm_write_restart().

15.31.2.33 l0_inflowgaugeloc

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_inflowgaugeloc
```

15.31.2.34 l0_l11_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_l11_id
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.35 l0_latitude

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l0_latitude
```

Referenced by mo_mrm_read_latlon::read_latlon().

15.31.2.36 l0_lcover_mrm

```
integer(i4), dimension(:, :), pointer, public mo_mrm_global_variables::l0_lcover_mrm
```

Referenced by mo_mhm_eval::mhm_eval().

15.31.2.37 l0_lcover_read

```
integer(i4), dimension(:, :), allocatable, target, public mo_mrm_global_variables::l0_lcover_read
```

15.31.2.38 l0_longitude

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l0_longitude
```

Referenced by mo_mrm_read_latlon::read_latlon().

15.31.2.39 l0_mask_mrm

```
logical, dimension(:), allocatable, target mo_mrm_global_variables::l0_mask_mrm
```

15.31.2.40 l0_ncells

```
integer(i4) mo_mrm_global_variables::l0_ncells
```

Referenced by mo_mrm_read_data::mrm_l0_variable_init(), and mo_mrm_write::write_configfile().

15.31.2.41 l0_streamnet

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l0_streamnet
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.42 l11_afloodplain

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_afloodplain
```

Referenced by mo_mhm_eval::mhm_eval(), and mo_mrm_restart::mrm_write_restart().

15.31.2.43 l11_areacell

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_areacell
```

Referenced by mo_mhm_eval::mhm_eval(), and mo_mrm_restart::mrm_write_restart().

15.31.2.44 l11_c1

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_c1
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_read_restart_states(), mo_mrm_init::mrm_↵
update_param(), mo_mrm_restart::mrm_write_restart(), mo_mrm_init::variables_alloc_routing(), and mo_mrm_↵
init::variables_default_init_routing().

15.31.2.45 l11_c2

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_c2
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_read_restart_states(), mo_mrm_init::mrm_↵
update_param(), mo_mrm_restart::mrm_write_restart(), mo_mrm_init::variables_alloc_routing(), and mo_mrm_↵
init::variables_default_init_routing().

15.31.2.46 l11_cellcoor

```
integer(i4), dimension(:,:), allocatable, public mo_mrm_global_variables::l11_cellcoor
```

Referenced by mo_mrm_net_startup::l11_set_network_topology(), and mo_mrm_restart::mrm_write_restart().

15.31.2.47 l11_colout

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_colout
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.48 l11_downbound_l0

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_downbound_l0
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.49 l11_downbound_l1

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_downbound_l1
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.50 l11_fcol

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_fcol
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.51 l11_fdir

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_fdir
```

Referenced by mo_mrm_net_startup::l11_set_network_topology(), and mo_mrm_restart::mrm_write_restart().

15.31.2.52 l11_fracfpimp

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_fracfpimp
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_read_restart_states(), mo_mrm_restart::mrm_write_restart(), mo_mrm_init::variables_alloc_routing(), and mo_mrm_init::variables_default_init_routing().

15.31.2.53 l11_fromn

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_fromn
```

Referenced by mo_mrm_net_startup::l11_routing_order(), mo_mrm_net_startup::l11_set_network_topology(), mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.54 l11_frow

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_frow
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.55 l11_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_id
```

Referenced by mo_mrm_net_startup::l11_set_network_topology(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.56 l11_k

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_k
```

Referenced by mo_mrm_restart::mrm_read_restart_states(), mo_mrm_restart::mrm_write_restart(), mo_mrm_init::variables_alloc_routing(), and mo_mrm_init::variables_default_init_routing().

15.31.2.57 l11_l1_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_l1_id
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_write_restart(), and mo_mrm_write::write_configfile().

15.31.2.58 l11_label

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_label
```

Referenced by mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.59 l11_leftbound_l0

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_leftbound_l0
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.60 l11_leftbound_l1

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_leftbound_l1
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.61 l11_length

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_length
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.62 l11_ncells

```
integer(i4), public mo_mrm_global_variables::l11_ncells
```

Referenced by mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.63 l11_netperm

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_netperm
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_write_restart()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.31.2.64 l11_noutlets

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_noutlets
```

Referenced by `mo_mhm_eval::mhm_eval()`.

15.31.2.65 l11_qmod

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_qmod
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_read_restart_states()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_init::variables_alloc_routing()`, and `mo_mrm_init::variables_default_init_routing()`.

15.31.2.66 l11_qout

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_qout
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_read_restart_states()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_init::variables_alloc_routing()`, and `mo_mrm_init::variables_default_init_routing()`.

15.31.2.67 l11_qtin

```
real(dp), dimension(:, :), allocatable, public mo_mrm_global_variables::l11_qtin
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_read_restart_states()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_init::variables_alloc_routing()`, and `mo_mrm_init::variables_default_init_routing()`.

15.31.2.68 l11_qtr

```
real(dp), dimension(:, :), allocatable, public mo_mrm_global_variables::l11_qtr
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_read_restart_states()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_init::variables_alloc_routing()`, and `mo_mrm_init::variables_default_init_routing()`.

15.31.2.69 l11_rect_latitude

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_rect_latitude
```

Referenced by `mo_mrm_write_fluxes_states::geocoordinates()`, and `mo_mrm_read_latlon::read_latlon()`.

15.31.2.70 l11_rect_longitude

`real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_rect_longitude`

Referenced by `mo_mrm_write_fluxes_states::geocoordinates()`, and `mo_mrm_read_latlon::read_latlon()`.

15.31.2.71 l11_rightbound_l0

`integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_rightbound_l0`

Referenced by `mo_mrm_restart::mrm_write_restart()`.

15.31.2.72 l11_rightbound_l1

`integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_rightbound_l1`

Referenced by `mo_mrm_restart::mrm_write_restart()`.

15.31.2.73 l11_rorder

`integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_rorder`

Referenced by `mo_mrm_restart::mrm_write_restart()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.31.2.74 l11_rowout

`integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_rowout`

Referenced by `mo_mrm_restart::mrm_write_restart()`.

15.31.2.75 l11_sink

`logical, dimension(:), allocatable, public mo_mrm_global_variables::l11_sink`

Referenced by `mo_mrm_restart::mrm_write_restart()`.

15.31.2.76 l11_slope

`real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_slope`

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_restart::mrm_write_restart()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.31.2.77 l11_tcol

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_tcol
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.78 l11_ton

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_ton
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.79 l11_trow

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_trow
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.80 l11_tsROUT

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_tsROUT
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_init::mrm_init_param(), mo_mrm_init::mrm_update_param(), and mo_mrm_restart::mrm_write_restart().

15.31.2.81 l11_upbound_l0

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_upbound_l0
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.82 l11_upbound_l1

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l11_upbound_l1
```

Referenced by mo_mrm_restart::mrm_write_restart().

15.31.2.83 l11_xi

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l11_xi
```

Referenced by mo_mrm_restart::mrm_read_restart_states(), mo_mrm_restart::mrm_write_restart(), mo_mrm_init::variables_alloc_routing(), and mo_mrm_init::variables_default_init_routing().

15.31.2.84 l1_areacell

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::l1_areacell
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_read_data::mrm_l1_variable_init(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.85 l1_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l1_id
```

Referenced by mo_mrm_net_startup::l1_variable_init(), and mo_mrm_restart::mrm_write_restart().

15.31.2.86 l1_l11_id

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::l1_l11_id
```

Referenced by mo_mrm_net_startup::l11_variable_init(), mo_mhm_eval::mhm_eval(), mo_mrm_restart::mrm_write_restart(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.87 l1_ncells

```
integer(i4) mo_mrm_global_variables::l1_ncells
```

Referenced by mo_mrm_read_data::mrm_l1_variable_init(), and mo_mrm_write::write_configfile().

15.31.2.88 l1_total_runoff_in

```
real(dp), dimension(:, :), allocatable, public mo_mrm_global_variables::l1_total_runoff_in
```

15.31.2.89 lcfilename

```
character(256), dimension(:), allocatable, public mo_mrm_global_variables::lcfilename
```

Referenced by mo_mrm_write::write_configfile().

15.31.2.90 lcyetid

```
integer(i4), dimension(:, :), allocatable, public mo_mrm_global_variables::lcyetid
```

Referenced by mo_mrm_write::write_configfile().

15.31.2.91 level0

```
type(gridgeoref), public mo_mrm_global_variables::level0
```

Referenced by `mo_mrm_net_startup::celldlength()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_net_startup::l11_flow_direction()`, `mo_mrm_read_data::mrm_l0_variable_init()`, `mo_mrm_read_data::mrm_l1_variable_init()`, `mo_mrm_read_data::mrm_read_l0_data()`, and `mo_mrm_read_latlon::read_latlon()`.

15.31.2.92 level1

```
type(gridgeoref), public mo_mrm_global_variables::level1
```

Referenced by `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_net_startup::l11_variable_init()`, and `mo_mrm_read_data::mrm_l1_variable_init()`.

15.31.2.93 level11

```
type(gridgeoref), public mo_mrm_global_variables::level11
```

Referenced by `mo_mrm_write_fluxes_states::createoutputfile()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_net_startup::l11_flow_direction()`, `mo_mrm_net_startup::l11_variable_init()`, and `mo_mrm_read_latlon::read_latlon()`.

15.31.2.94 level110

```
type(gridgeoref), public mo_mrm_global_variables::level110
```

15.31.2.95 mhm_details

```
character(1024), public mo_mrm_global_variables::mhm_details
```

15.31.2.96 mrm_coupling_mode

```
integer(i4) mo_mrm_global_variables::mrm_coupling_mode
```

Referenced by `mo_mrm_init::mrm_init()`, `mo_mrm_read_data::mrm_read_l0_data()`, `mo_mrm_write::mrm_write()`, `mo_mrm_read_config::read_mrm_config()`, `mo_mrm_read_config::read_mrm_config_coupling()`, and `mo_mrm_write::write_configfile()`.

15.31.2.97 mrm_runoff

```
real(dp), dimension(:, :), allocatable, public mo_mrm_global_variables::mrm_runoff
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_read_data::mrm_read_discharge()`, and `mo_mrm_write::mrm_write()`.

15.31.2.98 nbasins

```
integer(i4), public mo_mrm_global_variables::nbasins
```

Referenced by mo_mrm_init::config_output(), mo_mrm_net_startup::l11_flow_direction(), mo_mrm_net_startup::l11_variable_init(), mo_mhm_eval::mhm_eval(), mo_mrm_eval::mrm_eval(), mo_mrm_init::mrm_init(), mo_mrm_init::mrm_init_param(), mo_mrm_read_data::mrm_l1_variable_init(), mo_mrm_read_data::mrm_read_discharge(), mo_mrm_read_data::mrm_read_l0_data(), mo_mrm_write::mrm_write(), mo_mrm_write::write_configfile(), and mo_mrm_write::write_daily_obs_sim_discharge().

15.31.2.99 ngaugestotal

```
integer(i4), public mo_mrm_global_variables::ngaugestotal
```

Referenced by mo_mrm_write::mrm_write(), mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.100 ninflowgaugestotal

```
integer(i4), public mo_mrm_global_variables::ninflowgaugestotal
```

Referenced by mo_write_ascii::write_configfile(), and mo_mrm_write::write_configfile().

15.31.2.101 nlcoverscene

```
integer(i4), public mo_mrm_global_variables::nlcoverscene
```

Referenced by mo_mrm_read_data::mrm_read_l0_data().

15.31.2.102 nmeasperday

```
integer(i4), public mo_mrm_global_variables::nmeasperday
```

Referenced by mo_mrm_objective_function_runoff::extract_runoff(), and mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf().

15.31.2.103 ntstepday

```
integer(i4), public mo_mrm_global_variables::ntstepday
```

Referenced by mo_mrm_objective_function_runoff::extract_runoff(), mo_mrm_write_fluxes_states::fluxesunit(), mo_mrm_write::mrm_write(), and mo_mrm_read_config::read_mrm_config().

15.31.2.104 outputflxstate_mrm

```
logical, dimension(noutflxstate) mo_mrm_global_variables::outputflxstate_mrm
```

Referenced by mo_mhm_eval::mhm_eval(), mo_mrm_write_fluxes_states::newoutputdataset(), and mo_mrm_write_fluxes_states::updatedataset().

15.31.2.105 perform_mpr

```
logical, public mo_mrm_global_variables::perform_mpr
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_init::mrm_init()`, and `mo_mrm_read_data::mrm_read_l0_data()`.

15.31.2.106 project_details

```
character(1024), public mo_mrm_global_variables::project_details
```

15.31.2.107 read_restart

```
logical, public mo_mrm_global_variables::read_restart
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_eval::mrm_eval()`, `mo_mrm_init::mrm_init()`, and `mo_mrm_write::write_configfile()`.

15.31.2.108 readper

```
type(period), dimension(:), allocatable, public mo_mrm_global_variables::readper
```

15.31.2.109 resolutionhydrology

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::resolutionhydrology
```

Referenced by `mo_mhm_eval::mhm_eval()`, `mo_mrm_read_data::mrm_l1_variable_init()`, and `mo_mrm_write::write_configfile()`.

15.31.2.110 resolutionrouting

```
real(dp), dimension(:), allocatable, public mo_mrm_global_variables::resolutionrouting
```

Referenced by `mo_mrm_net_startup::l1_variable_init()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_init::mrm_init_param()`, `mo_mrm_init::mrm_update_param()`, `mo_write_ascii::write_configfile()`, and `mo_mrm_write::write_configfile()`.

15.31.2.111 setup_description

```
character(1024), public mo_mrm_global_variables::setup_description
```

15.31.2.112 simper

```
type(period), dimension(:), allocatable, public mo_mrm_global_variables::simper
```

Referenced by mo_mrm_write_fluxes_states::fluxesunit(), mo_mhm_eval::mhm_eval(), mo_mrm_read_data::mrm_read_total_runoff(), mo_mrm_write::mrm_write(), and mo_mrm_write::write_configfile().

15.31.2.113 simulation_type

```
character(1024), public mo_mrm_global_variables::simulation_type
```

15.31.2.114 timestep

```
integer(i4), public mo_mrm_global_variables::timestep
```

Referenced by mo_mrm_write_fluxes_states::fluxesunit(), mo_mhm_eval::mhm_eval(), mo_mrm_init::mrm_init_param(), mo_mrm_read_data::mrm_read_total_runoff(), and mo_mrm_write::write_configfile().

15.31.2.115 timestep_model_inputs

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::timestep_model_inputs
```

Referenced by mo_mhm_eval::mhm_eval().

15.31.2.116 timestep_model_outputs_mrm

```
integer(i4) mo_mrm_global_variables::timestep_model_outputs_mrm
```

Referenced by mo_mrm_write_fluxes_states::fluxesunit(), and mo_mhm_eval::mhm_eval().

15.31.2.117 warmingdays_mrm

```
integer(i4), dimension(:), allocatable, public mo_mrm_global_variables::warmingdays_mrm
```

Referenced by mo_mrm_objective_function_runoff::extract_runoff(), mo_mhm_eval::mhm_eval(), and mo_mrm_write::mrm_write().

15.31.2.118 warmper

```
type(period), dimension(:), allocatable, public mo_mrm_global_variables::warmper
```

Referenced by mo_mrm_write::write_configfile().

15.31.2.119 write_restart

```
logical, public mo_mrm_global_variables::write_restart
```

Referenced by mo_mrm_write::mrm_write(), and mo_mrm_write::write_configfile().

15.32 mo_mrm_init Module Reference

Wrapper for initializing Routing.

Functions/Subroutines

- subroutine, public [mrm_init](#) (L0_mask, L0_elev, L0_LCover)
Initialize all mRM variables at all levels (i.e., L0, L1, and L11).
- subroutine [print_startup_message](#) ()
- subroutine [config_output](#) ()
- subroutine, public [variables_default_init_routing](#) ()
Default initialization mRM related L11 variables.
- subroutine [l0_check_input_routing](#) (iBasin)
- subroutine [variables_alloc_routing](#) (iBasin)
- subroutine [mrm_init_param](#) (iBasin, param)
- subroutine, public [mrm_update_param](#) (iBasin, param)

15.32.1 Detailed Description

Wrapper for initializing Routing.

Calling all routines to initialize all mRM variables

Authors

Luis Samaniego, Rohini Kumar and Stephan Thober

Date

Aug 2015

15.32.2 Function/Subroutine Documentation

15.32.2.1 config_output()

```
subroutine mo_mrm_init::config_output ( )
```

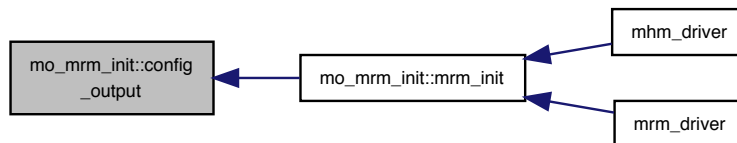
References [mo_mrm_global_variables::basin_mrm](#), [mo_mrm_global_variables::dirgauges](#), [mo_mrm_global_variables::dirlcover](#), [mo_mrm_global_variables::dirmorpho](#), [mo_mrm_global_variables::dirout](#), [mo_mrm_file::file_defoutput](#), [mo_mrm_file::file_namelist_mrm](#), [mo_mrm_file::file_namelist_param_mrm](#), [mo_kind::i4](#), [mo_message::message\(\)](#), and [mo_mrm_global_variables::nbasins](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.32.2.2 l0_check_input_routing()

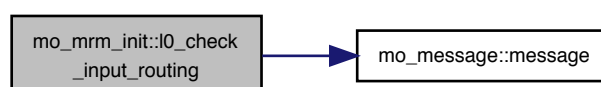
```

subroutine mo_mrm_init::l0_check_input_routing (
    integer(i4) iBasin )
  
```

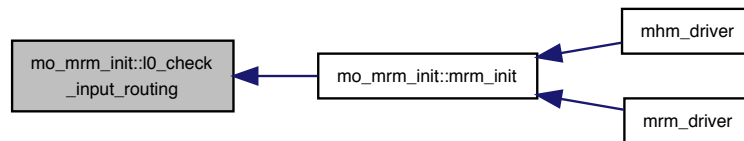
References `mo_mrm_global_variables::basin_mrm`, `mo_kind::i4`, `mo_mrm_global_variables::l0_facc`, `mo_mrm_global_variables::l0_fdir`, `mo_message::message()`, `mo_message::message_text`, and `mo_mrm_constants::nodata_i4`.

Referenced by `mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.32.2.3 mrm_init()

```

subroutine, public mo_mrm_init::mrm_init (
    logical, dimension(:), intent(in), optional, target L0_mask,
    real(dp), dimension(:), intent(in), optional, target L0_elev,
    integer(i4), dimension(:,,:), intent(in), optional, target L0_LCover )
  
```

Initialize all mRM variables at all levels (i.e., L0, L1, and L11).

Initialize all mRM variables at all levels (i.e., L0, L1, and L11) either with default values or with values from restart file. The L0 mask (L0_mask), L0 elevation (L0_elev), and L0 land cover (L0_LCover) can be provided as optional variables to save memory because these variable will then not be read in again.

Parameters

in	<i>logical, dimension(:), target, optional :: L0_mask - L0 mask</i>	
in	<i>real(dp), dimension(:), target, optional :: L0_elev - L0 elevation</i>	
in	<i>integer(i4), dimension(:,,:), target, optional :: L0_LCover - L0 land cover</i>	None

Author

Stephan Thober

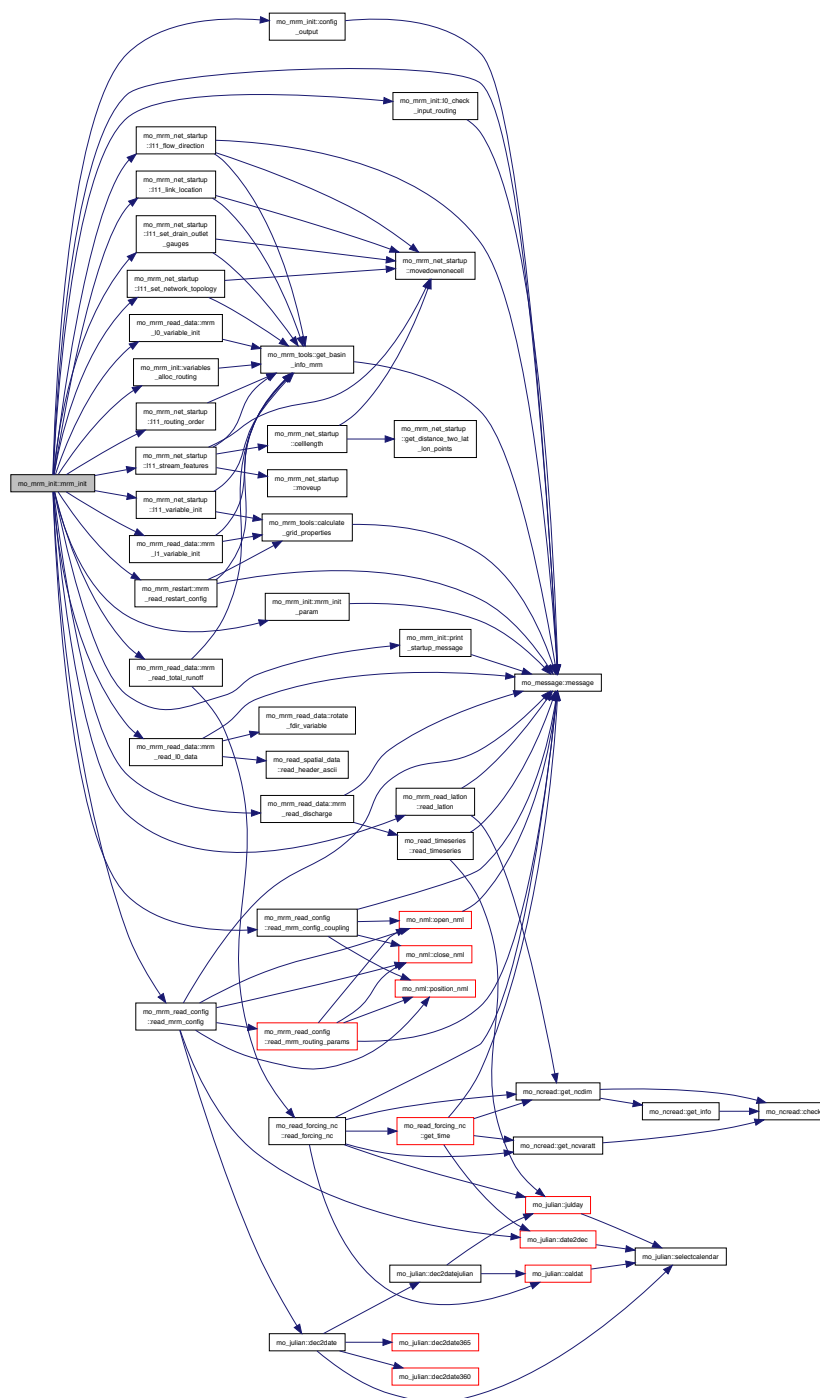
Date

Aug 2015

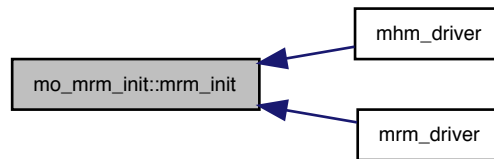
References mo_mrm_global_variables::basin_mrm, config_output(), mo_mrm_global_variables::dirrestartin, mo_↵
_kind::dp, mo_common_variables::global_parameters, mo_kind::i4, mo_mrm_global_variables::l0_basin, l0_↵
check_input_routing(), mo_mrm_net_startup::l11_flow_direction(), mo_mrm_net_startup::l11_link_location(), mo_↵
_mrm_net_startup::l11_routing_order(), mo_mrm_net_startup::l11_set_drain_outlet_gauges(), mo_mrm_net_↵
startup::l11_set_network_topology(), mo_mrm_net_startup::l11_stream_features(), mo_mrm_net_startup::l11_↵
_variable_init(), mo_message::message(), mo_mrm_global_variables::mrm_coupling_mode, mrm_init_param(),
mo_mrm_read_data::mrm_l0_variable_init(), mo_mrm_read_data::mrm_l1_variable_init(), mo_mrm_read_data_↵
::mrm_read_discharge(), mo_mrm_read_data::mrm_read_l0_data(), mo_mrm_restart::mrm_read_restart_config(),
mo_mrm_read_data::mrm_read_total_runoff(), mo_mrm_global_variables::nbasins, mo_mrm_constants::nodata_↵
_i4, mo_mrm_global_variables::perform_mpr, print_startup_message(), mo_common_variables::processmatrix,
mo_mrm_read_latlon::read_latlon(), mo_mrm_read_config::read_mrm_config(), mo_mrm_read_config::read_↵
mrm_config_coupling(), mo_mrm_global_variables::read_restart, and variables_alloc_routing().

Referenced by mhm_driver(), and mrm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.32.2.4 mrm_init_param()

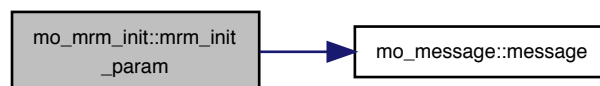
```

subroutine mo_mrm_init::mrm_init_param (
    integer(i4), intent(in) iBasin,
    real(dp), dimension(:), intent(in) param )
  
```

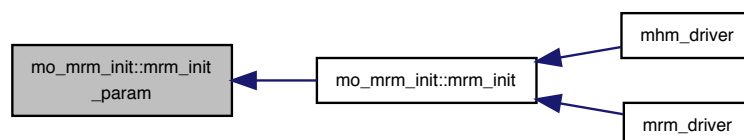
References `mo_mrm_global_variables::basin_mrm`, `mo_kind::dp`, `mo_mrm_constants::given_ts`, `mo_mrm_constants::hoursecs`, `mo_kind::i4`, `mo_mrm_global_variables::iflag_coordinate_sys`, `mo_mrm_global_variables::l11_tsout`, `mo_message::message()`, `mo_mrm_global_variables::nbasins`, `mo_common_variables::processmatrix`, `mo_mrm_global_variables::resolutionrouting`, and `mo_mrm_global_variables::timestep`.

Referenced by `mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.32.2.5 mrm_update_param()

```

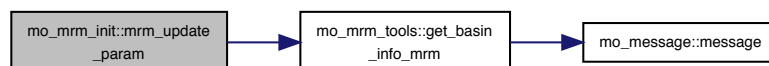
subroutine, public mo_mrm_init::mrm_update_param (
    integer(i4), intent(in) iBasin,
    real(dp), dimension(1), intent(in) param )

```

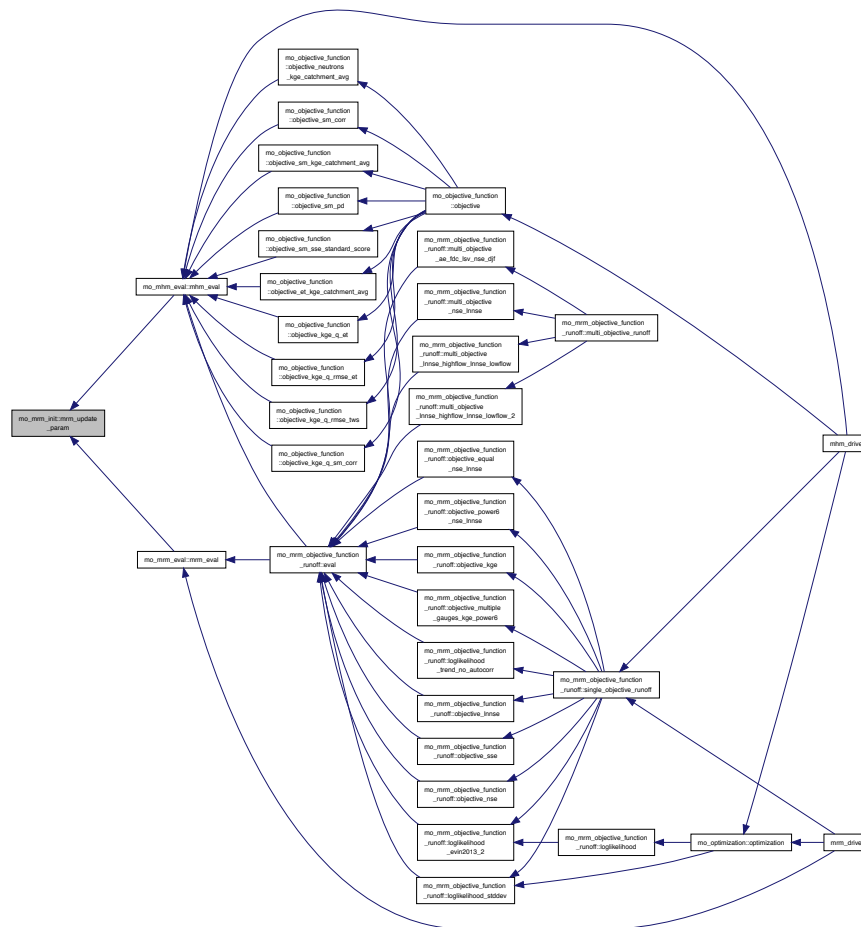
References mo_kind::dp, mo_mrm_tools::get_basin_info_mrm(), mo_mrm_constants::given_ts, mo_kind::i4, mo_mrm_global_variables::iflag_cordinate_sys, mo_mrm_global_variables::l11_c1, mo_mrm_global_variables::l11_c2, mo_mrm_global_variables::l11_tsrou, mo_mrm_global_variables::resolutionrouting, and mo_mrm_constants::rout_space_weight.

Referenced by mo_mhm_eval::mhm_eval(), and mo_mrm_eval::mrm_eval().

Here is the call graph for this function:



Here is the caller graph for this function:



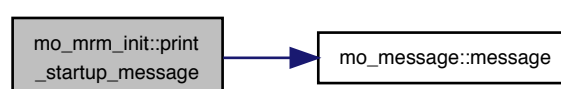
15.32.2.6 print_startup_message()

```
subroutine mo_mrm_init::print_startup_message ( )
```

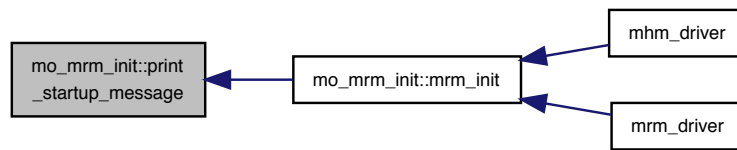
References mo_mrm_file::file_defoutput, mo_mrm_file::file_main, mo_mrm_file::file_namelist_mrm, mo_mrm_file::file_namelist_param_mrm, mo_kind::i4, mo_message::message(), mo_message::message_text, mo_string::utils::separator, mo_mrm_file::version, and mo_mrm_file::version_date.

Referenced by mrm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



15.32.2.7 variables_alloc_routing()

```

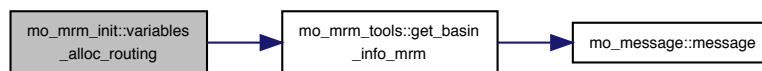
subroutine mo_mrm_init::variables_alloc_routing (
    integer(i4), intent(in) iBasin )

```

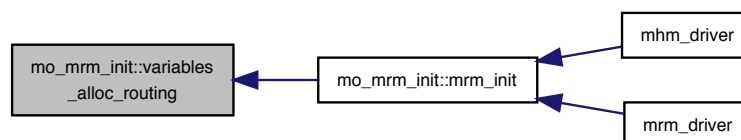
References `mo_kind::dp`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_kind::i4`, `mo_mrm_global_variables::l11_c1`, `mo_mrm_global_variables::l11_c2`, `mo_mrm_global_variables::l11_fracpimp`, `mo_mrm_global_variables::l11_k`, `mo_mrm_global_variables::l11_qmod`, `mo_mrm_global_variables::l11_qout`, `mo_mrm_global_variables::l11_qtin`, `mo_mrm_global_variables::l11_qtr`, `mo_mrm_global_variables::l11_xi`, and `mo_mrm_constants::nroutingstates`.

Referenced by `mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



Functions/Subroutines

- subroutine, public [reg_rout](#) (param, length, slope, fFPimp, TS, C1, C2)
Regionalized routing.

15.33.1 Detailed Description

Perform Multiscale Parameter Regionalization on Routing Parameters.

This module contains the subroutine for calculating the regionalized routing parameters (beta-parameters) given the five global routing parameters (gamma) at the level 0 scale.

Author

Luis Samaniego, Stephan Thober

Date

Aug 2015

15.33.2 Function/Subroutine Documentation

15.33.2.1 reg_rout()

```
subroutine, public mo_mrm_mpr::reg_rout (
    real(dp), dimension(5), intent(in) param,
    real(dp), dimension(:), intent(in) length,
    real(dp), dimension(:), intent(in) slope,
    real(dp), dimension(:), intent(in) fFPimp,
    real(dp), intent(in) TS,
    real(dp), dimension(:), intent(out) C1,
    real(dp), dimension(:), intent(out) C2 )
```

Regionalized routing.

sets up the Regionalized Routing parameters

Global parameters needed (see mhm_parameter.nml):

- param(1) = muskingumTravelTime_constant
- param(2) = muskingumTravelTime_riverLength
- param(3) = muskingumTravelTime_riverSlope
- param(4) = muskingumTravelTime_impervious
- param(5) = muskingumAttenuation_riverSlope

Parameters

in	real(dp) :: param(5)	- five input parameters
----	----------------------	-------------------------

Parameters

in	<i>real(dp) :: length(:)</i>	- [m] total length
in	<i>real(dp) :: slope(:)</i>	- average slope
in	<i>real(dp) :: fFPimp(:)</i>	- fraction of the flood plain with impervious layer
in	<i>real(dp) :: TS</i>	- [h] time step in
out	<i>real(dp) :: C1(:)</i>	- routing parameter C1 (Chow, 25-41)
out	<i>real(dp) :: C2(:)</i>	- routing parameter C2 (")

Author

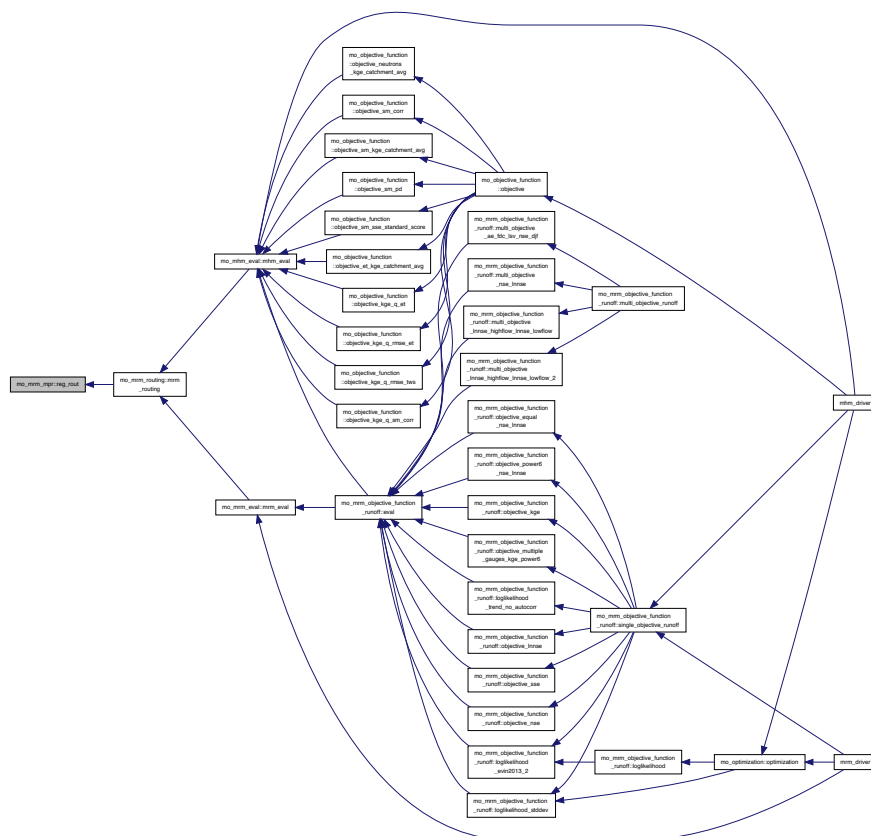
Stephan Thober, Rohini Kumar

Date

Dec 2012

Referenced by `mo_mrm_routing::mrm_routing()`.

Here is the caller graph for this function:



15.34 mo_mrm_net_startup Module Reference

Startup drainage network for mHM.

Functions/Subroutines

- subroutine, public [l11_variable_init](#) (iBasin)
Cell numbering at ROUTING LEVEL-11.
- subroutine, public [l11_flow_direction](#) (iBasin)
Determine the flow direction of the upscaled river network at level L11.
- subroutine, public [l11_set_network_topology](#) (iBasin)
Set network topology.
- subroutine, public [l11_routing_order](#) (iBasin)
Find routing order, headwater cells and sink.
- subroutine, public [l11_link_location](#) (iBasin)
Estimate the LO (row,col) location for each routing link at level L11.
- subroutine, public [l11_set_drain_outlet_gauges](#) (iBasin)
Draining cell identification and Set gauging node.
- subroutine, public [l11_stream_features](#) (iBasin)
Stream features (stream network and floodplain)
- subroutine, public [l11_fraction_sealed_floodplain](#) (nLinks, LCover0, floodPlain0, areaCell0, nLinkAFloodPlain, LCClassImp, nLinkFracFPimp)
If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module [mo_set_netcdf_restart](#).
- subroutine [moveup](#) (elev0, fDir0, fi, fj, ss, nn)
- subroutine [movedownonecell](#) (fDir, iRow, jCol)
- subroutine [celllength](#) (iBasin, fDir, iRow, jCol, iCoorSystem, length)
- subroutine, public [get_distance_two_lat_lon_points](#) (lat1, long1, lat2, long2, distance_out)
estimate distance in [m] between two points in a lat-lon

15.34.1 Detailed Description

Startup drainage network for mHM.

This module initializes the drainage network at L11 in mHM.

- Delineation of drainage network at level 11.
- Setting network topology (i.e. nodes and link).
- Determining routing order.
- Determining cell locations for network links.
- Find drainage outlet.
- Determine stream (links) features.

Authors

Luis Samaniego

Date

Dec 2012

15.34.2 Function/Subroutine Documentation

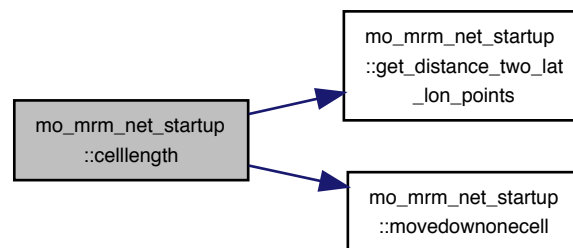
15.34.2.1 celllength()

```
subroutine mo_mrm_net_startup::celllength (
    integer(i4), intent(in) iBasin,
    integer(i4), intent(in) fDir,
    integer(i4), intent(in) iRow,
    integer(i4), intent(in) jCol,
    integer(i4), intent(in) iCoorSystem,
    real(dp), intent(out) length )
```

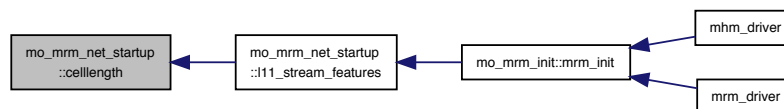
References `get_distance_two_lat_lon_points()`, `mo_mrm_global_variables::level0`, `movedownonecell()`, and `mo_constants::sqrt2_dp`.

Referenced by `l11_stream_features()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.2 get_distance_two_lat_lon_points()

```
subroutine, public mo_mrm_net_startup::get_distance_two_lat_lon_points (
    real(dp), intent(in) lat1,
    real(dp), intent(in) long1,
    real(dp), intent(in) lat2,
```

```

real(dp), intent(in) long2,
real(dp), intent(out) distance_out )

```

estimate distance in [m] between two points in a lat-lon

estimate distance in [m] between two points in a lat-lon

Parameters

in	<i>real(dp) :: lat1</i>	latitude of point-1
in	<i>real(dp) :: long1</i>	longitude of point-1
in	<i>real(dp) :: lat2</i>	latitude of point-2
in	<i>real(dp) :: long2</i>	longitude of point-2
out	<i>real(dp) :: distance_out</i>	distance between two points [m]

Author

Rohini Kumar

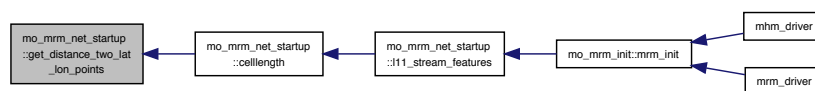
Date

May 2014

References mo_constants::radiusearth_dp, and mo_constants::twopi_dp.

Referenced by celllength().

Here is the caller graph for this function:



15.34.2.3 l11_flow_direction()

```

subroutine, public mo_mrm_net_startup::l11_flow_direction (
    integer(i4), intent(in) iBasin )

```

Determine the flow direction of the upscaled river network at level L11.

The hydrographs generated at each cell are routed through the drainage network at level-11 towards their outlets. The drainage network at level-11 is conceptualized as a graph whose nodes are hypothetically located at the center of each grid cell connected by links that represent the river reaches. The flow direction of a link correspond to the direction towards a neighboring cell in which the net flow accumulation (outflows minus inflows) attains its maximum value. The net flow accumulation across a cell's boundary at level-11 is estimated based on flow direction and flow accumulation obtained at level-0 ([Routing Network](#)). Note: level-1 denotes the modeling level, whereas level-L11 is at least as coarse as level-1. Experience has shown that routing can be done at a coarser resolution as level-1, hence the level-11 was introduced.

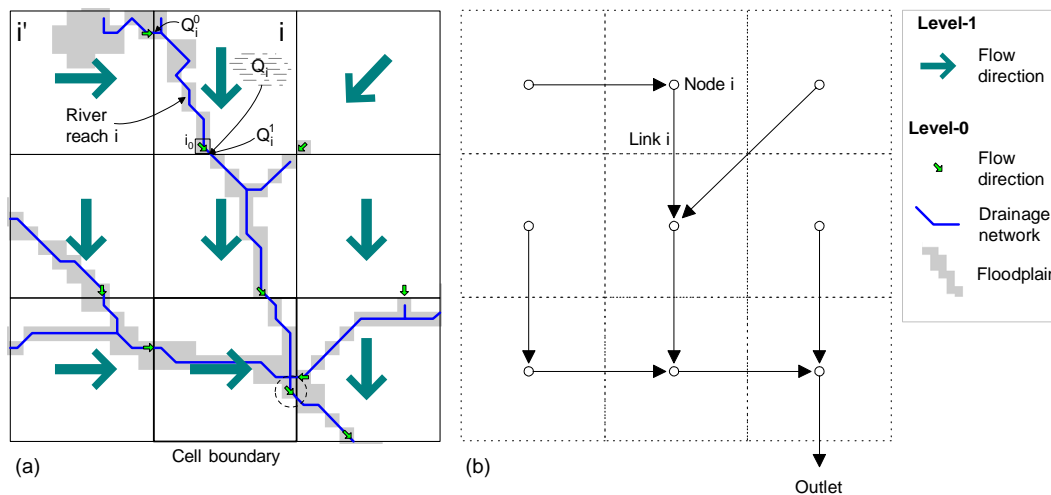


Figure 15.1: Upscaling routing network from L0 to L1 (or L11)

left panel depicts a schematic derivation of a drainage network at the level-11 based on level-0 flow direction and flow accumulation. The dotted line circle denotes the point with the highest flow accumulation within a grid cell. The topology of a typical drainage routing network at level-11 is shown in the right panel. Gray color areas denote the flood plains estimated in `mo_init_mrm`, where the network upscaling is also carried out. For the sake of simplicity, it is assumed that all runoff leaving a given cell would exit through a major direction. Note that multiple outlets can exist within the modelling domain. If a variable is added or removed here, then it also has to be added or removed in the subroutine `L11_config_set` in module `mo_restart` and in the subroutine `set_L11_config` in module `mo_set_netcdf_restart`

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

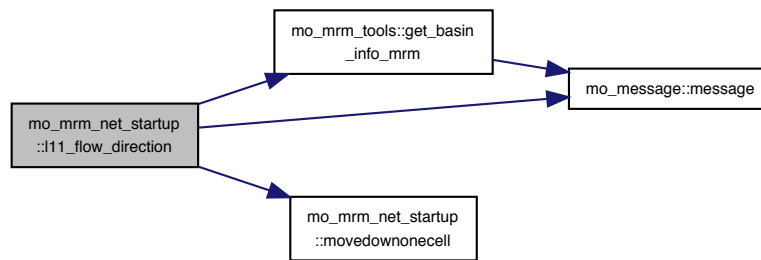
Date

Dec 2005

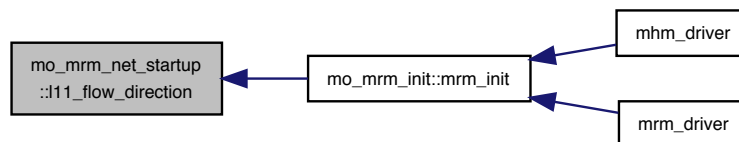
References `mo_mrm_global_variables::basin_mrm`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_kind::i4`, `mo_mrm_global_variables::l0_drasc`, `mo_mrm_global_variables::l0_facc`, `mo_mrm_global_variables::l0_fdir`, `mo_mrm_global_variables::level0`, `mo_mrm_global_variables::level11`, `mo_message::message()`, `movedownonecell()`, `mo_mrm_global_variables::nbasins`, and `mo_mrm_constants::nodata_i4`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.4 l11_fraction_sealed_floodplain()

```

subroutine, public mo_mrm_net_startup::l11_fraction_sealed_floodplain (
    integer(i4), intent(in) nLinks,
    integer(i4), dimension(:), intent(in) LCover0,
    integer(i4), dimension(:), intent(in) floodPlain0,
    real(dp), dimension(:), intent(in) areaCell0,
    real(dp), dimension(:), intent(in) nLinkAFloodPlain,
    integer(i4), intent(in) LCClassImp,
    real(dp), dimension(nlinks), intent(out) nLinkFracFPimp )
  
```

If a variable is added or removed here, then it also has to be added or removed in the subroutine `L11_config_set` in module `mo_restart` and in the subroutine `set_L11_config` in module `mo_set_netcdf_restart`.

Fraction of the flood plain with impervious cover ([Routing Network](#)"). This proportion is used to regionalize the Muskingum parameters. Samaniego et al. [13] found out that this fraction is one of the statistically significant predictor variables of peak discharge in mesoscale basins.

Parameters

in	<i>integer(i4) :: nLinks</i>	number of links for a given basin
in	<i>integer(i4) :: LCover0</i>	land cover id field (basin)
in	<i>integer(i4) :: floodPlain0</i>	floodplains of stream i (basin)

15.34.2.5 l11_link_location()

```
subroutine, public mo_mrm_net_startup::l11_link_location (
    integer(i4), intent(in) iBasin )
```

Estimate the LO (row,col) location for each routing link at level L11.

If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module mo_set_netcdf_restart

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Note

Cell location can ONLY be called after routing order is done.

Author

Luis Samaniego

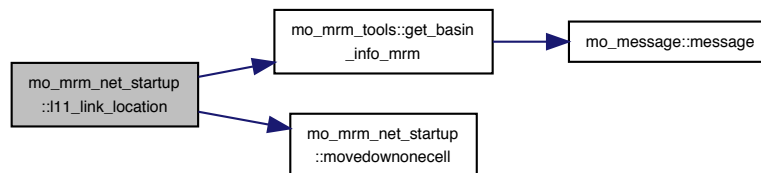
Date

Dec 2005

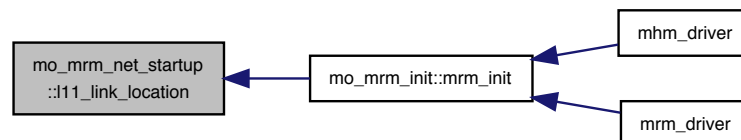
References `mo_mrm_global_variables::basin_mrm`, `mo_mrm_tools::get_basin_info_mrm()`, `movedownonecell()`, and `mo_mrm_constants::nodata_i4`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.6 l11_routing_order()

```
subroutine, public mo_mrm_net_startup::l11_routing_order (
    integer(i4), intent(in) iBasin )
```

Find routing order, headwater cells and sink.

Find routing order, headwater cells and sink.

If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module mo_set_netcdf_restart

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

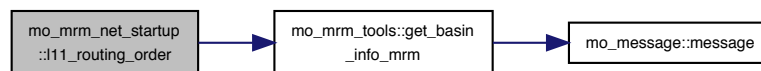
Date

Dec 2005

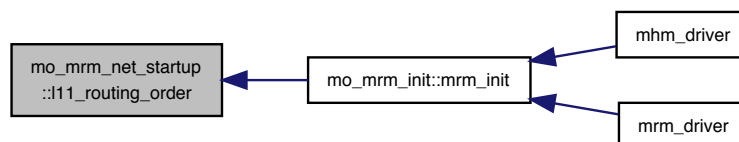
References [mo_mrm_tools::get_basin_info_mrm\(\)](#), [mo_mrm_global_variables::l11_fromn](#), and [mo_mrm_constants::nodata_i4](#).

Referenced by [mo_mrm_init::mrm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.7 l11_set_drain_outlet_gauges()

```
subroutine, public mo_mrm_net_startup::l11_set_drain_outlet_gauges (
    integer(i4), intent(in) iBasin )
```

Draining cell identification and Set gauging node.

Perform the following tasks:

- Draining cell identification (cell at L0 to draining cell outlet at L11).
- Set gauging nodes
If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_↔
config_set in module [mo_restart](#) and in the subroutine set_L11_config in module mo_set_netcdf_restart

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

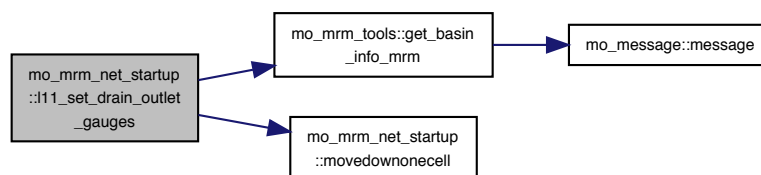
Date

Dec 2005

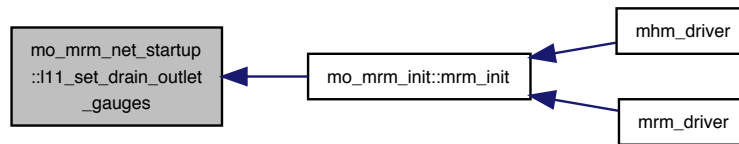
References [mo_mrm_global_variables::basin_mrm](#), [mo_mrm_tools::get_basin_info_mrm\(\)](#), [mo_mrm_global_variables::l0_fdir](#), [movedownonecell\(\)](#), and [mo_mrm_constants::nodata_i4](#).

Referenced by [mo_mrm_init::mrm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.8 l11_set_network_topology()

```
subroutine, public mo_mrm_net_startup::l11_set_network_topology (
    integer(i4), intent(in) iBasin )
```

Set network topology.

Set network topology from and to node for all links at level-11 ([Routing Network](#)).

If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module mo_set_netcdf_restart.

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

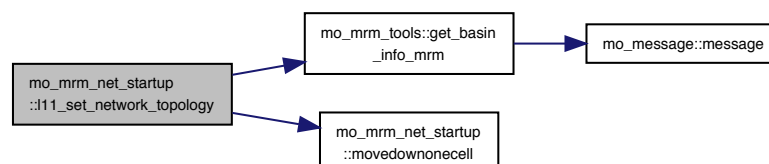
Date

Dec 2005

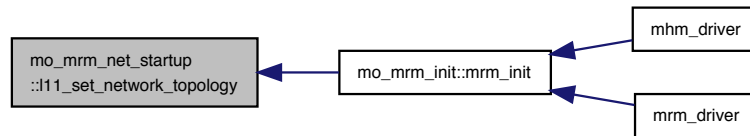
References [mo_mrm_tools::get_basin_info_mrm\(\)](#), [mo_mrm_global_variables::l11_cellcoor](#), [mo_mrm_global_variables::l11_fdir](#), [mo_mrm_global_variables::l11_fromn](#), [mo_mrm_global_variables::l11_id](#), [movedownonecell\(\)](#), and [mo_mrm_constants::nodata_i4](#).

Referenced by [mo_mrm_init::mrm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.9 l11_stream_features()

```
subroutine, public mo_mrm_net_startup::l11_stream_features (
    integer(i4), intent(in) iBasin )
```

Stream features (stream network and floodplain)

Stream features (stream network and floodplain)

If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module mo_set_netcdf_restart

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

Date

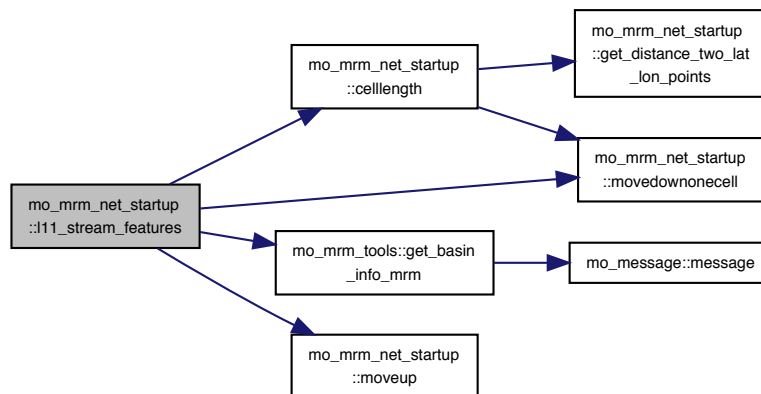
Dec 2005

stack(nNodes, 2)

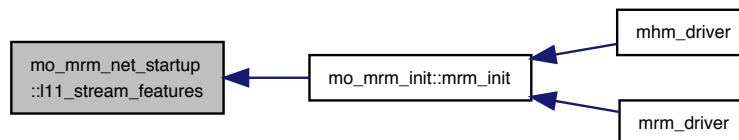
References celllength(), mo_mrm_tools::get_basin_info_mrm(), mo_mrm_global_variables::l0_elev_mrm, move-downonecell(), moveup(), mo_mrm_constants::nodata_dp, mo_mrm_constants::nodata_i4, and mo_common_variables::processmatrix.

Referenced by mo_mrm_init::mrm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



15.34.2.10 l11_variable_init()

```

subroutine, public mo_mrm_net_startup::l11_variable_init (
    integer(i4), intent(in) iBasin )

```

Cell numbering at ROUTING LEVEL-11.

Cell numbering at ROUTING LEVEL-11

List of Level- 0 and 1 cells contained within a given Level-11 cell.

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Luis Samaniego

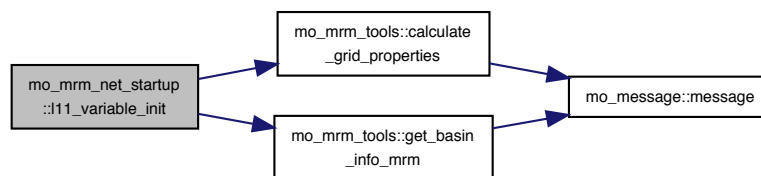
Date

Dec 2005

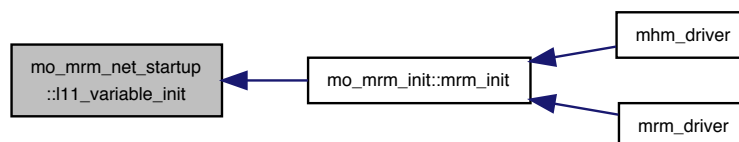
References mo_mrm_global_variables::basin_mrm, mo_mrm_tools::calculate_grid_properties(), mo_mrm_tools::get_basin_info_mrm(), mo_kind::i4, mo_mrm_global_variables::l0_areacell, mo_mrm_global_variables::l1_id, mo_mrm_global_variables::l1_l1_id, mo_mrm_global_variables::level1, mo_mrm_global_variables::level11, mo_mrm_global_variables::nbasins, mo_mrm_constants::nodata_dp, mo_mrm_constants::nodata_i4, and mo_mrm_global_variables::resolutionrouting.

Referenced by mo_mrm_init::mrm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



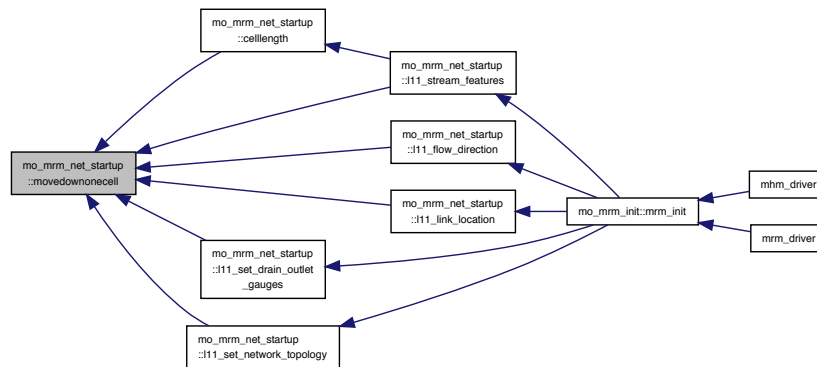
15.34.2.11 movedownonecell()

```

subroutine mo_mrm_net_startup::movedownonecell (
    integer(i4), intent(in) fDir,
    integer(i4), intent(inout) iRow,
    integer(i4), intent(inout) jCol )
  
```

Referenced by celllength(), l11_flow_direction(), l11_link_location(), l11_set_drain_outlet_gauges(), l11_set_network_topology(), and l11_stream_features().

Here is the caller graph for this function:



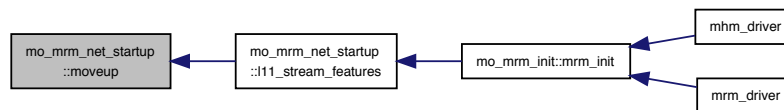
15.34.2.12 moveup()

```
subroutine mo_mrm_net_startup::moveup (
    real(dp), dimension(:, :), intent(in), allocatable elev0,
    integer(i4), dimension(:, :), intent(in), allocatable fDir0,
    integer(i4), intent(in) fi,
    integer(i4), intent(in) fj,
    integer(i4), dimension(:, :), intent(inout) ss,
    integer(i4), intent(inout) nn )
```

References mo_mrm_constants::deltah.

Referenced by l11_stream_features().

Here is the caller graph for this function:



15.35 mo_mrm_objective_function_runoff Module Reference

Objective Functions for Optimization of mHM/mRM against runoff.

.

Functions/Subroutines

- real(dp) function, public [single_objective_runoff](#) (parameterset)
Wrapper for objective functions optimizing against runoff.

- subroutine, public [multi_objective_runoff](#) (parameterset, multi_objectives)
Wrapper for multi-objective functions where at least one is regarding runoff.
- real(dp) function, public [loglikelihood](#) (parameterset)
Wrapper for loglikelihood functions.
- real(dp) function, public [loglikelihood_stddev](#) (parameterset, stddev, stddev_new, likeli_new)
Logarithmic likelihood function with removed linear trend and Lag(1)-autocorrelation.
- real(dp) function [loglikelihood_evin2013_2](#) (parameterset, regularize)
Logarithmised likelihood with linear error model and lag(1)-autocorrelation of the relative errors.
- real(dp) function [parameter_regularization](#) (paraset, prior, bounds, mask)
- real(dp) function [loglikelihood_trend_no_autocorr](#) (parameterset, stddev_old, stddev_new, likeli_new)
Logarithmic likelihood function with linear trend removed.
- real(dp) function [objective_lnnse](#) (parameterset)
Objective function of logarithmic NSE.
- real(dp) function [objective_sse](#) (parameterset)
Objective function of SSE.
- real(dp) function [objective_nse](#) (parameterset)
Objective function of NSE.
- real(dp) function [objective_equal_nse_lnnse](#) (parameterset)
Objective function equally weighting NSE and lnNSE.
- real(dp) function, dimension(2) [multi_objective_nse_lnnse](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [multi_objective_lnnse_highflow_lnnse_lowflow](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [multi_objective_lnnse_highflow_lnnse_lowflow_2](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [multi_objective_ae_fdc_lsv_nse_djf](#) (parameterset)
Multi-objective function with absolute error of Flow Duration Curves low-segment volume and nse of DJF's discharge.
- real(dp) function [objective_power6_nse_lnnse](#) (parameterset)
Objective function of combined NSE and lnNSE with power of 5 i.e. the p-norm with p=5.
- real(dp) function [objective_kge](#) (parameterset)
Objective function of KGE.
- real(dp) function [objective_multiple_gauges_kge_power6](#) (parameterset)
combined objective function based on KGE raised to the power 6
- subroutine, public [extract_runoff](#) (gaugeld, runoff, runoff_agg, runoff_obs, runoff_obs_mask)
extracts runoff data from global variables
- subroutine [eval](#) (parameterset, runoff, basin_avg_tws)
returns mHM_eval or mRM_eval given preprocessor flag

15.35.1 Detailed Description

Objective Functions for Optimization of mHM/mRM against runoff.

.

This module provides a wrapper for several objective functions used to optimize mRM/mHM against runoff.

If the objective contains besides runoff another variable like TWS move it to [mHM/mo_objective_function.f90](#). If it is only regarding runoff implement it here.

All the objective functions are supposed to be minimized!

- (1) SO: Q: 1.0 - NSE
- (2) SO: Q: 1.0 - lnNSE
- (3) SO: Q: 1.0 - 0.5*(NSE+lnNSE)
- (4) SO: Q: -1.0 * loglikelihood with trend removed from absolute errors and then lag(1)-autocorrelation removed
- (5) SO: Q: $((1-NSE)**6 + (1-\lnNSE)**6)**(1/6)$

- (6) SO: Q: SSE
 (7) SO: Q: $-1.0 * \text{loglikelihood with trend removed from absolute errors}$
 (8) SO: Q: $-1.0 * \text{loglikelihood with trend removed from the relative errors and then lag(1)-autocorrelation removed}$
 (9) SO: Q: 1.0 - KGE (Kling-Gupta efficiency measure)
 (14) SO: Q: $\text{sum}[(1.0 - \text{KGE}_i) / n\text{Gauges}]^{**6}^{**}(1/6) > \text{combination of KGE of every gauging station based on a power-6 norm}$
 (16) MO: Q: 1st objective: 1.0 - NSE
 Q: 2nd objective: 1.0 - lnNSE
 (18) MO: Q: 1st objective: 1.0 - lnNSE(Q_highflow) (95% percentile)
 Q: 2nd objective: 1.0 - lnNSE(Q_lowflow) (5% of data range)
 (19) MO: Q: 1st objective: 1.0 - lnNSE(Q_highflow) (non-low flow)
 Q: 2nd objective: 1.0 - lnNSE(Q_lowflow) (5% of data range)eshold for Q
 (20) MO: Q: 1st objective: absolute difference in FDC's low-segment volume
 Q: 2nd objective: 1.0 - NSE of discharge of months DJF

Authors

Juliane Mai

Date

Dec 2012

15.35.2 Function/Subroutine Documentation

15.35.2.1 eval()

```
subroutine mo_mrm_objective_function_runoff::eval (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), dimension(:,:), intent(out), optional, allocatable runoff,
    real(dp), dimension(:,:), intent(out), optional, allocatable basin_avg_tws )
```

returns mHM_eval or mRM_eval given preprocessor flag

call mHM_eval if MRM2MHM preprocessor flag is used while compilation or mRM_eval otherwise

Parameters

in	<i>integer(i4) :: parameterset</i>	- mHM or mRM parameter set
out	<i>real(dp), optional :: runoff(:)</i>	- simulated runoff

Author

Stephan Thober

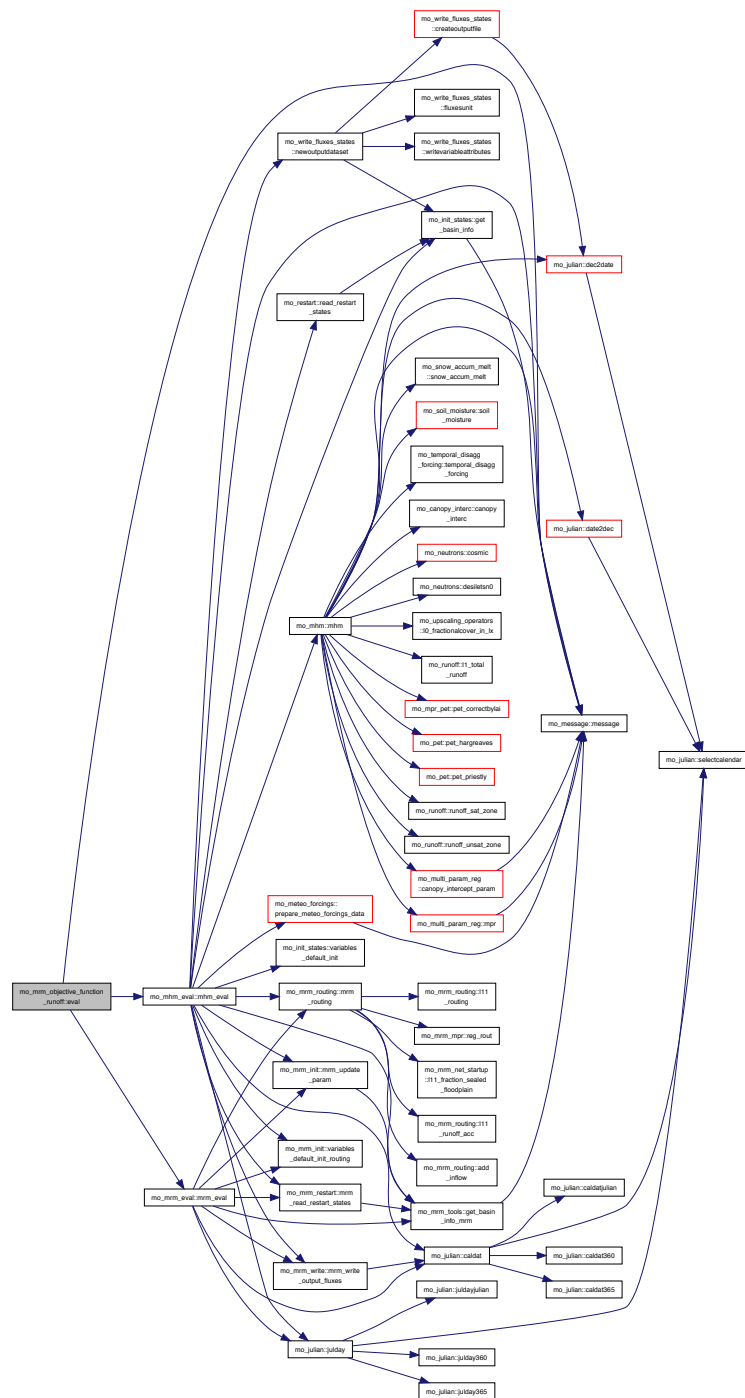
Date

Oct 2015

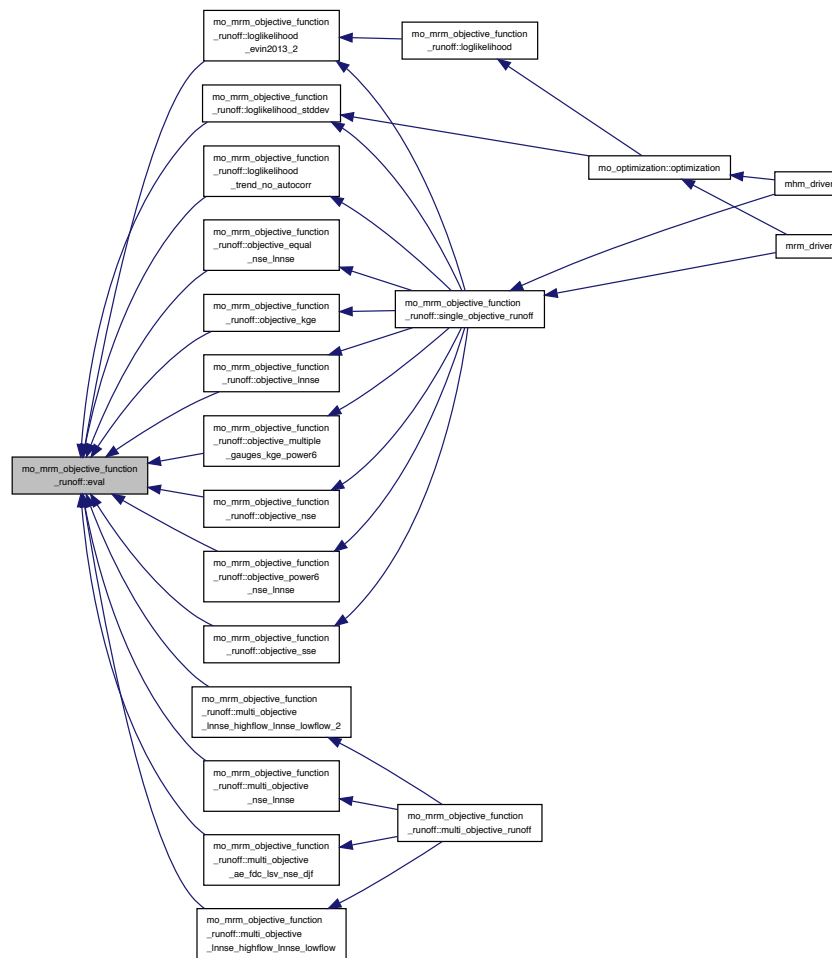
References `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, and `mo_mrm_eval::mrm_eval()`.

Referenced by `loglikelihood_evin2013_2()`, `loglikelihood_stddev()`, `loglikelihood_trend_no_autocorr()`, `multi_↵objective_ae_fdc_lsv_nse_djf()`, `multi_objective_lnnse_highflow_lnnse_lowflow()`, `multi_objective_lnnse_highflow_↵_lnnse_lowflow_2()`, `multi_objective_nse_lnnse()`, `objective_equal_nse_lnnse()`, `objective_kge()`, `objective_lnnse()`, `objective_multiple_gauges_kge_power6()`, `objective_nse()`, `objective_power6_nse_lnnse()`, and `objective_sse()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.2 extract_runoff()

```

subroutine, public mo_mrm_objective_function_runoff::extract_runoff (
  integer(i4), intent(in) gaugeId,
  real(dp), dimension(:,:), intent(in) runoff,
  real(dp), dimension(:), intent(out), allocatable runoff_agg,
  real(dp), dimension(:), intent(out), allocatable runoff_obs,
  logical, dimension(:), intent(out), allocatable runoff_obs_mask )

```

extracts runoff data from global variables

extracts simulated and measured runoff from global variables, such that they overlay exactly. For measured runoff, only the runoff during the evaluation period are cut, not succeeding nodata values. For simulated runoff, warming days as well as succeeding nodata values are neglected and the simulated runoff is aggregated to the resolution of the observed runoff.

Parameters

in	<i>integer(i4) :: gaugeID</i>	- ID of the current gauge to process
----	-------------------------------	--------------------------------------

Parameters

in	<i>real(dp) :: runoff(:)</i>	- simulated runoff at this gauge
out	<i>real(dp) :: runoff_agg(:)</i>	- aggregated simulated runoff at this gauge
out	<i>real(dp) :: runoff_obs(:)</i>	- extracted observed runoff
out	<i>logical :: runoff_obs_mask(:)</i>	- masking non-negative values in runoff_obs

Author

Stephan Thober

Date

Jan 2015

References `mo_mrm_global_variables::evalper`, `mo_mrm_global_variables::gauge`, `mo_message::message()`, `mo_mrm_global_variables::nmeasperday`, `mo_mrm_global_variables::ntstepday`, and `mo_mrm_global_variables::warmingdays_mrm`.

Referenced by `loglikelihood_evin2013_2()`, `loglikelihood_stddev()`, `loglikelihood_trend_no_autocorr()`, `multi_objective_ae_fdc_lsv_nse_djf()`, `multi_objective_lnnse_highflow_lnnse_lowflow()`, `multi_objective_lnnse_highflow_lnnse_lowflow_2()`, `multi_objective_nse_lnnse()`, `objective_equal_nse_lnnse()`, `objective_kge()`, `mo_objective_function::objective_kge_q_et()`, `mo_objective_function::objective_kge_q_rmse_et()`, `mo_objective_function::objective_kge_q_rmse_tws()`, `mo_objective_function::objective_kge_q_sm_corr()`, `objective_lnnse()`, `objective_multiple_gauges_kge_power6()`, `objective_nse()`, `objective_power6_nse_lnnse()`, and `objective_sse()`.

Here is the call graph for this function:



Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	--	--

Returns

real(dp) :: loglikelihood — loglikelihood function value

Note

The wrapped functions must return real loglikelihoods, i.e. errors are either known from observations or modelled.

Author

Juliane Mai

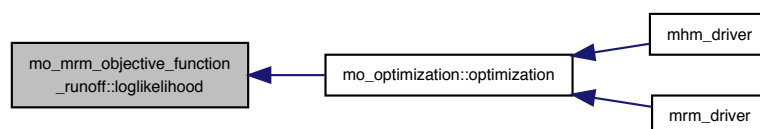
Date

Dec 2012

References `loglikelihood_evin2013_2()`, `mo_common_variables::opti_function`, and `mo_common_variables::opti_↵` method.

Referenced by `mo_optimization::optimization()`.

Here is the caller graph for this function:



15.35.2.4 loglikelihood_evin2013_2()

```
real(dp) function mo_mrm_objective_function_runoff::loglikelihood_evin2013_2 (
    real(dp), dimension(:), intent(in) parameterset,
    logical, intent(in), optional regularize )
```

Logarithmised likelihood with linear error model and lag(1)-autocorrelation of the relative errors.

This loglikelihood uses a linear error model and a lag(1)-autocorrelation on the relative errors. This is approach 2 of the paper Evin et al. (WRR, 2013).

This is opti_function = 8.

mHM then adds two extra (local) parameters for the error model in mhm_driver, which get optimised together with the other, global parameters.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
in	<i>real(dp) :: stddev_old</i>	standard deviation of data
out	<i>real(dp), optional :: stddev_new</i>	standard deviation of errors with removed trend and correlation between model run using parameter set and observation
out	<i>real(dp), optional :: likeli_new</i>	logarithmic likelihood determined with stddev_new instead of stddev

Returns

real(dp) :: loglikelihood_evin2013_2 — logarithmic likelihood using given stddev but remove optimal trend and lag(1)-autocorrelation in errors (absolute between running model with parameterset and observation)

Note

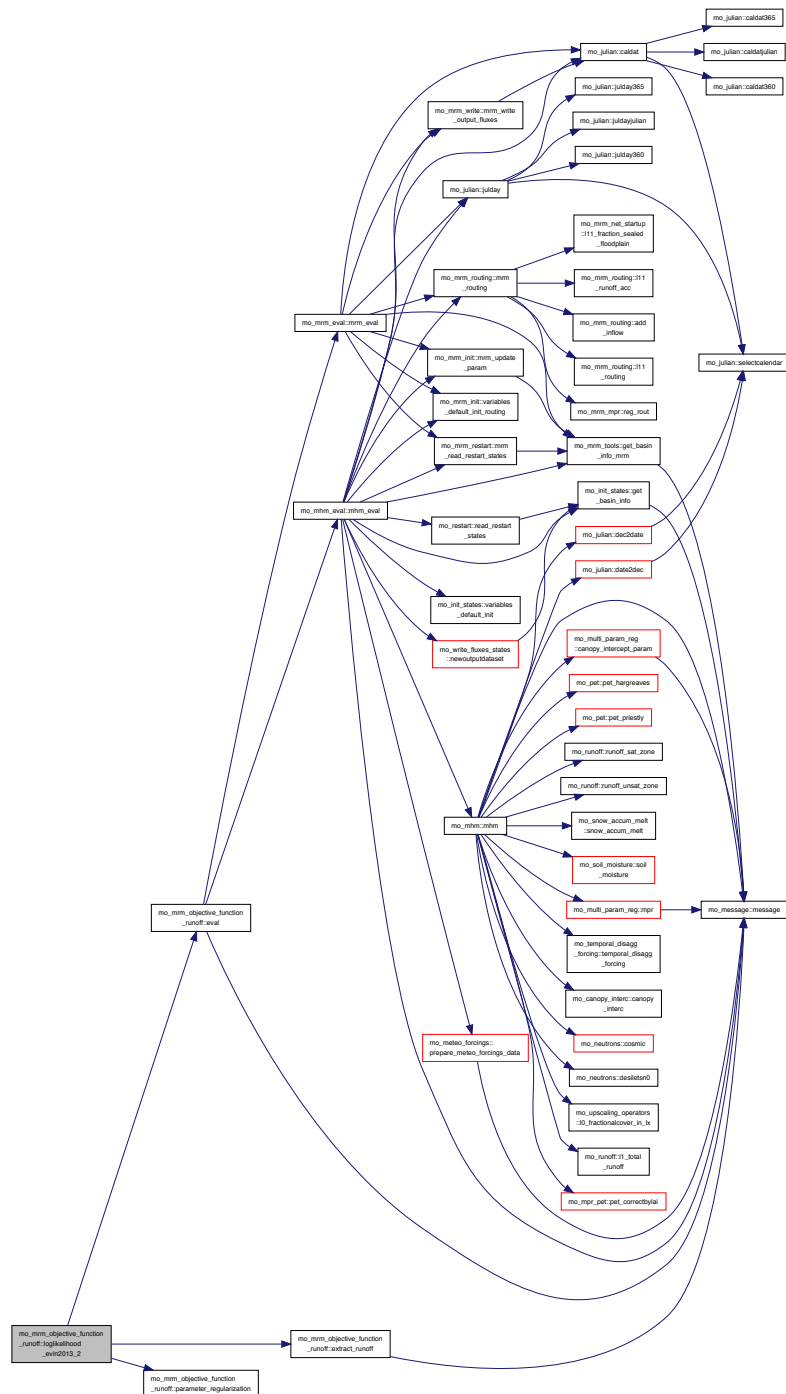
Does not work with MCMC yet.

Author

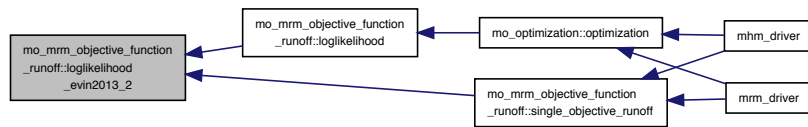
Juliane Mai and Matthias Cuntz

Mar 2014

Referenced by `loglikelihood()`, and `single_objective_runoff()`.



Here is the caller graph for this function:



15.35.2.5 loglikelihood_stddev()

```

real(dp) function, public mo_mrm_objective_function_runoff::loglikelihood_stddev (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), intent(in) stddev,
    real(dp), intent(out), optional stddev_new,
    real(dp), intent(out), optional likeli_new )
  
```

Logarithmic likelihood function with removed linear trend and Lag(1)-autocorrelation.

The logarithmic likelihood function is used when mHM runs in MCMC mode.

It can also be used for optimization when selecting the likelihood in the namelist as *opti_function*.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
in	<i>real(dp) :: stddev</i>	standard deviation of data
out	<i>real(dp), optional :: stddev_new</i>	standard deviation of errors with removed trend and correlation between model run using parameter set and observation
out	<i>real(dp), optional :: likeli_new</i>	logarithmic likelihood determined with stddev_new instead of stddev

Returns

real(dp) :: loglikelihood_stddev — logarithmic likelihood using given stddev but remove optimal trend and lag(1)-autocorrelation in errors (absolute between running model with parameterset and observation)

Note

Input values must be floating points.

Author

Juliane Mai

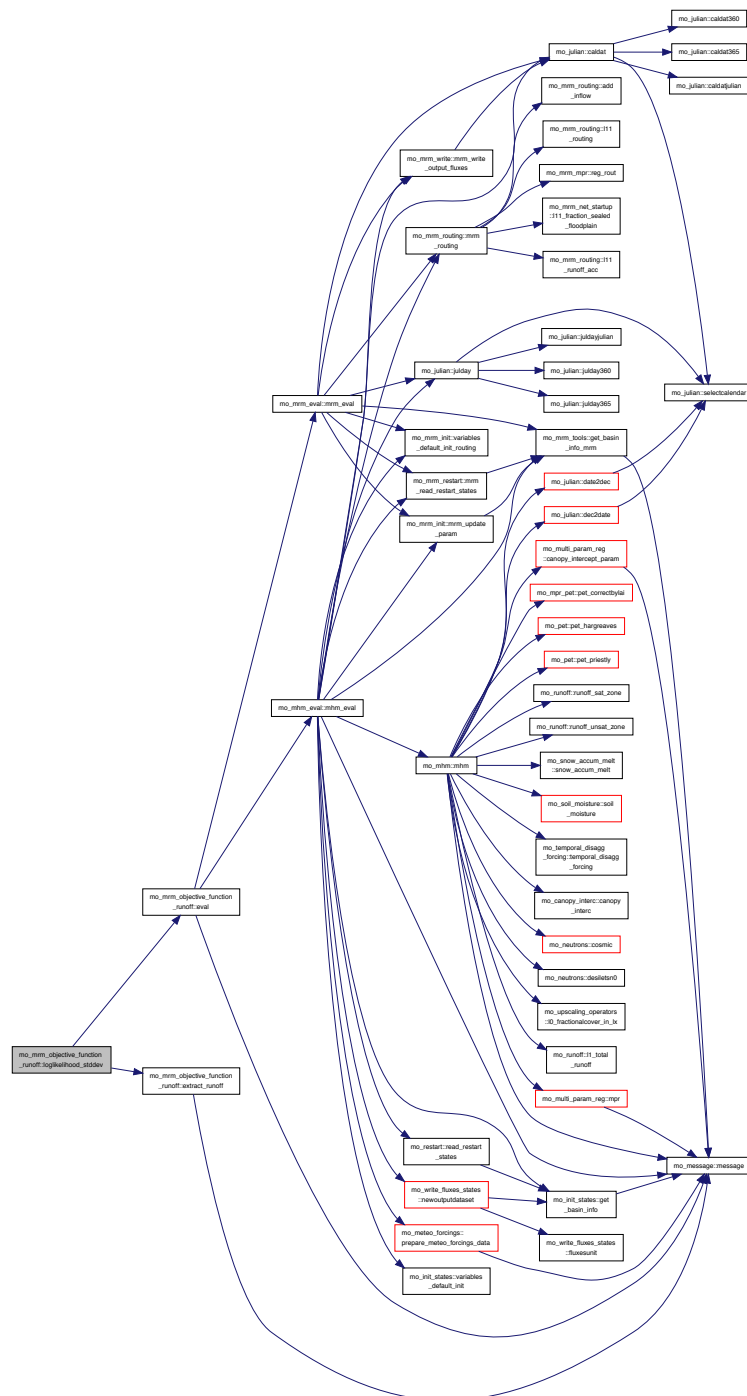
Date

Dec 2012

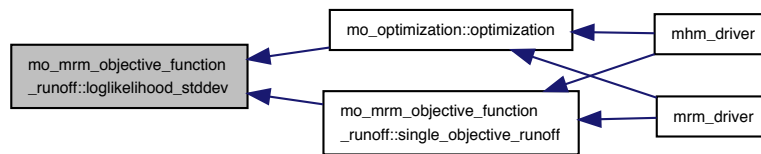
References *eval()*, and *extract_runoff()*.

Referenced by *mo_optimization::optimization()*, and *single_objective_runoff()*.

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.6 loglikelihood_trend_no_autocorr()

```

real(dp) function mo_mrm_objective_function_runoff::loglikelihood_trend_no_autocorr (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), intent(in) stddev_old,
    real(dp), intent(out), optional stddev_new,
    real(dp), intent(out), optional likeli_new )

```

Logarithmic likelihood function with linear trend removed.

The logarithmic likelihood function is used when mHM runs in MCMC mode.

It can also be used for optimization when selecting the likelihood in the namelist as *opti_function*.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
in	<i>real(dp) :: stddev</i>	standard deviation of data
out	<i>real(dp), optional :: stddev_new</i>	standard deviation of errors with removed trend between model run using parameter set and observation
out	<i>real(dp), optional :: likeli_new</i>	logarithmic likelihood determined with stddev_new instead of stddev

Returns

real(dp) :: loglikelihood_trend_no_autocorr — logarithmic likelihood using given stddev but remove optimal trend in errors (absolute between running model with parameterset and observation)

Note

Input values must be floating points.

Author

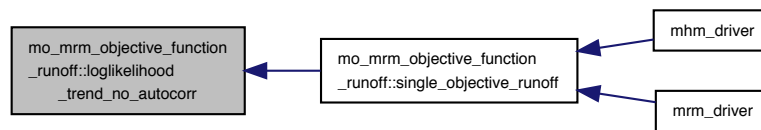
Juliane Mai and Matthias Cuntz

Mar 2014

Referenced by `single_objective_runoff()`.

[illegible]

Here is the caller graph for this function:



15.35.2.7 multi_objective_ae_fdc_lsv_nse_djf()

```

real(dp) function, dimension(2) mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf (
    real(dp), dimension(:), intent(in) parameterset )
  
```

Multi-objective function with absolute error of Flow Duration Curves low-segment volume and nse of DJF's discharge.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

The first objective is using the routine "FlowDurationCurves" from "mo_signatures" to determine the low-segment volume of the FDC. The objective is the absolute difference between the observed volume and the simulated volume. For the second objective the discharge of the winter months December, January and February are extracted from the time series. The objective is then the Nash-Sutcliffe efficiency NSE of the observed winter discharge against the simulated winter discharge.

The observed data Q_{obs} are global in this module. To calibrate this objective you need a multi-objective optimizer like PA-DDS.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp), dimension(2) :: multi_objective_ae_fdc_lsv_nse_djf — objective function value (which will be e.g. minimized by an optimization routine like PA-DDS)

Note

Input values must be floating points.

Actually, $1 - ae$ and $1 - nse$ will be returned such that it can be minimized.

Author

Juliane Mai

15.35.2.8 multi_objective_lnnse_highflow_lnnse_lowflow()

```
real(dp) function, dimension(2) mo_mrm_objective_function_runoff::multi_objective_lnnse_↵
highflow_lnnse_lowflow (
    real(dp), dimension(:), intent(in) parameterset )
```

Multi-objective function with NSE and lnNSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

A timepoint t of the observed data is marked as a lowflow timepoint t_{low} if

$$Q_{obs}(t) < \min(Q_{obs}) + 0.05 * (\max(Q_{obs}) - \min(Q_{obs}))$$

and a timepoint t of the observed data is marked as a highflow timepoint t_{high} if

$$t_{high} \text{ if } Q_{obs}(i) > \text{percentile}(Q_{obs}, 95.)$$

This timepoint identification is only performed for the observed data.

The first objective is the logarithmic Nash-Sutcliffe model efficiency coefficient \lnNSE_{high} of discharge values at high-flow timepoints

$$\lnNSE_{high} = 1 - \frac{\sum_{i=1}^{N_{high}} (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

. The second objective is the logarithmic Nash-Sutcliffe model efficiency coefficient \lnNSE_{low} of discharge values at low-flow timepoints

$$\lnNSE_{low} = 1 - \frac{\sum_{i=1}^{N_{low}} (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

. Both objectives are returned. The observed data Q_{obs} are global in this module. To calibrate this objective you need a multi-objective optimizer like PA-DDS.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	--	--

Returns

real(dp), dimension(2) :: multi_objective_lnnse_highflow_lnnse_lowflow — objective function value (which will be e.g. minimized by an optimization routine like PA-DDS)

Note

Input values must be floating points.

Actually, $1 - nse$ and $1 - lnnse$ will be returned such that it can be minimized.

Author

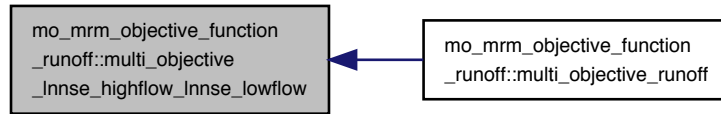
Juliane Mai

Oct 2015

Referenced by multi_objective_runoff().

[illegible]

Here is the caller graph for this function:



15.35.2.9 multi_objective_lnnse_highflow_lnnse_lowflow_2()

```

real(dp) function, dimension(2) mo_mrm_objective_function_runoff::multi_objective_lnnse_↵
highflow_lnnse_lowflow_2 (
    real(dp), dimension(:), intent(in) parameterset )
  
```

Multi-objective function with NSE and lnNSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

A timepoint t of the observed data is marked as a lowflow timepoint t_{low} if

$$Q_{obs}(t) < \min(Q_{obs}) + 0.05 * (\max(Q_{obs}) - \min(Q_{obs}))$$

and all other timepoints are marked as a highflow timepoints t_{high} . This timepoint identification is only performed for the observed data.

The first objective is the logarithmic Nash-Sutcliffe model efficiency coefficient $lnNSE_{high}$ of discharge values at high-flow timepoints

$$lnNSE_{high} = 1 - \frac{\sum_{i=1}^{N_{high}} (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

. The second objective is the logarithmic Nash-Sutcliffe model efficiency coefficient $lnNSE_{low}$ of discharge values at low-flow timepoints

$$lnNSE_{low} = 1 - \frac{\sum_{i=1}^{N_{low}} (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

. Both objectives are returned. The observed data Q_{obs} are global in this module. To calibrate this objective you need a multi-objective optimizer like PA-DDS.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	--	--

Returns

real(dp), dimension(2) :: multi_objective_lnnse_highflow_lnnse_lowflow_2 — objective function value (which will be e.g. minimized by an optimization routine like PA-DDS)

Note

Input values must be floating points.
Actually, $1 - nse$ and $1 - lnnse$ will be returned such that it can be minimized.

Author

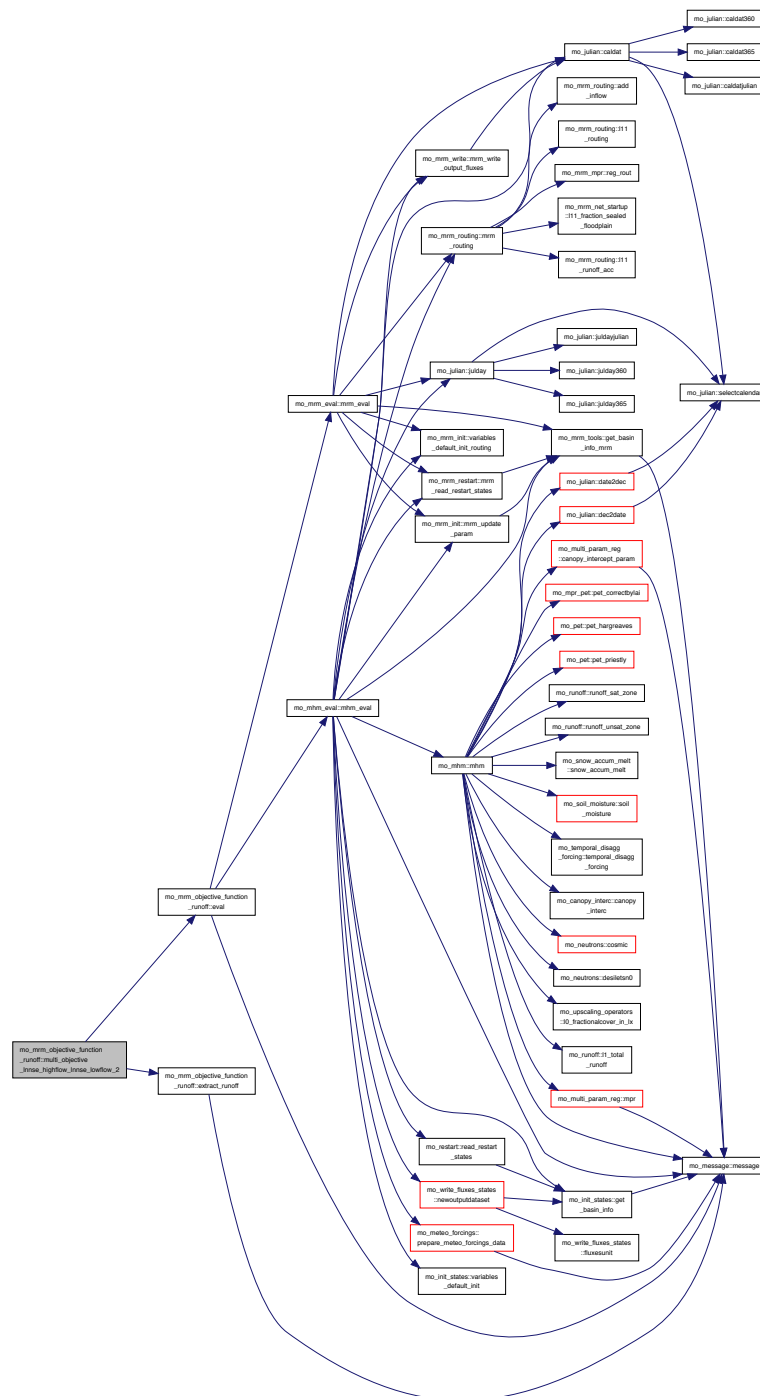
Juliane Mai

Date

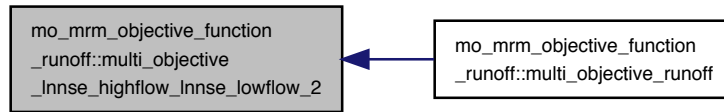
Oct 2015

References `eval()`, and `extract_runoff()`.
Referenced by `multi_objective_runoff()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.10 multi_objective_nse_lnnse()

```
real(dp) function, dimension(2) mo_mrm_objective_function_runoff::multi_objective_nse_lnnse (
    real(dp), dimension(:), intent(in) parameterset )
```

Multi-objective function with NSE and lnNSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the Nash-Sutcliffe model efficiency coefficient NSE

$$NSE = 1 - \frac{\sum_{i=1}^N (Q_{obs}(i) - Q_{model}(i))^2}{\sum_{i=1}^N (Q_{obs}(i) - \bar{Q}_{obs})^2}$$

and the logarithmic Nash-Sutcliffe model efficiency coefficient $lnNSE$

$$lnNSE = 1 - \frac{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

are calculated and both returned. The observed data Q_{obs} are global in this module. To calibrate this objective you need a multi-objective optimizer like PA-DDS.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp), dimension(2) :: multi_objective_nse_lnnse — objective function value (which will be e.g. minimized by an optimization routine like PA-DDS)

Note

Input values must be floating points.

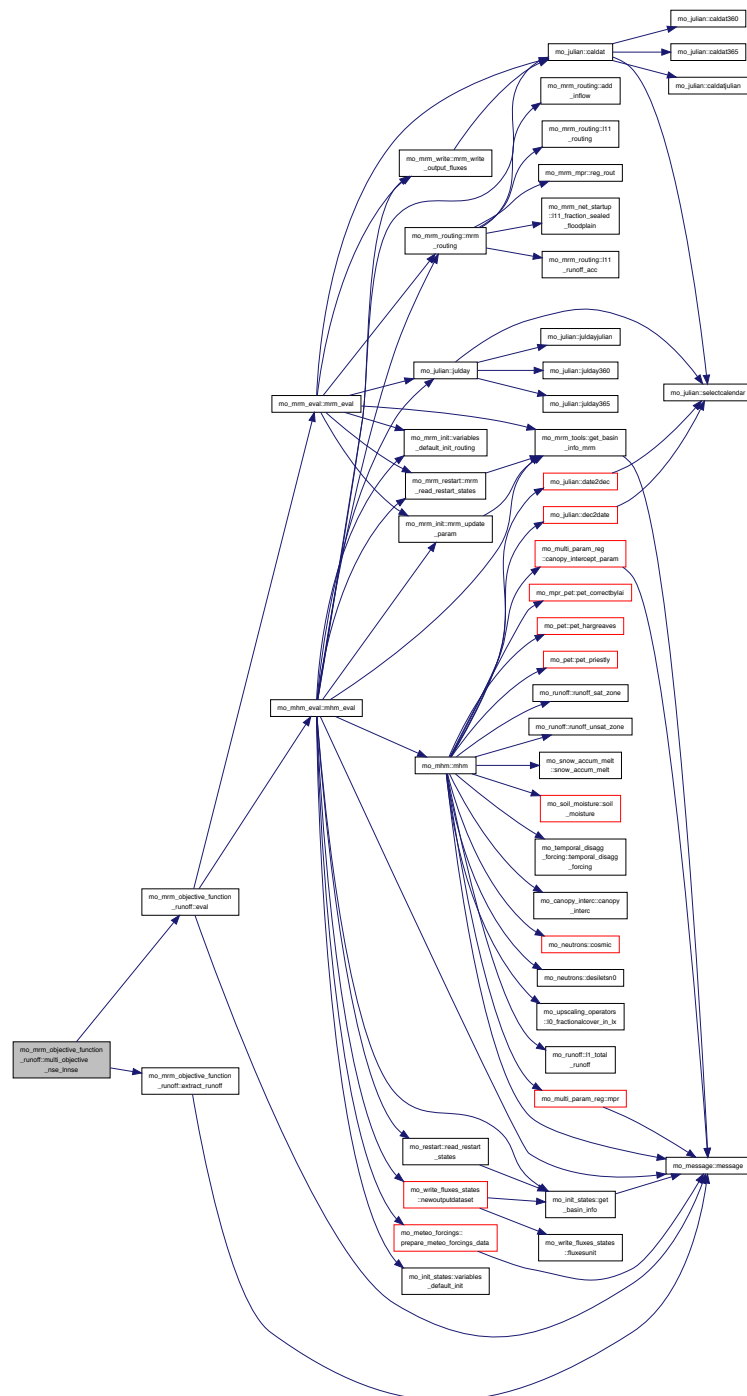
Actually, $1 - nse$ and $1 - lnnse$ will be returned such that it can be minimized.

Author

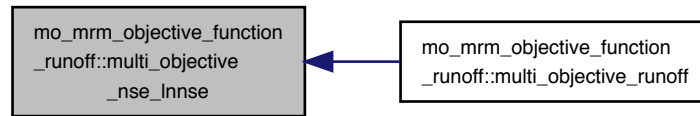
Juliane Mai

Oct 2015

Referenced by multi_objective_runoff().



Here is the caller graph for this function:



15.35.2.11 multi_objective_runoff()

```

subroutine, public mo_mrm_objective_function_runoff::multi_objective_runoff (
    real(dp), dimension(:), intent(in) parameterset,
    real(dp), dimension(:), intent(out), allocatable multi_objectives )
  
```

Wrapper for multi-objective functions where at least one is regarding runoff.

The functions selects the objective function case defined in a namelist, i.e. the global variable *opti_function*. It return the multiple objective function values for a specific parameter set.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
out	<i>real(dp) :: multi_objectives(:)</i>	1D-array with multiple objective function values

Note

Input values must be floating points.

Author

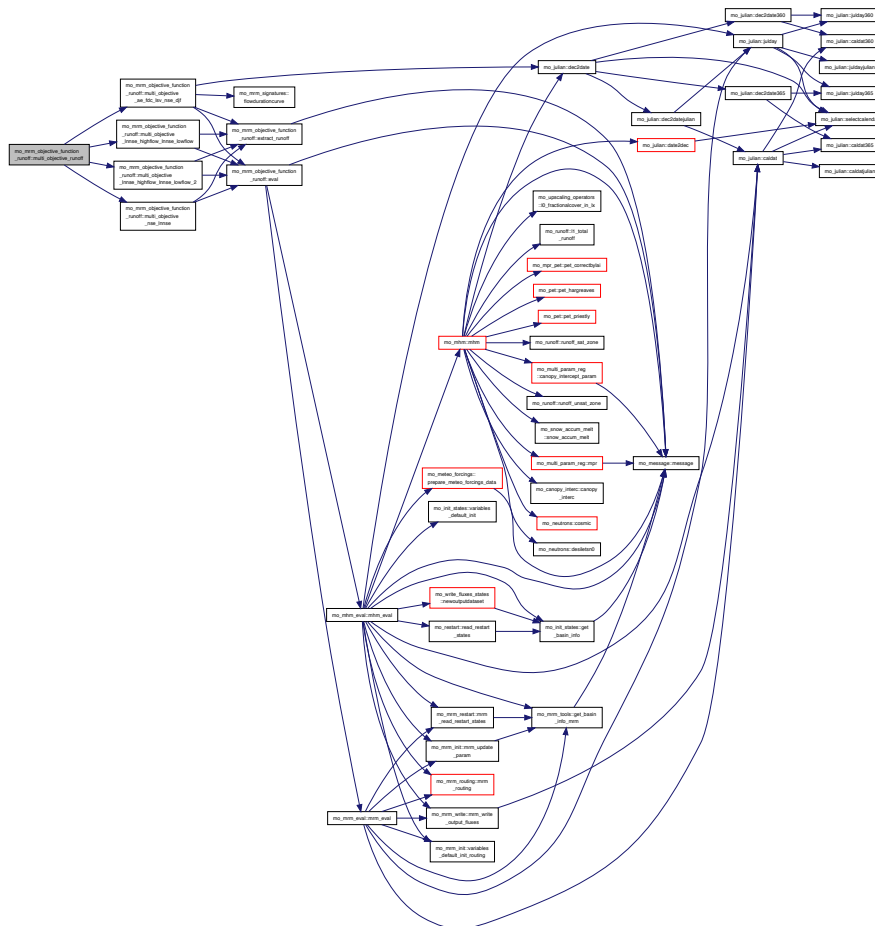
Juliane Mai

Date

Oct 2015

References `multi_objective_ae_fdc_lsv_nse_djf()`, `multi_objective_lnnse_highflow_lnnse_lowflow()`, `multi_objective_lnnse_highflow_lnnse_lowflow_2()`, `multi_objective_nse_lnnse()`, and `mo_common_variables::opti_` function.

Here is the call graph for this function:



15.35.2.12 objective_equal_nse_lnnse()

```
real(dp) function mo_mrm_objective_function_runoff::objective_equal_nse_lnnse (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function equally weighting NSE and lnNSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the Nash-Sutcliffe model efficiency coefficient *NSE*

$$NSE = 1 - \frac{\sum_{i=1}^N (Q_{obs}(i) - Q_{model}(i))^2}{\sum_{i=1}^N (Q_{obs}(i) - \bar{Q}_{obs})^2}$$

and the logarithmic Nash-Sutcliffe model efficiency coefficient *lnNSE*

$$lnNSE = 1 - \frac{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

are calculated and added up equally weighted:

$$obj_value = \frac{1}{2}(NSE + \ln NSE)$$

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	--	--

Returns

real(dp) :: objective_equal_nse_lnnse — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.
Actually, $1 - 0.5 * (nse + \ln nse)$ will be returned such that it can be minimized.

Author

Juliane Mai

Date

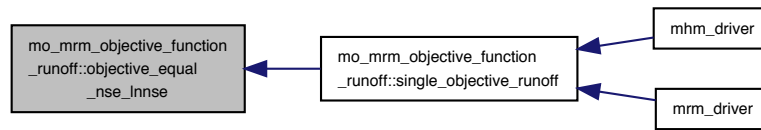
May 2013

References `eval()`, and `extract_runoff()`.

Referenced by `single_objective_runoff()`.

Generated on December 1, 2017

Here is the caller graph for this function:



15.35.2.13 objective_kge()

```
real(dp) function mo_mrm_objective_function_runoff::objective_kge (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of KGE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore, the Kling-Gupta model efficiency coefficient KGE

$$KGE = 1.0 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment correlation coefficient

α = ratio of simulated mean to observed mean

β = ratio of simulated standard deviation to observed standard deviation

is calculated and the objective function is

$$obj_value = 1.0 - KGE$$

$(1 - KGE)$ is the objective since we always apply minimization methods. The minimal value of $(1 - KGE)$ is 0 for the optimal KGE of 1.0.

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

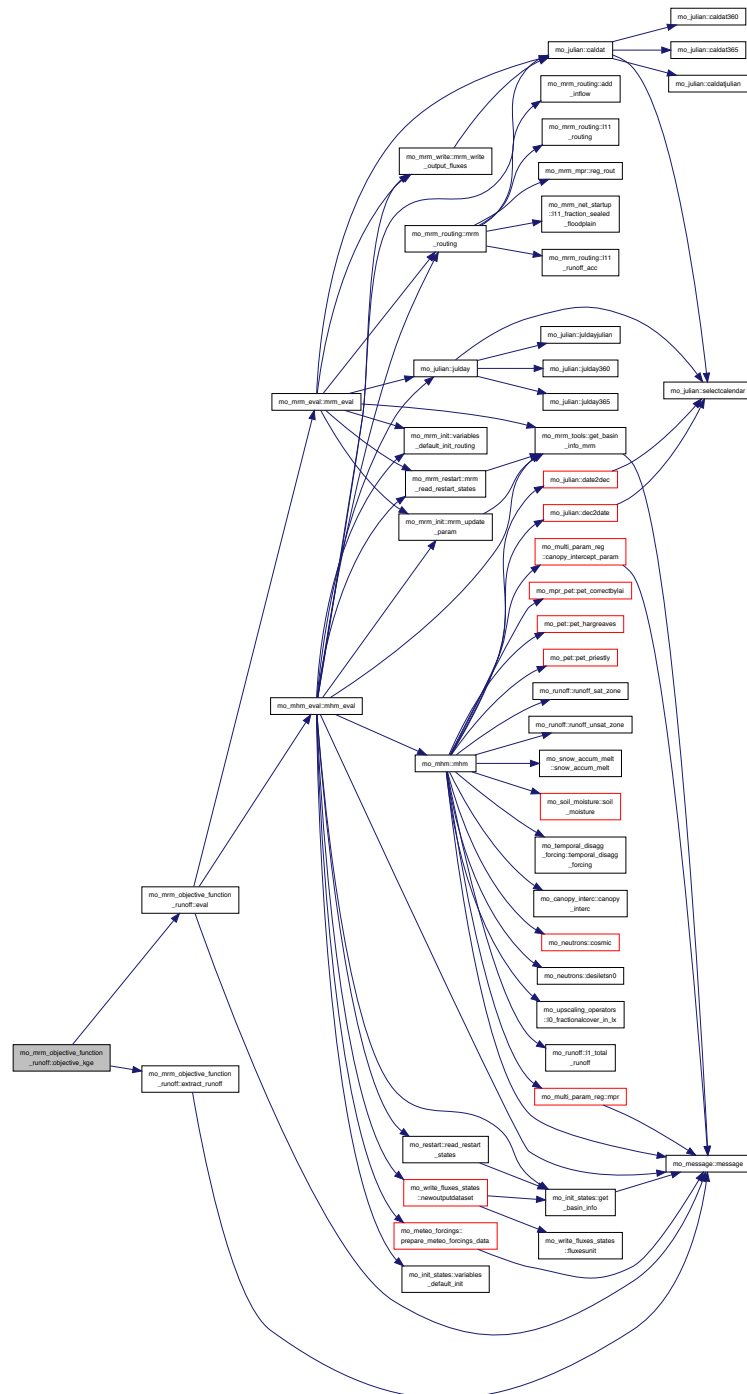
real(dp) :: objective_kge — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

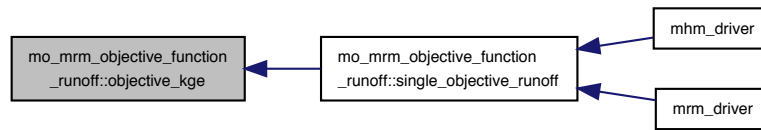
Input values must be floating points.

Actually, KGE will be returned such that it can be minimized. Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." Journal of Hydrology 377.1 (2009): 80-91.

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.14 objective_lnnse()

```
real(dp) function mo_mrm_objective_function_runoff::objective_lnnse (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of logarithmic NSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the logarithmic Nash-Sutcliffe model efficiency coefficient $\ln NSE$

$$\ln NSE = 1 - \frac{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

is calculated.

$$obj_value = \ln NSE$$

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_lnnse — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.
Actually, $1 - \ln nse$ will be returned such that it can be minimized.

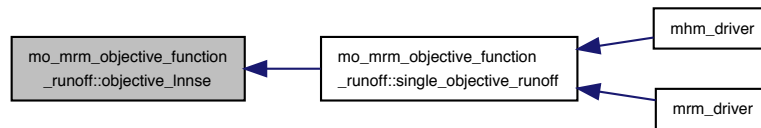
Author

Juliane Mai

May 2013

Referenced by `single_objective_runoff()`.

Here is the caller graph for this function:



15.35.2.15 objective_multiple_gauges_kge_power6()

```
real(dp) function mo_mrm_objective_function_runoff::objective_multiple_gauges_kge_power6 (
    real(dp), dimension(:), intent(in) parameterset )
```

combined objective function based on KGE raised to the power 6

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore, the Kling-Gupta model efficiency coefficient KGE for a given gauging station

$$KGE = 1.0 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment correlation coefficient

α = ratio of simulated mean to observed mean

β = ratio of simulated standard deviation to observed standard deviation

is calculated and the objective function for a given gauging station (i) is

$$\phi_i = 1.0 - KGE_i$$

ϕ_i is the objective since we always apply minimization methods. The minimal value of ϕ_i is 0 for the optimal KGE of 1.0.

Finally, the overall OF is estimated based on the power-6 norm to combine the ϕ_i from all gauging stations (N).

$$OF = \sqrt[6]{\sum ((1.0 - KGE_i)/N)^6}$$

.

The observed data \form#139 are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_multiple_gauges_kge_power6 — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Actually, OF will be returned such that it can be minimized. Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." Journal of Hydrology 377.1 (2009): 80-91.

Author

Rohini Kumar

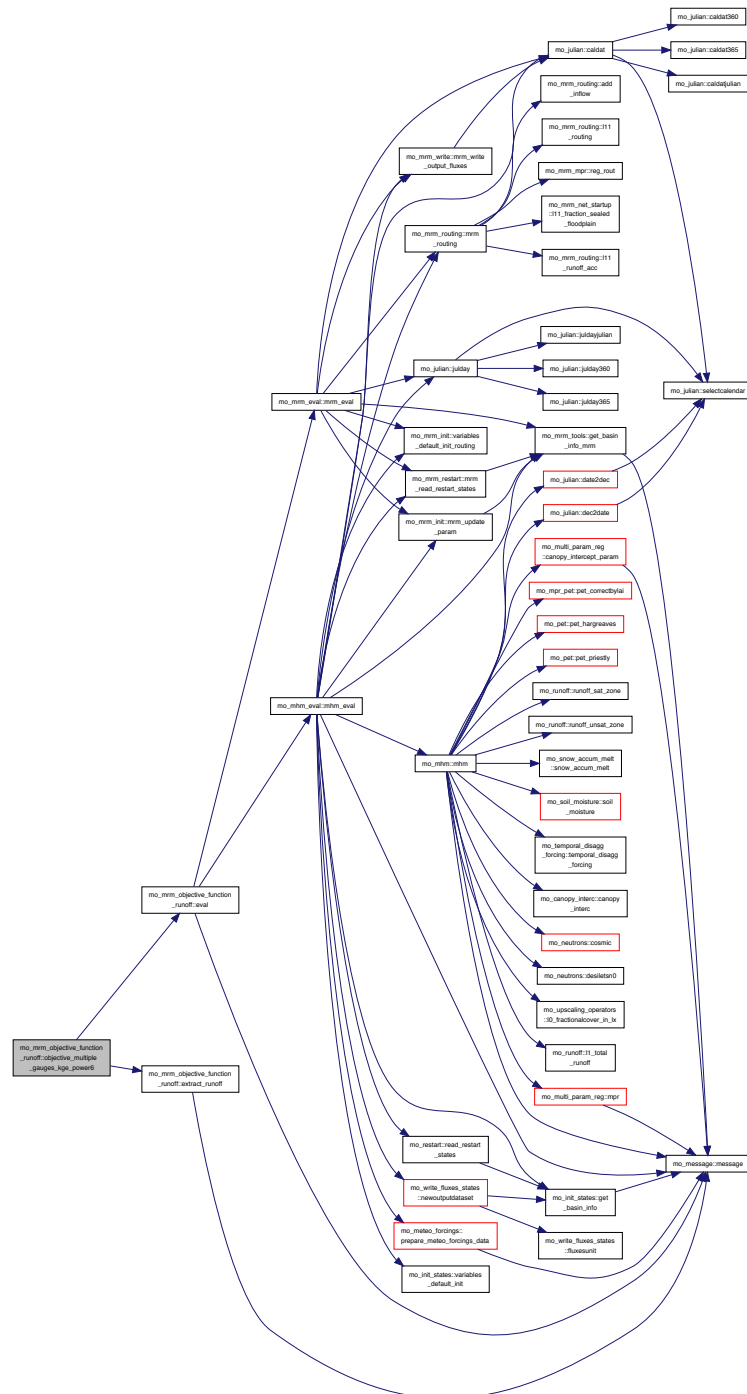
Date

March 2015

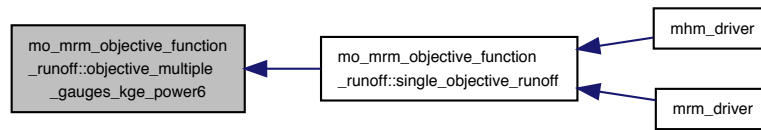
References eval(), and extract_runoff().

Referenced by single_objective_runoff().

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.16 objective_nse()

```
real(dp) function mo_mrm_objective_function_runoff::objective_nse (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of NSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the Nash-Sutcliffe model efficiency coefficient NSE

$$NSE = 1 - \frac{\sum_{i=1}^N (Q_{obs}(i) - Q_{model}(i))^2}{\sum_{i=1}^N (Q_{obs}(i) - \bar{Q}_{obs})^2}$$

is calculated and the objective function is

$$obj_value = 1 - NSE$$

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_nse — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.
Actually, $1 - NSE$ will be returned such that it can be minimized.

Author

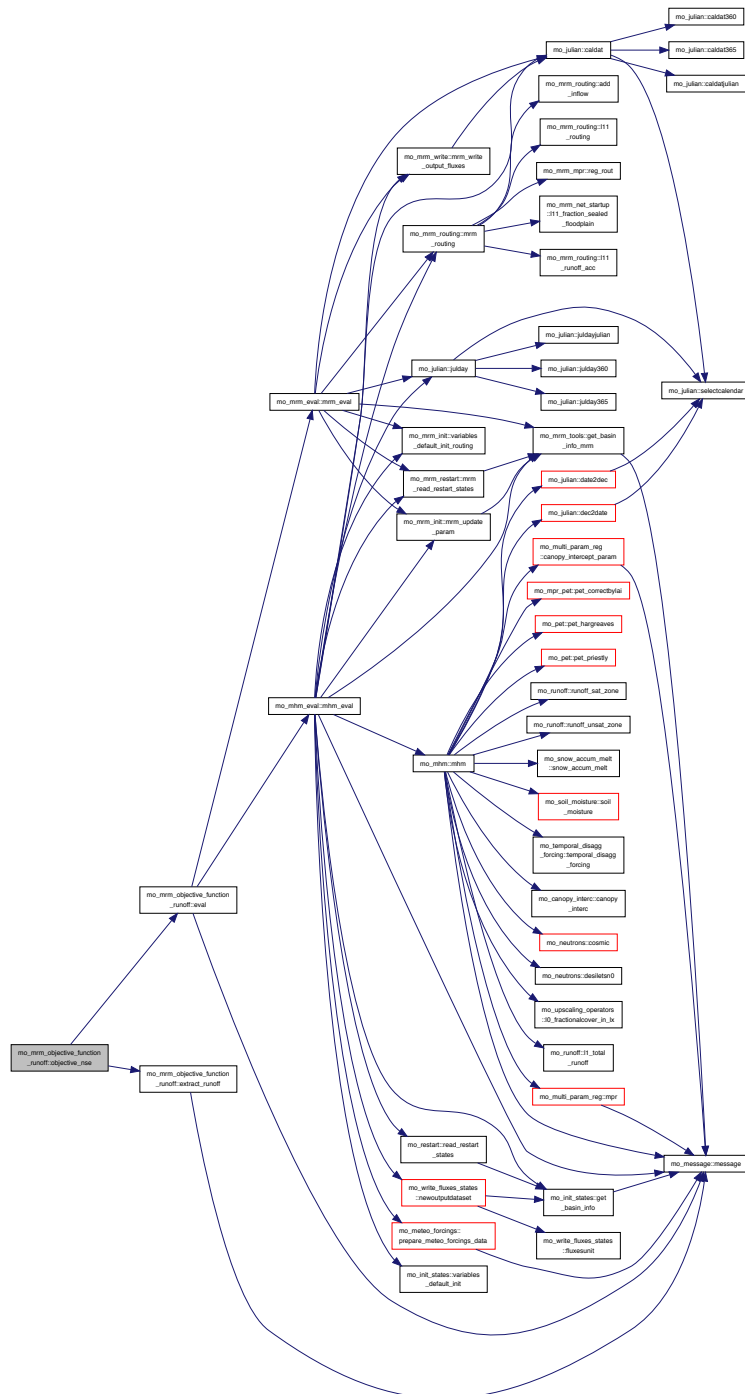
Juliane Mai

Date

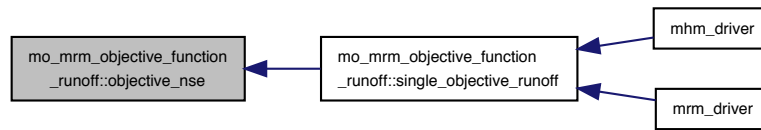
May 2013

References `eval()`, and `extract_runoff()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.35.2.17 objective_power6_nse_lnnse()

```
real(dp) function mo_mrm_objective_function_runoff::objective_power6_nse_lnnse (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of combined NSE and lnNSE with power of 5 i.e. the p-norm with p=5.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the Nash-Sutcliffe model efficiency coefficient NSE

$$NSE = 1 - \frac{\sum_{i=1}^N (Q_{obs}(i) - Q_{model}(i))^2}{\sum_{i=1}^N (Q_{obs}(i) - \bar{Q}_{obs})^2}$$

and the logarithmic Nash-Sutcliffe model efficiency coefficient $lnNSE$

$$lnNSE = 1 - \frac{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln Q_{model}(i))^2}{\sum_{i=1}^N (\ln Q_{obs}(i) - \ln \bar{Q}_{obs})^2}$$

are calculated and added up equally weighted:

$$obj_value = \sqrt[6]{(1 - NSE)^6 + (1 - lnNSE)^6}$$

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_power6_nse_lnnse — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Actually, $\sqrt[6]{(1 - NSE)^6 + (1 - lnNSE)^6}$ will be returned such that it can be minimized and converges to 0.

Author

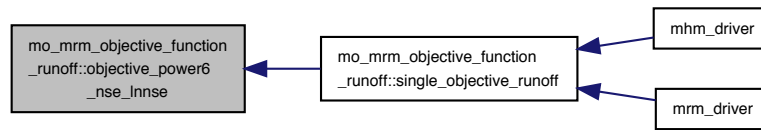
Juliane Mai and Matthias Cuntz

March 2014

Referenced by `single_objective_runoff()`.

[illegible]

Here is the caller graph for this function:



15.35.2.18 objective_sse()

```
real(dp) function mo_mrm_objective_function_runoff::objective_sse (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of SSE.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data. Therefore, the sum squared errors

$$SSE = \sum_{i=1}^N (Q_{obs}(i) - Q_{model}(i))^2$$

is calculated and the objective function is

$$obj_value = SSE$$

The observed data Q_{obs} are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_sse — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Juliane Mai and Matthias Cuntz

Date

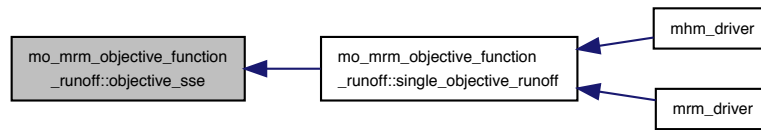
March 2014

References `eval()`, and `extract_runoff()`.

Referenced by `single_objective_runoff()`.



Here is the caller graph for this function:



15.35.2.19 parameter_regularization()

```

real(dp) function mo_mrm_objective_function_runoff::parameter_regularization (
    real(dp), dimension(:), intent(in) paraset,
    real(dp), dimension(size(paraset)), intent(in) prior,
    real(dp), dimension(size(paraset),2), intent(in) bounds,
    logical, dimension(size(paraset)), intent(in) mask )
  
```

References `mo_constants::pi_dp`.

Referenced by `loglikelihood_evin2013_2()`.

Here is the caller graph for this function:



15.35.2.20 single_objective_runoff()

```

real(dp) function, public mo_mrm_objective_function_runoff::single_objective_runoff (
    real(dp), dimension(:), intent(in) parameterset )
  
```

Wrapper for objective functions optimizing against runoff.

The function selects the objective function case defined in a namelist, i.e. the global variable `opti_function`. It returns the objective function value for a specific parameter set.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Juliane Mai

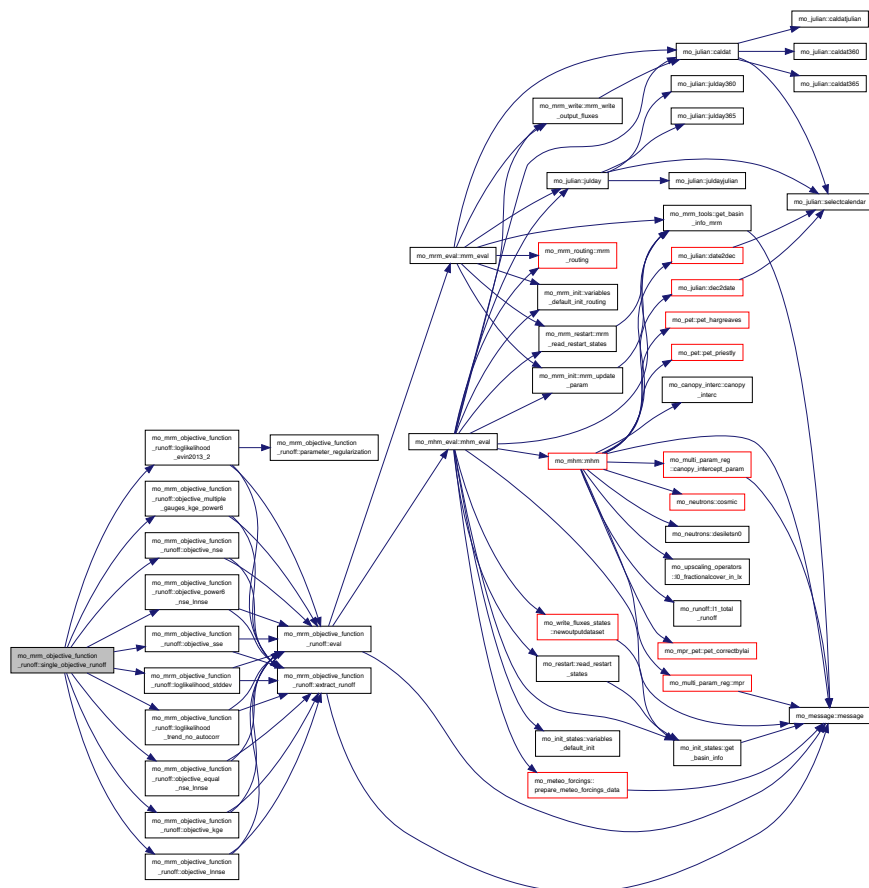
Date

Dec 2012

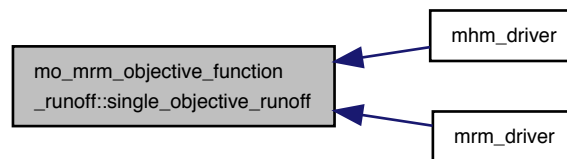
References `loglikelihood_evin2013_2()`, `loglikelihood_stddev()`, `loglikelihood_trend_no_autocorr()`, `objective_←
equal_nse_lnnse()`, `objective_kge()`, `objective_lnnse()`, `objective_multiple_gauges_kge_power6()`, `objective_nse()`,
`objective_power6_nse_lnnse()`, `objective_sse()`, and `mo_common_variables::opti_function`.

Referenced by `mhm_driver()`, and `mrm_driver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.36 mo_mrm_read_config Module Reference

read mRM config

Functions/Subroutines

- subroutine, public [read_mrm_config_coupling](#) ()
Read the coupling mode of mRM.
- subroutine, public [read_mrm_config](#) (do_message, readLatLon)
Read the general config of mRM.
- subroutine [read_mrm_routing_params](#) (processCase, file_namelist)
- logical function [in_bound](#) (params)

15.36.1 Detailed Description

read mRM config

This module contains all mRM subroutines related to reading the mRM configuration either from file or copy from mHM.

Authors

Stephan Thober

Date

Aug 2015

15.36.2 Function/Subroutine Documentation

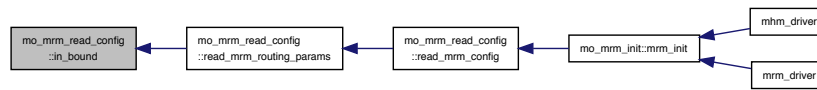
15.36.2.1 in_bound()

```

logical function mo_mrm_read_config::in_bound (
    real(dp), dimension(:, :), intent(in) params )
  
```

Referenced by `read_mrm_routing_params()`.

Here is the caller graph for this function:



15.36.2.2 read_mrm_config()

```

subroutine, public mo_mrm_read_config::read_mrm_config (
    logical, intent(in) do_message,
    logical, intent(out) readLatLon )

```

Read the general config of mRM.

Depending on the variable `mrm_coupling_config`, the mRM config is either read from `mrm.nml` and parameters from `mrm_parameter.nml` or copied from mHM.

Parameters

in	<i>logical :: do_message</i>	- flag for writing mHM standard messages
out	<i>logical :: readLatLon</i>	- flag for reading LatLon file

Author

Stephan Thober

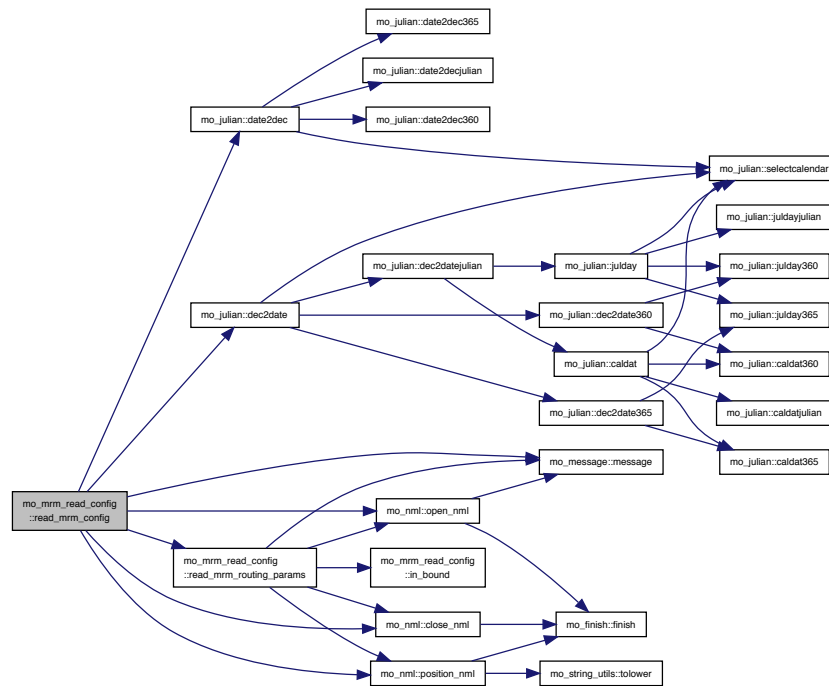
Date

Aug 2015

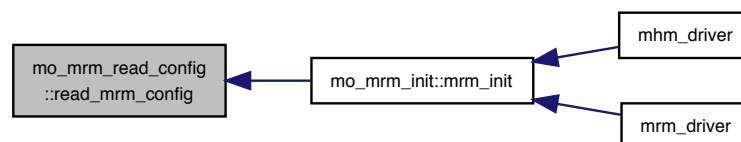
References `mo_nml::close_nml()`, `mo_julian::date2dec()`, `mo_julian::dec2date()`, `mo_mrm_file::file_defoutput`, `mo_mrm_file::file_namelist_mrm`, `mo_mrm_file::file_namelist_param_mrm`, `mo_kind::i4`, `mo_mrm_constants::maxnogauges`, `mo_message::message()`, `mo_mrm_global_variables::mrm_coupling_mode`, `mo_mrm_constants::nodata_i4`, `mo_common_variables::nprocesses`, `mo_mrm_global_variables::ntstepday`, `mo_nml::open_nml()`, `mo_common_variables::optimize`, `mo_nml::position_nml()`, `mo_common_variables::processmatrix`, `read_mrm_routing_params()`, `mo_mrm_file::udefoutput`, and `mo_mrm_file::unamelist_mrm`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.36.2.3 read_mrm_config_coupling()

```
subroutine, public mo_mrm_read_config::read_mrm_config_coupling ( )
```

Read the coupling mode of mRM.

Read the variable `mrm_coupling_mode` from `coupling_config` namelist There are three options: `mrm_coupling_mode` = 0 - Stand-alone version = 1 - Coupled to a Hydrologic or land-surface model = 2 - Coupled to mHM

The difference between the second and third option is the read of the configuration. The second option reads the configuration from `mrm.nml` and parameters from `mrm_parameters.nml`. The third option takes the configuration from `mhm.nml` and parameters from `mhm_parameters.nml`.

Note

No mrm_coupling_mode greater than 2 can be given.

Author

Stephan Thober

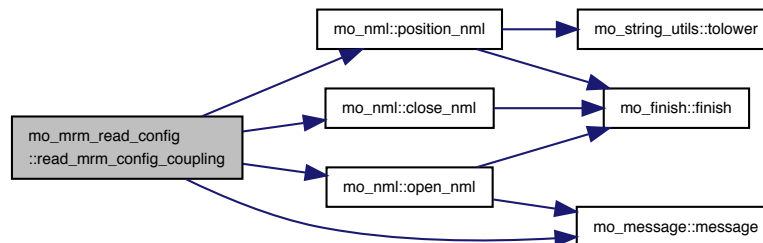
Date

Aug 2015

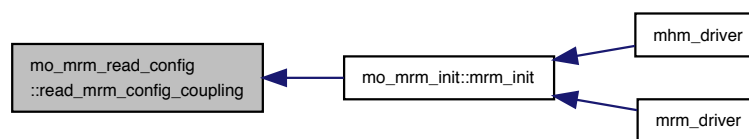
References mo_nml::close_nml(), mo_mrm_file::file_namelist_mrm, mo_message::message(), mo_mrm_global_variables::mrm_coupling_mode, mo_nml::open_nml(), mo_nml::position_nml(), and mo_mrm_file::unamelist_mrm.

Referenced by mo_mrm_init::mrm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



15.36.2.4 read_mrm_routing_params()

```

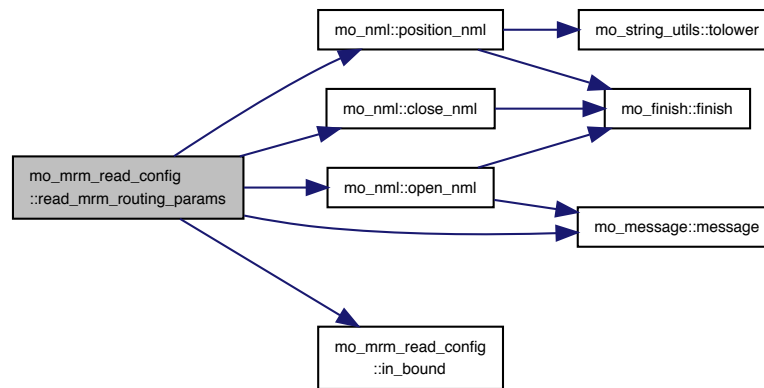
subroutine mo_mrm_read_config::read_mrm_routing_params (
    integer(i4), intent(in) processCase,
    character(256), intent(in) file_namelist )

```

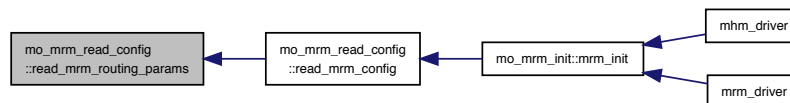
References mo_nml::close_nml(), mo_common_variables::global_parameters, in_bound(), mo_message::message(), mo_mrm_constants::ncolpars, mo_nml::open_nml(), mo_nml::position_nml(), mo_common_variables::processmatrix, and mo_mrm_file::unamelist_param.

Referenced by read_mrm_config().

Here is the call graph for this function:



Here is the caller graph for this function:



15.37 mo_mrm_read_data Module Reference

This module contains all routines to read mRM data from file.

Functions/Subroutines

- subroutine, public [mrm_read_l0_data](#) (L0_mask, L0_elev, L0_LCover)
read L0 data from file
- subroutine, public [mrm_l0_variable_init](#) (iBasin)
level 0 variable initialization
- subroutine, public [mrm_l1_variable_init](#) (iBasin)
level 1 variable initialization
- subroutine, public [mrm_read_discharge](#) ()
Read discharge timeseries from file.
- subroutine, public [mrm_read_total_runoff](#) (iBasin)
read simulated runoff that is to be routed
- subroutine [rotate_fdir_variable](#) (x)

15.37.1 Detailed Description

This module contains all routines to read mRM data from file.

Authors

Stephan Thober

Date

Aug 2015

15.37.2 Function/Subroutine Documentation

15.37.2.1 mrm_l0_variable_init()

```
subroutine, public mo_mrm_read_data::mrm_l0_variable_init (
    integer(i4), intent(in) iBasin )
```

level 0 variable initialization

following tasks are performed for L0 data sets

- cell id & numbering
- storage of cell coordinates (row and column id) If a variable is added or removed here, then it also has to be added or removed in the subroutine `config_variables_set` in module `mo_restart` and in the subroutine `set_config` in module `mo_set_netcdf_restart`

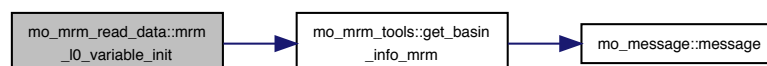
Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
in, out	<i>integer(i4), dimension(:) :: soilId_isPresent</i>	flag to indicate whether a given soil-id is present or not, DIMENSION [nSoilTypes]

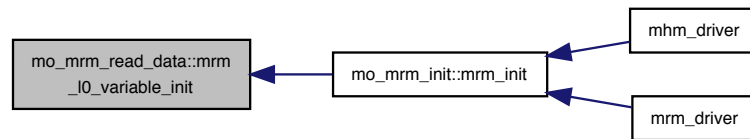
References `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_global_variables::iflag_cordinate_sys`, `mo_mrm↔_global_variables::l0_areacell`, `mo_mrm_global_variables::l0_cellcoor`, `mo_mrm_global_variables::l0_id`, `mo_↔mrm_global_variables::l0_ncells`, `mo_mrm_global_variables::level0`, `mo_mrm_constants::nodata_dp`, `mo_mrm_↔constants::nodata_i4`, `mo_constants::radiusearth_dp`, and `mo_constants::twopi_dp`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.37.2.2 mrm_l1_variable_init()

```
subroutine, public mo_mrm_read_data::mrm_l1_variable_init (
    integer(i4), intent(in) iBasin )
```

level 1 variable initialization

mRM only requires to initialize L1_areaCell and L1_mask

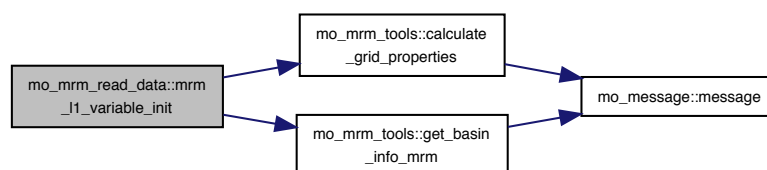
Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

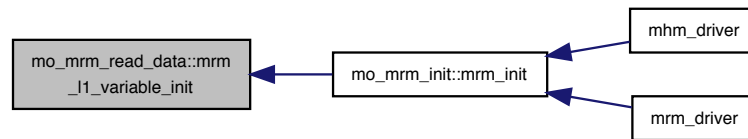
References `mo_mrm_global_variables::basin_mrm`, `mo_mrm_tools::calculate_grid_properties()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_kind::i4`, `mo_mrm_global_variables::l0_areacell`, `mo_mrm_global_variables::l1_areacell`, `mo_mrm_global_variables::l1_ncells`, `mo_mrm_global_variables::level0`, `mo_mrm_global_variables::level1`, `mo_mrm_global_variables::nbasins`, `mo_mrm_constants::nodata_dp`, and `mo_mrm_global_variables::resolutionhydrology`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.37.2.3 mrm_read_discharge()

```
subroutine, public mo_mrm_read_data::mrm_read_discharge ( )
```

Read discharge timeseries from file.

Read Observed discharge at the outlet of a catchment and at the inflow of a catchment. Allocate global runoff variable that contains the simulated runoff after the simulation.

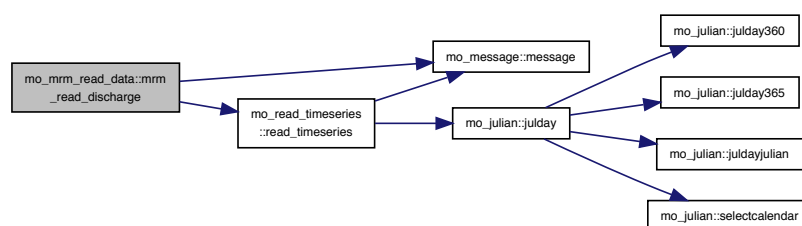
Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

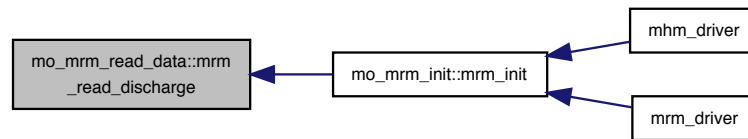
References `mo_message::message()`, `mo_mrm_global_variables::mrm_runoff`, `mo_mrm_global_variables::nbasins`, `mo_mrm_constants::nodata_dp`, `mo_common_variables::optimize`, `mo_read_timeseries::read_timeseries()`, and `mo_mrm_file::udischarge`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.37.2.4 mrm_read_l0_data()

```

subroutine, public mo_mrm_read_data::mrm_read_l0_data (
    logical, dimension(:), intent(in), optional, target L0_mask,
    real(dp), dimension(:), intent(in), optional, target L0_elev,
    integer(i4), dimension(:, :), intent(in), optional, target L0_LCover )
  
```

read L0 data from file

With the exception of `L0_mask`, `L0_elev`, and `L0_LCover`, all L0 variables are read from file. The former three are only read if they are not provided as variables.

Parameters

in	<i>logical, dimension(:), target, optional :: L0_mask - L0 mask</i>	
in	<i>real(dp), dimension(:), target, optional :: L0_elev - L0 elevation</i>	
in	<i>integer(i4), dimension(:, :), target, optional :: L0_LCover - L0 land cover</i>	None

Author

Juliane Mai, Matthias Zink, and Stephan Thober

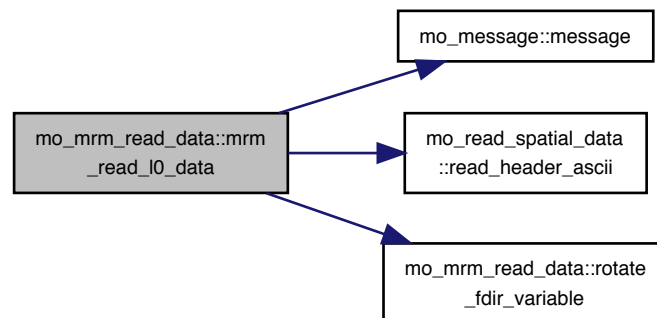
Date

Aug 2015

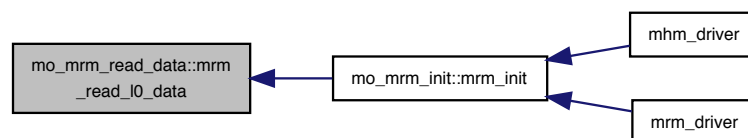
References `mo_mrm_file::file_facc`, `mo_mrm_global_variables::level0`, `mo_message::message()`, `mo_mrm_global_variables::mrm_coupling_mode`, `mo_mrm_global_variables::nbasins`, `mo_mrm_global_variables::nlcoverscene`, `mo_mrm_constants::nodata_dp`, `mo_mrm_constants::nodata_i4`, `mo_mrm_global_variables::perform_mpr`, `mo_common_variables::processmatrix`, `mo_read_spatial_data::read_header_ascii()`, `rotate_fdir_variable()`, and `mo_mrm_file::ufacc`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.37.2.5 mrm_read_total_runoff()

```

subroutine, public mo_mrm_read_data::mrm_read_total_runoff (
    integer(i4), intent(in) iBasin )

```

read simulated runoff that is to be routed

read spatio-temporal field of total runoff that has been simulated by a hydrologic model or land surface model. This total runoff will then be aggregated to the level 11 resolution and then routed through the stream network.

Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

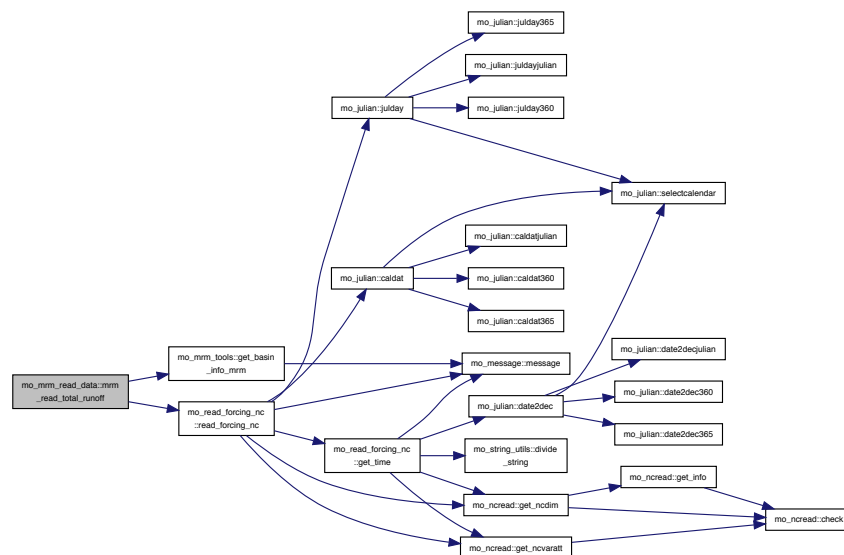
Note

The file containing total runoff must be named `total_runoff.nc`. This file must contain a double precision float variable with the name "total_runoff". There must also be an integer variable time with the units hours, days, months or years.

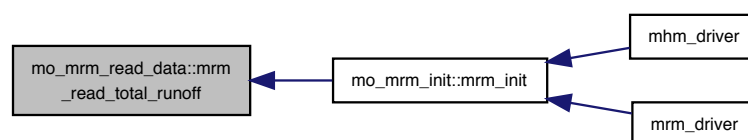
References `mo_common_variables::alma_convention`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_constants::hoursecs`, `mo_mrm_constants::nodata_dp`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_mrm_global_variables::simper`, and `mo_mrm_global_variables::timestep`.

Referenced by `mo_mrm_init::mrm_init()`.

Here is the call graph for this function:



Here is the caller graph for this function:



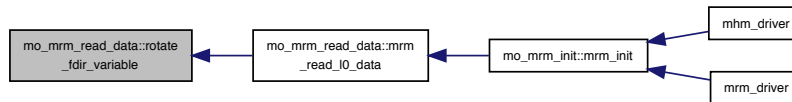
15.37.2.6 rotate_fdir_variable()

```
subroutine mo_mrm_read_data::rotate_fdir_variable (
    integer(i4), dimension(:, :), intent(inout) x )
```

References mo_mrm_constants::nodata_i4.

Referenced by mrm_read_l0_data().

Here is the caller graph for this function:



15.38 mo_mrm_read_latlon Module Reference

reading latitude and longitude coordinates for each basin

Functions/Subroutines

- subroutine, public [read_latlon](#) (ii)
reads latitude and longitude coordinates

15.38.1 Detailed Description

reading latitude and longitude coordinates for each basin

Authors

Stephan Thober

Date

Nov 2013

15.38.2 Function/Subroutine Documentation

15.38.2.1 read_latlon()

```
subroutine, public mo_mrm_read_latlon::read_latlon (
    integer(i4), intent(in) ii )
```

reads latitude and longitude coordinates

reads latitude and longitude coordinates from netcdf file for each basin and appends it to the global variables latitude and longitude.

Parameters

in	<i>integer(i4) :: ii</i>	basin index File name of the basins must be xxx_latlon.nc, where xxx is the basin id. Variable names in the netcdf file have to be 'lat' for latitude and 'lon' for longitude.
----	--------------------------	--

Author

Stephan Thober

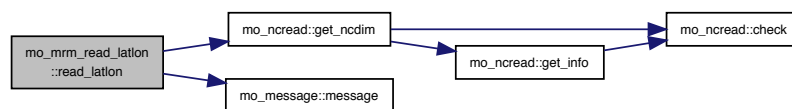
Date

Nov 2013

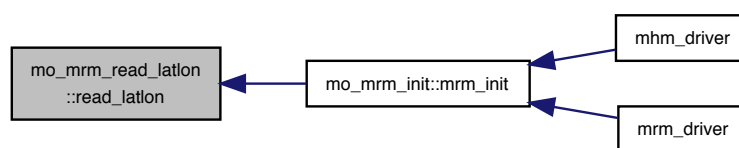
References mo_mrm_global_variables::basin_mrm, mo_mrm_global_variables::filelatlon, mo_ncread::get_ncdim(), mo_mrm_global_variables::l0_basin, mo_mrm_global_variables::l0_latitude, mo_mrm_global_variables::l0_longitude, mo_mrm_global_variables::l11_rect_latitude, mo_mrm_global_variables::l11_rect_longitude, mo_mrm_global_variables::level0, mo_mrm_global_variables::level11, and mo_message::message().

Referenced by mo_mrm_init::mrm_init().

Here is the call graph for this function:



Here is the caller graph for this function:



15.39 mo_mrm_restart Module Reference

Restart routines.

Functions/Subroutines

- subroutine, public [mrm_write_restart](#) (iBasin, OutPath)
write routing states and configuration
- subroutine, public [mrm_read_restart_states](#) (iBasin, InPath)

read routing states

- subroutine, public [mrm_read_restart_config](#) (iBasin, InPath)

reads Level 11 configuration from a restart directory

15.39.1 Detailed Description

Restart routines.

This module contains the subroutines for reading and writing routing related variables to file.

Authors

Stephan Thober

Date

Aug 2015

15.39.2 Function/Subroutine Documentation

15.39.2.1 mrm_read_restart_config()

```
subroutine, public mo_mrm_restart::mrm_read_restart_config (
    integer(i4), intent(in) iBasin,
    character(256), intent(in) InPath )
```

reads Level 11 configuration from a restart directory

read Level 11 configuration variables from a given restart directory and initializes all Level 11 configuration variables, that are initialized in L11_variable_init, contained in module [mo_startup](#).

Parameters

in	<i>integer(i4) :: iBasin</i>	number of basin
in	<i>character(256) :: InPath</i>	Input Path including trailing slash

Note

Restart Files must have the format, as if it would have been written by subroutine write_restart_files

Author

Stephan Thober

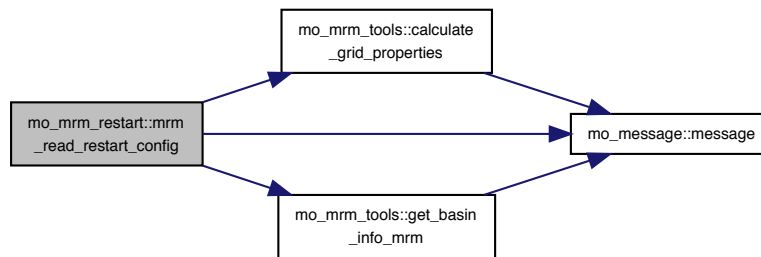
Date

Apr 2013

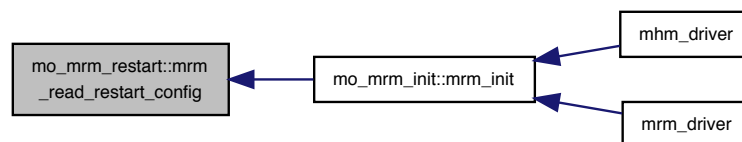
References [mo_mrm_tools::calculate_grid_properties\(\)](#), [mo_kind::dp](#), [mo_mrm_tools::get_basin_info_mrm\(\)](#), [mo_kind::i4](#), [mo_mrm_global_variables::l0_basin](#), [mo_message::message\(\)](#), [mo_mrm_constants::nodata_dp](#), [mo_mrm_constants::nodata_i4](#), and [mo_common_variables::processmatrix](#).

Referenced by [mo_mrm_init::mrm_init\(\)](#).

Here is the call graph for this function:



Here is the caller graph for this function:



15.39.2.2 mrm_read_restart_states()

```

subroutine, public mo_mrm_restart::mrm_read_restart_states (
    integer(i4), intent(in) iBasin,
    character(256), intent(in) InPath )

```

read routing states

This subroutine reads the routing states from `mRM_states_<basin_id>.nc` that has to be in the given path directory. This subroutine has to be called directly each time the `mHM_eval` or `mRM_eval` is called such that the the states are always the same at the first simulation time step, crucial for optimization.

Parameters

in	<i>integer(i4) :: iBasin</i>	number of basin
in	<i>character(256) :: InPath</i>	Input Path including trailing slash

Note

This subroutine has to be called directly each time the `mHM_eval` or `mRM_eval` is called such that the the states are always the same at the first simulation time step, crucial for optimization.

Author

Stephan Thober

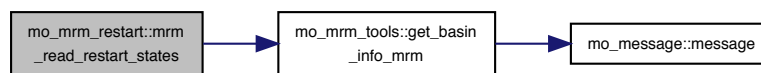
Date

Sep 2015

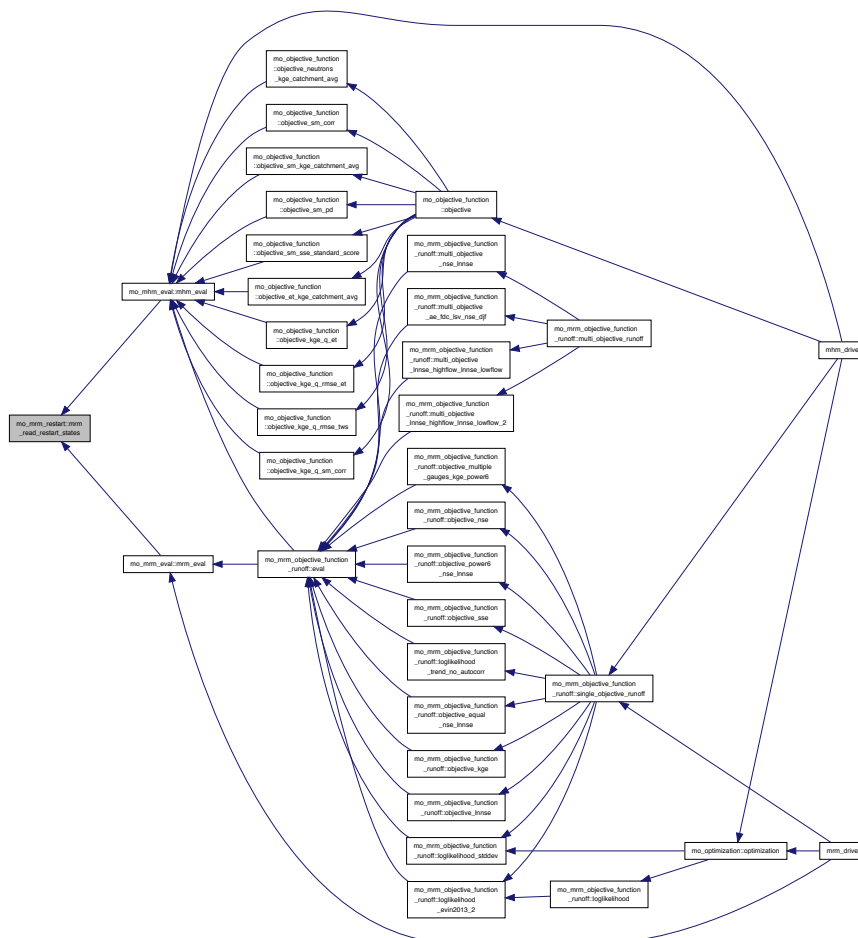
References `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_global_variables::l11_c1`, `mo_mrm_global_variables::l11_c2`, `mo_mrm_global_variables::l11_fracfpimp`, `mo_mrm_global_variables::l11_k`, `mo_mrm_global_variables::l11_qmod`, `mo_mrm_global_variables::l11_qout`, `mo_mrm_global_variables::l11_qtin`, `mo_mrm_global_variables::l11_qtr`, `mo_mrm_global_variables::l11_xi`, and `mo_mrm_constants::nroutingstates`.

Referenced by `mo_mhm_eval::mhm_eval()`, and `mo_mrm_eval::mrm_eval()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.39.2.3 mrm_write_restart()

```
subroutine, public mo_mrm_restart::mrm_write_restart (
    integer(i4), intent(in) iBasin,
    character(256), dimension(:), intent(in) OutPath )
```

write routing states and configuration

write configuration and state variables to a given restart directory.

Parameters

in	<i>integer(i4) :: iBasin</i>	number of basin
in	<i>character(256) :: InPath</i>	Input Path including trailing slash

Note

can only be used after mHM write_restart has been called because state variables are added to the file containing the state variables of mHM. This file must exist.

Author

Stephan Thober

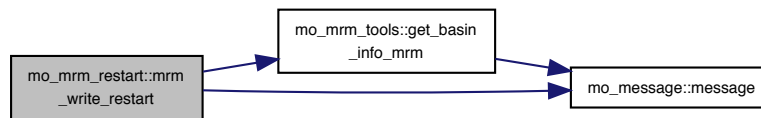
Date

Aug 2015

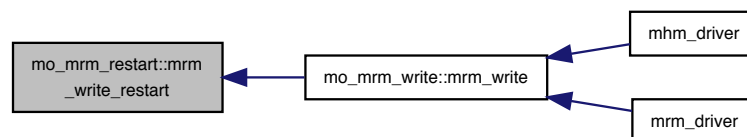
References mo_mrm_global_variables::basin_mrm, mo_mrm_tools::get_basin_info_mrm(), mo_mrm_global_variables::l0_areacell, mo_mrm_global_variables::l0_cellcoor, mo_mrm_global_variables::l0_dracell, mo_mrm_global_variables::l0_drasc, mo_mrm_global_variables::l0_floodplain, mo_mrm_global_variables::l0_id, mo_mrm_global_variables::l0_l1_id, mo_mrm_global_variables::l0_streamnet, mo_mrm_global_variables::l1_afloodplain, mo_mrm_global_variables::l1_areacell, mo_mrm_global_variables::l1_c1, mo_mrm_global_variables::l1_c2, mo_mrm_global_variables::l1_cellcoor, mo_mrm_global_variables::l1_colout, mo_mrm_global_variables::l1_downbound_l0, mo_mrm_global_variables::l1_downbound_l1, mo_mrm_global_variables::l1_fcol, mo_mrm_global_variables::l1_fdir, mo_mrm_global_variables::l1_fracfpimp, mo_mrm_global_variables::l1_fromn, mo_mrm_global_variables::l1_frow, mo_mrm_global_variables::l1_id, mo_mrm_global_variables::l1_k, mo_mrm_global_variables::l1_l1_id, mo_mrm_global_variables::l1_label, mo_mrm_global_variables::l1_leftbound_l0, mo_mrm_global_variables::l1_leftbound_l1, mo_mrm_global_variables::l1_length, mo_mrm_global_variables::l1_netperm, mo_mrm_global_variables::l1_qmod, mo_mrm_global_variables::l1_qout, mo_mrm_global_variables::l1_qtin, mo_mrm_global_variables::l1_qtr, mo_mrm_global_variables::l1_rightbound_l0, mo_mrm_global_variables::l1_rightbound_l1, mo_mrm_global_variables::l1_rorder, mo_mrm_global_variables::l1_rowout, mo_mrm_global_variables::l1_sink, mo_mrm_global_variables::l1_slope, mo_mrm_global_variables::l1_tcol, mo_mrm_global_variables::l1_ton, mo_mrm_global_variables::l1_trow, mo_mrm_global_variables::l1_tsout, mo_mrm_global_variables::l1_upbound_l0, mo_mrm_global_variables::l1_upbound_l1, mo_mrm_global_variables::l1_xi, mo_mrm_global_variables::l1_areacell, mo_mrm_global_variables::l1_id, mo_mrm_global_variables::l1_l1_id, mo_message::message(), mo_mrm_constants::nodata_dp, mo_mrm_constants::nodata_i4, mo_mrm_constants::nroutingstates, and mo_common_variables::processmatrix.

Referenced by mo_mrm_write::mrm_write().

Here is the call graph for this function:



Here is the caller graph for this function:



15.40 mo_mrm_routing Module Reference

Performs runoff routing for mHM at level L11.

Functions/Subroutines

- subroutine, public [mrm_routing](#) (processCase, global_routing_param, L1_total_runoff, L1_areaCell, L1_L11_Id, L11_areaCell, L11_L1_Id, L11_netPerm, L11_fromN, L11_toN, L11_nOutlets, timestep, ts_RoutFactor, nNodes, nInflowGauges, InflowGaugeIndexList, InflowGaugeHeadwater, InflowGaugeNodeList, InflowDischarge, nGauges, gaugeIndexList, gaugeNodeList, map_flag, L0_LCover, L0_floodPlain, L0_areaCell, L11_aFloodPlain, L11_length, L11_slope, L11_C1, L11_C2, L11_qOut, L11_qTIN, L11_qTR, L11_qMod, GaugeDischarge, L11_FracFPimp, do_mpr_routing)
route water given runoff
- subroutine [l11_runoff_acc](#) (qAll, efecArea, L1_L11_Id, L11_areaCell, L11_L1_Id, TS, map_flag, qAcc)
total runoff accumulation at L11.
- subroutine [add_inflow](#) (nInflowGauges, InflowIndexList, InflowHeadwater, InflowNodeList, QInflow, qOut)
Adds inflow discharge to the runoff produced at the cell where the inflow is occurring.
- subroutine [l11_routing](#) (nNodes, nLinks, netPerm, netLink_fromN, netLink_toN, netLink_C1, netLink_C2, netNode_qOUT, nInflowGauges, InflowHeadwater, InflowNodeList, netNode_qTIN, netNode_qTR, netNode_qMod)
Performs runoff routing for mHM at L11 upscaled network ([Routing Network](#)).

15.40.1 Detailed Description

Performs runoff routing for mHM at level L11.

This module performs flood routing at a given time step through the stream network at level L11 to the sink cell. The Muskingum flood routing algorithm is used.

Author

Luis Samaniego

Date

Dec 2012

15.40.2 Function/Subroutine Documentation**15.40.2.1 add_inflow()**

```

subroutine mo_mrm_routing::add_inflow (
    integer(i4), intent(in) nInflowGauges,
    integer(i4), dimension(:), intent(in) InflowIndexList,
    logical, dimension(:), intent(in) InflowHeadwater,
    integer(i4), dimension(:), intent(in) InflowNodeList,
    real(dp), dimension(:), intent(in) QInflow,
    real(dp), dimension(:), intent(inout) qOut )

```

Adds inflow discharge to the runoff produced at the cell where the inflow is occurring.

If a inflow gauge is given, then this routine is adding the values to the runoff produced at the grid cell where the inflow is happening. The values are not directly added to the river network. If this cell is not a headwater then the streamflow produced upstream will be neglected.

Parameters

in	<i>integer(i4) :: nInflowGauges</i>	number of inflow gauges
in	<i>integer(i4) :: InflowIndexList</i>	index of inflow points
in	<i>logical :: InflowHeadwater</i>	flag to consider headwater cells of inflow gauge
in	<i>integer(i4) :: InflowNodeList</i>	L11 ID of inflow points
in	<i>real(dp) :: QInflow</i>	[m3 s-1] inflowing water
in, out	<i>real(dp) :: qOut</i>	[m3 s-1] Series of attenuated runoff

Author

Stephan Thober & Matthias Zink

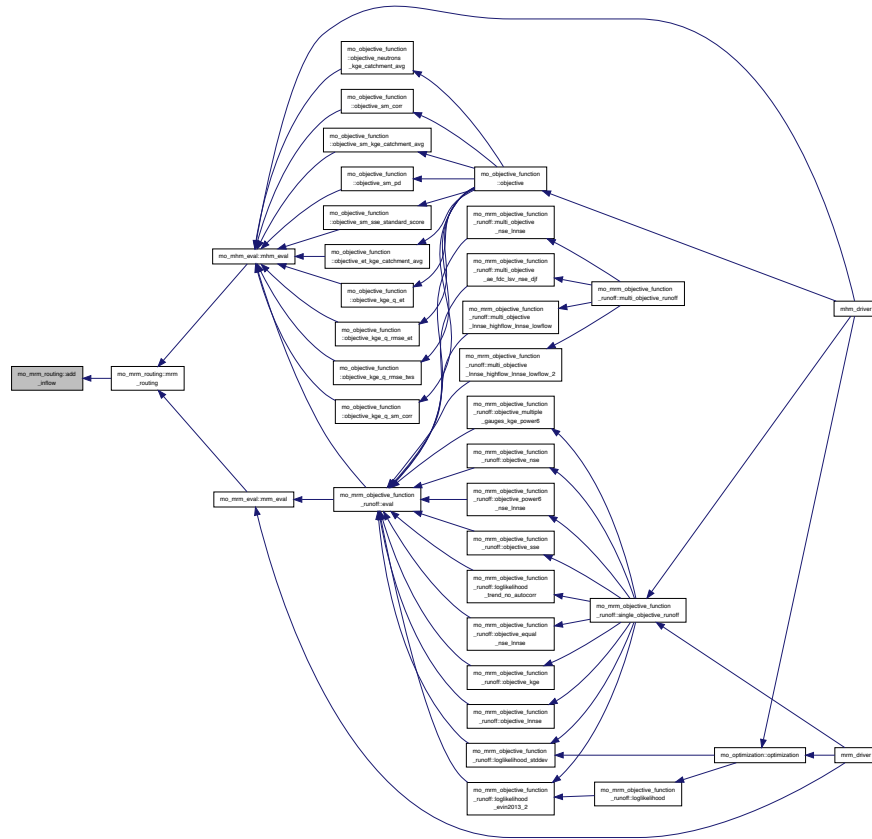
Date

Jul 2016

References mo_kind::dp, and mo_kind::i4.

Referenced by mrm_routing().

Here is the caller graph for this function:



15.40.2.2 l11_routing()

```

subroutine mo_mrm_routing::l11_routing (
  integer(i4), intent(in) nNodes,
  integer(i4), intent(in) nLinks,
  integer(i4), dimension(:), intent(in) netPerm,
  integer(i4), dimension(:), intent(in) netLink_fromN,
  integer(i4), dimension(:), intent(in) netLink_toN,
  real(dp), dimension(:), intent(in) netLink_C1,
  real(dp), dimension(:), intent(in) netLink_C2,
  real(dp), dimension(:), intent(in) netNode_qOUT,
  integer(i4), intent(in) nInflowGauges,
  logical, dimension(:), intent(in) InflowHeadwater,
  integer(i4), dimension(:), intent(in) InflowNodeList,
  real(dp), dimension(:, :), intent(inout) netNode_qTIN,
  real(dp), dimension(:, :), intent(inout) netNode_qTR,
  real(dp), dimension(nnodes), intent(out) netNode_Qmod )

```

Performs runoff routing for mHM at L11 upscaled network ([Routing Network](#)).

Hydrograph routing is carried out with the Muskingum algorithm [7]. This simplification of the St. Venant equations is justified in mHM because the potential areas of application of this model would hardly exhibit abruptly changing hydrographs with supercritical flows. The discharge leaving the river reach located on cell i $Q_i^1(t)$ at time step t can be determined by

$$Q_i^1(t) = Q_i^1(t-1) + c_1 (Q_i^0(t-1) - Q_i^1(t-1)) + c_2 (Q_i^0(t) - Q_i^0(t-1))$$

with

$$Q_i^0(t) = Q_{i'}(t) + Q_{i'}^1(t)$$

$$c_1 = \frac{\Delta t}{\kappa(1-\xi) + \frac{\Delta t}{2}}$$

$$c_2 = \frac{\frac{\Delta t}{2} - \kappa\xi}{\kappa(1-\xi) + \frac{\Delta t}{2}}$$

where

Q_i^0 and Q_i^1 denote the discharge entering and leaving the river reach located on cell i respectively.

$Q_{i'}$ is the contribution from the upstream cell i' .

κ Muskingum travel time parameter.

ξ Muskingum attenuation parameter.

Δt time interval in hours.

t Time index for each Δt interval.

To improve performance, a routing sequence "netPerm" is required. This permutation is determined in the mo_↔ init_mrm routine.

Parameters

in	<i>integer(i4) :: nNodes</i>	number of network nodes = nCells1
in	<i>integer(i4) :: nLinks</i>	number of stream segment (reaches)
in	<i>integer(i4) :: netPerm</i>	routing order of a given basin (permutation)
in	<i>integer(i4) :: netLink_fromN</i>	from node
in	<i>integer(i4) :: netLink_toN</i>	to node
in	<i>real(dp) :: netLink_C1</i>	routing parameter C1 ([7] p. 25-41)
in	<i>real(dp) :: netLink_C2</i>	routing parameters C2 (id)
in	<i>real(dp) :: netNode_qOUT</i>	Total outflow from cells (given basin) L11 at time tt in [m3 s-1]
in, out	<i>real(dp) :: netNode_qTIN(nNodes,2)</i>	Total discharge inputs at t-1 and t
in, out	<i>real(dp) :: netNode_qTR(nNodes,2)</i>	Routed outflow leaving a node ([15])
out	<i>real(dp) :: netNode_Qmod(nNodes)</i>	Simulated discharge [m3 s-1]

Note

A basin outlet should be identified.

Author

Luis Samaniego

Date

Dec 2005

Referenced by mrm_routing().

Parameters

in	<i>integer(i4) :: TS</i>	time step in [s]
in	<i>logical :: map_flag</i>	Flag indicating whether routing resolution is higher than hydrologic one
out	<i>real(dp) :: qAcc</i>	aggregated runoff at L11 [m3 s-1]

Author

Luis Samaniego

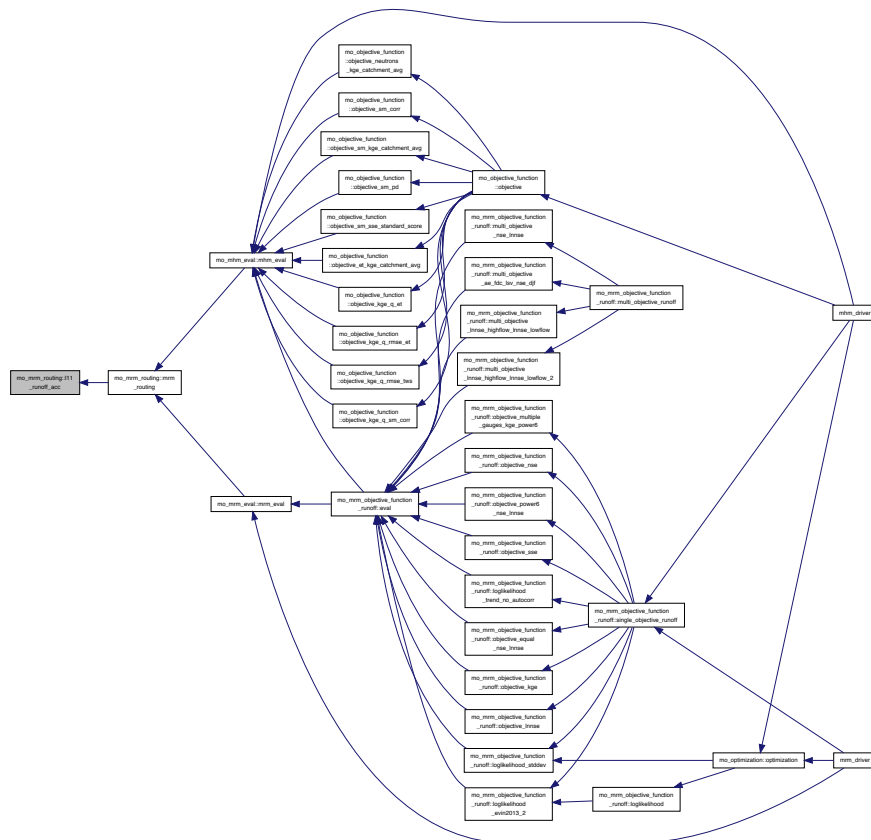
Date

Jan 2013

References mo_mrm_constants::hoursecs, and mo_mrm_constants::nodata_dp.

Referenced by mrm_routing().

Here is the caller graph for this function:



15.40.2.4 mrm_routing()

```

subroutine, public mo_mrm_routing::mrm_routing (
  integer(i4), intent(in) processCase,

```

```

real(dp), dimension(:), intent(in) global_routing_param,
real(dp), dimension(:), intent(in) L1_total_runoff,
real(dp), dimension(:), intent(in) L1_areaCell,
integer(i4), dimension(:), intent(in) L1_L11_Id,
real(dp), dimension(:), intent(in) L11_areaCell,
integer(i4), dimension(:), intent(in) L11_L1_Id,
integer(i4), dimension(:), intent(in) L11_netPerm,
integer(i4), dimension(:), intent(in) L11_fromN,
integer(i4), dimension(:), intent(in) L11_toN,
integer(i4), intent(in) L11_nOutlets,
integer(i4), intent(in) timestep,
real(dp), intent(in) tsRoutFactor,
integer(i4), intent(in) nNodes,
integer(i4), intent(in) nInflowGauges,
integer(i4), dimension(:), intent(in) InflowGaugeIndexList,
logical, dimension(:), intent(in) InflowGaugeHeadwater,
integer(i4), dimension(:), intent(in) InflowGaugeNodeList,
real(dp), dimension(:), intent(in) InflowDischarge,
integer(i4), intent(in) nGauges,
integer(i4), dimension(:), intent(in) gaugeIndexList,
integer(i4), dimension(:), intent(in) gaugeNodeList,
logical, intent(in) map_flag,
integer(i4), dimension(:), intent(in) L0_LCover,
integer(i4), dimension(:), intent(in) L0_floodPlain,
real(dp), dimension(:), intent(in) L0_areaCell,
real(dp), dimension(:), intent(in) L11_aFloodPlain,
real(dp), dimension(:), intent(in) L11_length,
real(dp), dimension(:), intent(in) L11_slope,
real(dp), dimension(:), intent(inout) L11_C1,
real(dp), dimension(:), intent(inout) L11_C2,
real(dp), dimension(:), intent(inout) L11_qOut,
real(dp), dimension(:, :), intent(inout) L11_qTIN,
real(dp), dimension(:, :), intent(inout) L11_qTR,
real(dp), dimension(:), intent(inout) L11_qMod,
real(dp), dimension(:), intent(inout) GaugeDischarge,
real(dp), dimension(:), intent(inout) L11_FracFPimp,
logical, intent(in), optional do_mpr_routing )

```

route water given runoff

This routine first performs mpr for the routing variables if required, then accumulates the runoff to the routing resolution and eventually routes the water in a third step. The last step is repeated multiple times if the routing timestep is smaller than the timestep of the hydrological timestep

Parameters

in	<i>integer(i4) :: processCase</i>	Process switch for routing
in	<i>real(dp), dimension(5) :: global_routing_param</i>	routing parameters
in	<i>real(dp), dimension(:) :: L1_total_runoff</i>	total runoff from L1 grid cells
in	<i>real(dp), dimension(:) :: L1_areaCell</i>	L1 cell area
in	<i>integer(i4), dimension(:) :: L1_L11_Id</i>	L1 cell ids on L11
in	<i>real(dp), dimension(:) :: L11_areaCell</i>	L11 cell area
in	<i>integer(i4), dimension(:) :: L11_L1_Id</i>	L11 cell ids on L1
in	<i>integer(i4), dimension(:) :: L11_netPerm</i>	L11 routing order
in	<i>integer(i4), dimension(:) :: L11_fromN</i>	L11 source grid cell order
in	<i>integer(i4), dimension(:) :: L11_toN</i>	L11 target grid cell order
in	<i>integer(i4) :: timestep</i>	simulation timestep in [h]

Parameters

in	<i>real(dp) :: tsRoutFactor</i>	factor between routing timestep and hydrological timestep
in	<i>integer(i4) :: nNodes</i>	number of nodes
in	<i>integer(i4) :: nInflowGauges</i>	number of inflow gauges
in	<i>integer(i4), dimension(:) :: InflowGaugeIndexList</i>	index list of inflow gauges
in	<i>logical, dimension(:) :: InflowGaugeHeadwater</i>	flag for headwater cell of inflow gauge
in	<i>integer(i4), dimension(:) :: InflowGaugeNodeList</i>	gauge node list at L11
in	<i>real(dp), dimension(:) :: InflowDischarge</i>	inflowing discharge at discharge gauge at current day
in	<i>integer(i4) :: nGauges</i>	number of recording gauges
in	<i>integer(i4), dimension(:) :: gaugeIndexList</i>	index list for outflow gauges
in	<i>integer(i4), dimension(:) :: gaugeNodeList</i>	gauge node list at L11
in	<i>logical :: map_flag</i>	flag indicating whether routing resolution is coarser than hydrologic resolution
in	<i>integer(i4), dimension(:) :: L0_LCover</i>	L0 land cover
in	<i>integer(i4), dimension(:) :: L0_floodPlain</i>	L0 fraction of flood plains
in	<i>real(dp), dimension(:) :: L0_areaCell</i>	L0 cell area
in	<i>real(dp), dimension(:) :: L11_aFloodPlain</i>	L11 area of flood plain
in	<i>real(dp), dimension(:) :: L11_length</i>	L11 link length
in	<i>real(dp), dimension(:) :: L11_slope</i>	L11 slope
in, out	<i>real(dp), dimension(:) :: L11_C1</i>	L11 muskingum parameter 1
in, out	<i>real(dp), dimension(:) :: L11_C2</i>	L11 muskingum parameter 2
in, out	<i>real(dp), dimension(:) :: L11_qOut</i>	total runoff from L11 grid cells
in, out	<i>real(dp), dimension(:, :) :: L11_qTIN</i>	L11 inflow to the reach
in, out	<i>real(dp), dimension(:, :) :: L11_qTR</i>	L11 routed outflow
in, out	<i>real(dp), dimension(:) :: L11_qMod</i>	modelled discharge at each grid cell
in, out	<i>real(dp), dimension(:) :: GaugeDischarge</i>	modelled discharge at each gauge
in, out	<i>real(dp), dimension(:) :: L11_FracFPimp</i>	L11 fraction of flood plain with impervious cover
in	<i>logical, optional :: do_mpr_routing</i>	indicate whether routing is to be performed"

Author

Stephan Thober

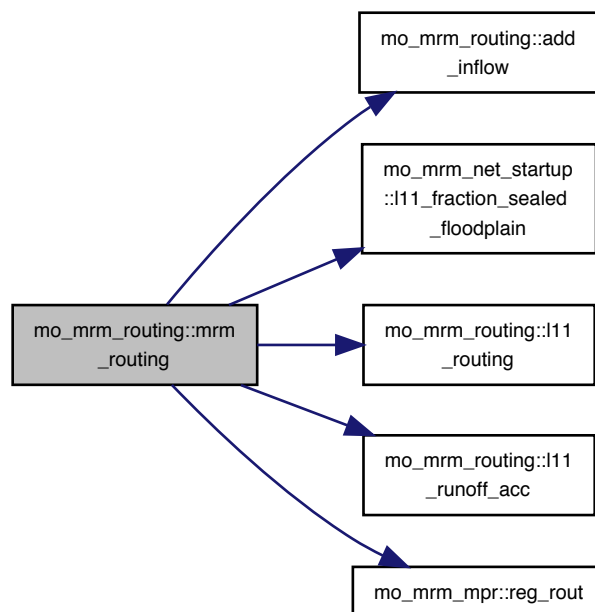
Date

Aug 2015

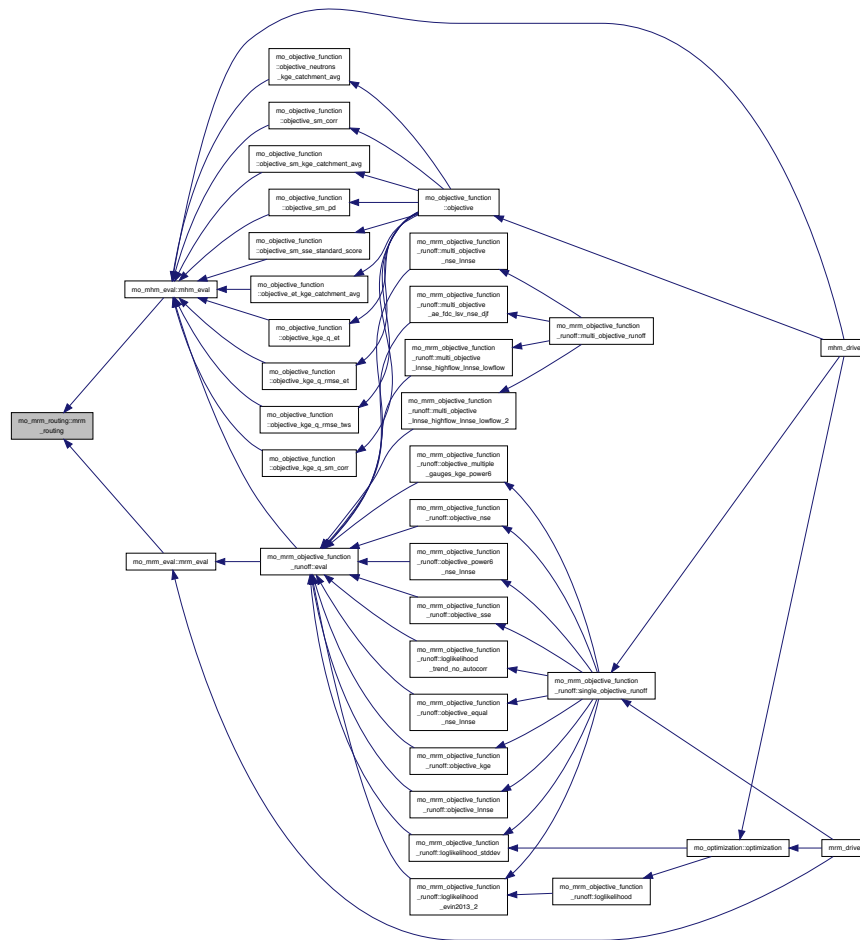
References `add_inflow()`, `mo_mrm_global_variables::is_start`, `mo_mrm_net_startup::l11_fraction_sealed_floodplain()`, `l11_routing()`, `l11_runoff_acc()`, and `mo_mrm_mpr::reg_rout()`.

Referenced by `mo_mhm_eval::mhm_eval()`, and `mo_mrm_eval::mrm_eval()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.41 mo_mrm_signatures Module Reference

Module with calculations for several hydrological signatures.

Functions/Subroutines

- real(dp) function, dimension(size(lags, 1)), public [autocorrelation](#) (data, lags, mask)
Autocorrelation of a given data series.
- real(dp) function, dimension(size(quantiles, 1)), public [flowdurationcurve](#) (data, quantiles, mask, concavity, _index, mid_segment_slope, mhigh_segment_volume, high_segment_volume, low_segment_volume)
Flow duration curves.
- subroutine, public [limb_densities](#) (data, mask, RLD, DLD)
Calculates limb densities.
- real(dp) function [maximummonthlyflow](#) (data, mask, yr_start, mo_start, dy_start)
Maximum of average flows per months.
- subroutine, public [moments](#) (data, mask, mean_data, stddev_data, median_data, max_data, mean_log, stddev_log, median_log, max_log)
Moments of data and log-transformed data, e.g. mean and standard deviation.

- `real(dp)` function, `dimension(size(quantiles, 1))`, public [peakdistribution](#) (`data`, `quantiles`, `mask`, `slope_peak`↔
_distribution)
Calculates the peak distribution.
- `real(dp)` function, public [runoffratio](#) (`data`, `basin_area`, `mask`, `precip_series`, `precip_sum`, `log_data`)
Runoff ratio (accumulated daily discharge [mm/d] / accumulated daily precipitation [mm/d]).
- `real(dp)` function, public [zeroflowratio](#) (`data`, `mask`)
Ratio of zero values to total number of data points.

15.41.1 Detailed Description

Module with calculations for several hydrological signatures.

This module contains calculations for hydrological signatures. It contains:

- Autocorrelation
- Rising and declining limb densities
- Flow duration curves
- Peak distribution

Authors

Remko Nijzink,

Date

March 2014

15.41.2 Function/Subroutine Documentation

15.41.2.1 autocorrelation()

```
real(dp) function, dimension(size(lags,1)), public mo_mrm_signatures::autocorrelation (
    real(dp), dimension(:), intent(in) data,
    integer(i4), dimension(:), intent(in) lags,
    logical, dimension(size(data,1)), intent(in), optional mask )
```

Autocorrelation of a given data series.

Calculates the autocorrelation of a data series at given lags. An optional argument for masking data points can be given. The function is basically a wrapper of the function `autocorr` from the module [mo_corr](#).

An optional mask of data points can be specified.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	Array of data
in	<i>integer(i4), dimension(:) :: lags</i>	Array of lags where autocorrelation is requested
in	<i>logical, dimension(size(data,1)) :: mask</i>	Mask for data points given Works only with 1d double precision input data.

Author

Juliane Mai

Date

Jun 2015

15.41.2.2 flowdurationcurve()

```

real(dp) function, dimension(size(quantiles,1)), public mo_mrm_signatures::flowdurationcurve (
    real(dp), dimension(:), intent(in) data,
    real(dp), dimension(:), intent(in) quantiles,
    logical, dimension(:), intent(in), optional mask,
    real(dp), intent(out), optional concavity_index,
    real(dp), intent(out), optional mid_segment_slope,
    real(dp), intent(out), optional mhigh_segment_volume,
    real(dp), intent(out), optional high_segment_volume,
    real(dp), intent(out), optional low_segment_volume )

```

Flow duration curves.

Calculates the flow duration curves for a given data vector. The Flow duration curve at a certain quantile x is the data point p where $x\%$ of the data points are above the value p .

Hence the function percentile of the module [mo_percentile](#) is used. But percentile is determining the point p where $x\%$ of the data points are below that value. Therefore, the given quantiles are transformed by $(1.0 - \text{quantile})$ to get the percentiles of exceedance probabilities.

Optionally, the concavity index CI can be calculated [Zhang2014]. CI is defined by

$$CI = \frac{q_{10\%} - q_{99\%}}{q_{1\%} - q_{99\%}}$$

where q_x is the data point where $x\%$ of the data points are above that value. Hence, exceedance probabilities are used.

Optionally, the FDC mid-segment slope FDC_{MSS} as used by Shafii et. al (2014) can be returned. The FDC_{MSS} is defined as

$$FDC_{MSS} = \log(q_{m_1}) - \log(q_{m_2})$$

where m_1 and m_2 are the lowest and highest flow exceedance probabilities within the midsegment of FDC. The settings $m_1 = 0.2$ and 0.7 are used by Shafii et. al (2014) and are implemented like that.

Optionally, the FDC medium high-segment volume FDC_{MHSV} as used by Shafii et. al (2014) can be returned. The FDC_{MHSV} is defined as

$$FDC_{MHSV} = \sum_{h=1}^H q_h$$

where $h = 1, 2, \dots, H$ are flow indices located within the high-flow segment (exceedance probabilities lower than m_1). H is the index of the maximum flow. The settings $m_1 = 0.2$ is used here to be consistent with the definitions of the low-segment (0.7-1.0) and the mid-segment (0.2-0.7).

Optionally, the FDC high-segment volume FDC_{HSV} as used by Shafii et. al (2014) can be returned. The FDC_{HSV} is defined as

$$FDC_{HSV} = \sum_{h=1}^H q_h$$

where $h = 1, 2, \dots, H$ are flow indices located within the high-flow segment (exceedance probabilities lower than m_1). H is the index of the maximum flow. The settings $m_1 = 0.02$ is used by Shafii et. al (2014) and is implemented like that.

Optionally, the FDC low-segment volume FDC_{LSV} as used by Shafii et. al (2014) can be returned. The FDC_{LSV} is defined as

$$FDC_{LSV} = - \sum_{l=1}^L (\log(q_l) - \log(q_L))$$

where $l = 1, 2, \dots, L$ are flow indices located within the low-flow segment (exceedance probabilities larger than m_1). L is the index of the minimum flow. The settings $m_1 = 0.7$ is used by Shafii et. al (2014) and is implemented like that.

An optional mask of data points can be specified.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	data series
in	<i>real(dp), dimension(:) :: Quantiles</i>	Percentages of exceedance
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask of data array
out	<i>real(dp), optional :: concavity_index</i>	concavity index as defined by Sauquet et al. (2011)
out	<i>real(dp), optional :: mid_segment_slope</i>	mid-segment slope as defined by Shafii et al. (2014)
out	<i>real(dp), optional :: mhigh_segment_volume</i>	medium high-segment volume
out	<i>real(dp), optional :: high_segment_volume</i>	high-segment volume as defined by Shafii et al. (2014)
out	<i>real(dp), optional :: low_segment_volume</i>	low-segment volume as defined by Shafii et al. (2014)

Returns

real(dp), dimension(size(quantiles,1)) :: FlowDurationCurve — Flow Duration Curve value at resp. quantile

Author

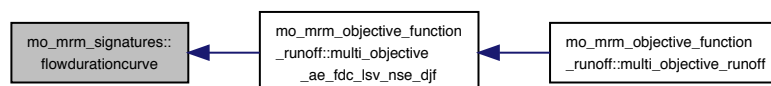
Remko Nijzink, Juliane Mai

Date

March 2014

Referenced by `mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf()`.

Here is the caller graph for this function:



15.41.2.3 limb_densities()

```

subroutine, public mo_mrm_signatures::limb_densities (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(size(data,1)), intent(in), optional mask,
    real(dp), intent(out), optional RLD,
    real(dp), intent(out), optional DLD )
  
```

Calculates limb densities.

Calculates rising and declining limb densities. The peaks of the given series are first determined by looking for points where preceding and subsequent datapoint are lower. Second, the number of datapoints with rising values

(nrise) and declining values (ndecline) are counted basically by comparing neighbors.

The duration the data increase (nrise) divided by the number of peaks (npeaks) gives the rising limb density RLD

$$RLD = t_{rise} / n_{peak}$$

whereas the duration the data decrease (ndecline) divided by the number of peaks (npeaks) gives the declining limb density DLD

$$DLD = t_{fall} / n_{peak}.$$

An optional mask of data points can be specified.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	data series
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask for data series
out	<i>real(dp), optional :: RLD</i>	rising limb density
out	<i>real(dp), optional :: DLD</i>	declining limb density

Author

Remko Nijzink

Date

March 2014

References mo_message::message().

Here is the call graph for this function:



15.41.2.4 maximummonthlyflow()

```

real(dp) function mo_mrm_signatures::maximummonthlyflow (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(size(data,1)), intent(in), optional mask,
    integer(i4), intent(in), optional yr_start,
    integer(i4), intent(in), optional mo_start,
    integer(i4), intent(in), optional dy_start )

```

Maximum of average flows per months.

Maximum of average flow per month is defined as

$$max_{monthlyflow} = Max(F(i), i = 1, ..12)$$

where \$F(i)\$ is the average flow of month i .

Parameters

in	<i>real(dp), dimension(:) :: data</i>	array of data
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask for data points given
in	<i>integer(i4) :: yr_start</i>	year of date of first data point given
in	<i>integer(i4) :: mo_start</i>	month of date of first data point given (default: 1)
in	<i>integer(i4) :: dy_start</i>	month of date of first data point given (default: 1)

Returns

real(dp) :: MaximumMonthlyFlow — Maximum of average flow per month Works only with 1d double precision input data.
Assumes data are daily values.

Author

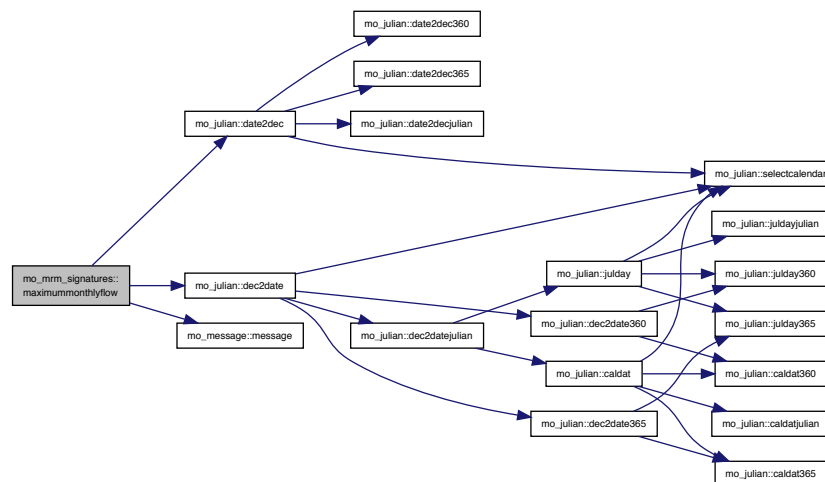
Juliane Mai

Date

Jun 2015

References `mo_julian::date2dec()`, `mo_julian::dec2date()`, and `mo_message::message()`.

Here is the call graph for this function:



15.41.2.5 moments()

```

subroutine, public mo_mrm_signatures::moments (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(size(data,1)), intent(in), optional mask,
    real(dp), intent(out), optional mean_data,
    real(dp), intent(out), optional stddev_data,

```

```

real(dp), intent(out), optional median_data,
real(dp), intent(out), optional max_data,
real(dp), intent(out), optional mean_log,
real(dp), intent(out), optional stddev_log,
real(dp), intent(out), optional median_log,
real(dp), intent(out), optional max_log )

```

Moments of data and log-transformed data, e.g. mean and standard deviation.

Returns several moments of data series given, i.e.

- mean of data
- standard deviation of data
- median of data
- maximum/ peak of data
- mean of log-transformed data
- standard deviation of log-transformed data
- median of log-transformed data
- maximum/ peak of log-transformed data An optional mask of data points can be specified.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	array of data
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask for data points given
out	<i>real(dp) :: mean_data</i>	mean of data
out	<i>real(dp) :: stddev_data</i>	standard deviation of data
out	<i>real(dp) :: median_data</i>	median of data
out	<i>real(dp) :: max_data</i>	maximum/ peak of data
out	<i>real(dp) :: mean_log</i>	mean of log-transformed data
out	<i>real(dp) :: stddev_log</i>	standard deviation of log-transformed data
out	<i>real(dp) :: median_log</i>	median of log-transformed data
out	<i>real(dp) :: max_log</i>	maximum/ peak of log-transformed data Works only with 1d double precision input data.

Author

Juliane Mai

Date

Jun 2015

References `mo_message::message()`.

Here is the call graph for this function:

**15.41.2.6 peakdistribution()**

```

real(dp) function, dimension(size(quantiles,1)), public mo_mrm_signatures::peakdistribution (
    real(dp), dimension(:), intent(in) data,
    real(dp), dimension(:), intent(in) quantiles,
    logical, dimension(size(data,1)), intent(in), optional mask,
    real(dp), intent(out), optional slope_peak_distribution )
  
```

Calculates the peak distribution.

First, the peaks of the time series given are identified. For the peak distribution only this subset of data points are considered. Second, the peak distribution at the quantiles given is calculated. Calculates the peak distribution at the quantiles given using [mo_percentile](#). Since the exceedance probabilities are usually used in hydrology the function percentile is used with (1.0-quantiles).

Optionally, the slope of the peak distribution between 10th and 50th percentile, i.e.

$$slope = \frac{peak_data_{0.1} - peak_data_{0.5}}{0.9 - 0.5}$$

can be returned.

An optional mask for the data points can be given.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	data array
in	<i>real(dp), dimension(:) :: quantiles</i>	requested quantiles for distribution
in	<i>logical, dimension(size(data,1)), optional :: mask</i>	mask of data array
in	<i>real(dp), optional :: slope_peak_distribution</i>	slope of the Peak distribution between 10th and 50th percentile

Returns

real(dp), dimension(size(quantiles,1)) :: PeakDistribution — Distribution of peak values at resp. quantiles

Author

Remko Nijzink

Date

March 2014

15.41.2.7 runoffratio()

```

real(dp) function, public mo_mrm_signatures::runoffratio (
    real(dp), dimension(:), intent(in) data,
    real(dp), intent(in) basin_area,
    logical, dimension(size(data,1)), intent(in), optional mask,
    real(dp), dimension(size(data,1)), intent(in), optional precip_series,
    real(dp), intent(in), optional precip_sum,
    logical, intent(in), optional log_data )

```

Runoff ratio (accumulated daily discharge [mm/d] / accumulated daily precipitation [mm/d]).

The runoff ratio is defined as

$$runoff_{ratio} = \frac{\sum_{t=1}^N q_t}{\sum_{t=1}^N p_t}$$

where p_t and q_t are precipitation and discharge, respectively.

Therefore, precipitation over the entire basin is required and both discharge and precipitation have to be converted to the same units [mm/d].

Input discharge is given in [m**3/s] as this is mHM default while precipitation has to be given in [mm/km**2 / day]. Either "precip_sum" or "precip_series" has to be specified. If "precip_series" is used the optional mask is also applied to precipitation values. The "precip_sum" is the accumulated "precip_series".

Optionally, a mask for the data (=discharge) can be given. If optional "log_data" is set to .true. the runoff ratio will be calculated as

$$runoff_{ratio} = \frac{\sum_{t=1}^N \log(q_t)}{\sum_{t=1}^N p_t}$$

where p_t and q_t are precipitation and discharge, respectively.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	array of data [m**3/s]
in	<i>real(dp) :: basin_area</i>	area of basin [km**2]
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask for data points given
in	<i>real(dp) :: precip_sum</i>	sum of daily precip. values of whole period [mm/km**2 / day]
in	<i>real(dp), dimension(size(data,1)) :: precip_series</i>	daily precipitation values [mm/km**2 / day]
in	<i>logical :: log_data</i>	ratio using logarithmic data Works only with 1d double precision input data.

Author

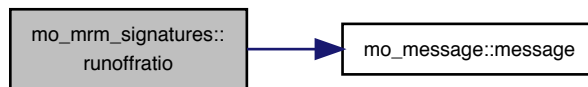
Juliane Mai

Date

Jun 2015

References mo_message::message().

Here is the call graph for this function:

**15.41.2.8 zeroflowratio()**

```

real(dp) function, public mo_mrm_signatures::zeroflowratio (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(size(data,1)), intent(in), optional mask )
  
```

Ratio of zero values to total number of data points.

An optional mask of data points can be specified.

Parameters

in	<i>real(dp), dimension(:) :: data</i>	array of data
in	<i>logical, dimension(size(data,1)) :: mask</i>	mask for data points given Works only with 1d double precision input data.

Author

Juliane Mai

Date

Jun 2015

References mo_message::message().

Here is the call graph for this function:



15.42 mo_mrm_tools Module Reference

Provide utility routines used within mRM.

Functions/Subroutines

- subroutine, public [get_basin_info_mrm](#) (iBasin, iLevel, nrows, ncols, ncells, iStart, iEnd, iStartMask, iEndMask, mask, xllcorner, yllcorner, cellsize)
Get basic basin information (e.g., nrows, ncols, indices, mask)
- subroutine, public [calculate_grid_properties](#) (nrowsIn, ncolsIn, xllcornerIn, yllcornerIn, cellsizeIn, nodata_valueIn, aimingResolution, nrowsOut, ncolsOut, xllcornerOut, yllcornerOut, cellsizeOut, nodata_valueOut)
Calculates basic grid properties at a required coarser level using information of a given finer level.

15.42.1 Detailed Description

Provide utility routines used within mRM.

This module contains subroutines that are used frequently to obtain basin and grid properties.

Author

Luis Samaniego

Date

Dec 2012

15.42.2 Function/Subroutine Documentation

15.42.2.1 calculate_grid_properties()

```
subroutine, public mo_mrm_tools::calculate_grid_properties (
    integer(i4), intent(in) nrowsIn,
    integer(i4), intent(in) ncolsIn,
    real(dp), intent(in) xllcornerIn,
    real(dp), intent(in) yllcornerIn,
    real(dp), intent(in) cellsizeIn,
    real(dp), intent(in) nodata_valueIn,
    real(dp), intent(in) aimingResolution,
    integer(i4), intent(out) nrowsOut,
    integer(i4), intent(out) ncolsOut,
    real(dp), intent(out) xllcornerOut,
    real(dp), intent(out) yllcornerOut,
    real(dp), intent(out) cellsizeOut,
    real(dp), intent(out) nodata_valueOut )
```

Calculates basic grid properties at a required coarser level using information of a given finer level.

Calculates basic grid properties at a required coarser level (e.g., L11) using information of a given finer level (e.g., L0). Basic grid properties such as nrows, ncols, xllcorner, yllcorner cellsize are estimated in this routine.

Parameters

in	<i>integer(i4) :: nrowsIn</i>	no. of rows at an input level
----	-------------------------------	-------------------------------

Parameters

in	<i>integer(i4) :: ncolsIn</i>	no. of cols at an input level
in	<i>real(dp) :: xllcornerIn</i>	xllcorner at an input level
in	<i>real(dp) :: yllcornerIn</i>	yllcorner at an input level
in	<i>real(dp) :: cellsizeIn</i>	cell size at an input level
in	<i>real(dp) :: nodata_valueIn</i>	nodata value at an input level
in	<i>real(dp) :: aimingResolution</i>	resolution of an output level
out	<i>integer(i4) :: nrowsOut</i>	no. of rows at an output level
out	<i>integer(i4) :: ncolsOut</i>	no. of cols at an output level
out	<i>real(dp) :: xllcornerOut</i>	xllcorner at an output level
out	<i>real(dp) :: yllcornerOut</i>	yllcorner at an output level
out	<i>real(dp) :: cellsizeOut</i>	cell size at an output level
out	<i>real(dp) :: nodata_valueOut</i>	nodata value at an output level

Note

resolutions of input and output levels should confirm each other.

Author

Matthias Zink & Rohini Kumar

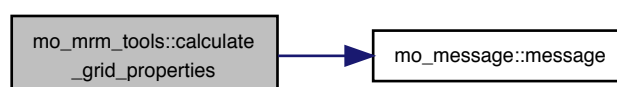
Date

Feb 2013

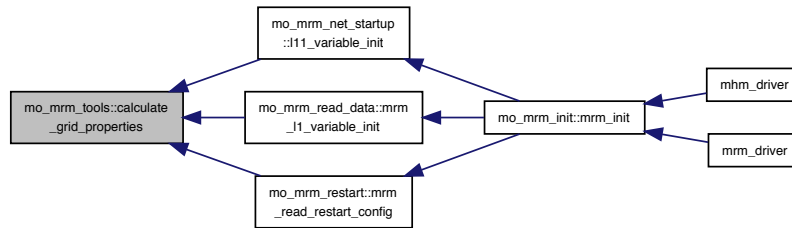
References `mo_kind::dp`, `mo_kind::i4`, and `mo_message::message()`.

Referenced by `mo_mrm_net_startup::l11_variable_init()`, `mo_mrm_read_data::mrm_l1_variable_init()`, and `mo_mrm_restart::mrm_read_restart_config()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.42.2.2 get_basin_info_mrm()

```

subroutine, public mo_mrm_tools::get_basin_info_mrm (
    integer(i4), intent(in) iBasin,
    integer(i4), intent(in) iLevel,
    integer(i4), intent(out) nRows,
    integer(i4), intent(out) nCols,
    integer(i4), intent(out), optional nCells,
    integer(i4), intent(out), optional iStart,
    integer(i4), intent(out), optional iEnd,
    integer(i4), intent(out), optional iStartMask,
    integer(i4), intent(out), optional iEndMask,
    logical, dimension(:,:), intent(out), optional, allocatable mask,
    real(dp), intent(out), optional xllcorner,
    real(dp), intent(out), optional yllcorner,
    real(dp), intent(out), optional cellsize )
  
```

Get basic basin information (e.g., nRows, nCols, indices, mask)

Get basic basin information (e.g., nRows, nCols, indices, mask) for different levels (L0, L1, L11, and L110).

Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
in	<i>integer(i4) :: iLevel</i>	level id (e.g., 0, 1, 11, 2)
out	<i>integer(i4) :: nRows</i>	no. of rows
out	<i>integer(i4) :: nCols</i>	no. of columns
out	<i>integer(i4) :: nCells</i>	no. of cells
out	<i>integer(i4) :: iStart</i>	start cell index of a given basin at a given level
out	<i>integer(i4) :: iEnd</i>	end cell index of a given basin at a given level
out	<i>integer(i4) :: iStartMask</i>	start cell index of mask a given basin at a given level
out	<i>integer(i4) :: iEndMask</i>	end cell index of mask a given basin at a given level
out	<i>logical, optional :: mask</i>	Mask at a given level
out	<i>real(dp), optional :: xllcorner</i>	x coordinate of the lowerleft corner at a given level
out	<i>real(dp), optional :: yllcorner</i>	y coordinate of the lowerleft corner at a given level
out	<i>real(dp), optional :: cellsize</i>	cell size at a given level

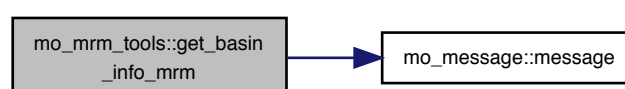
Note

This subroutine cannot be called for level 2, that does not exist within mRM.

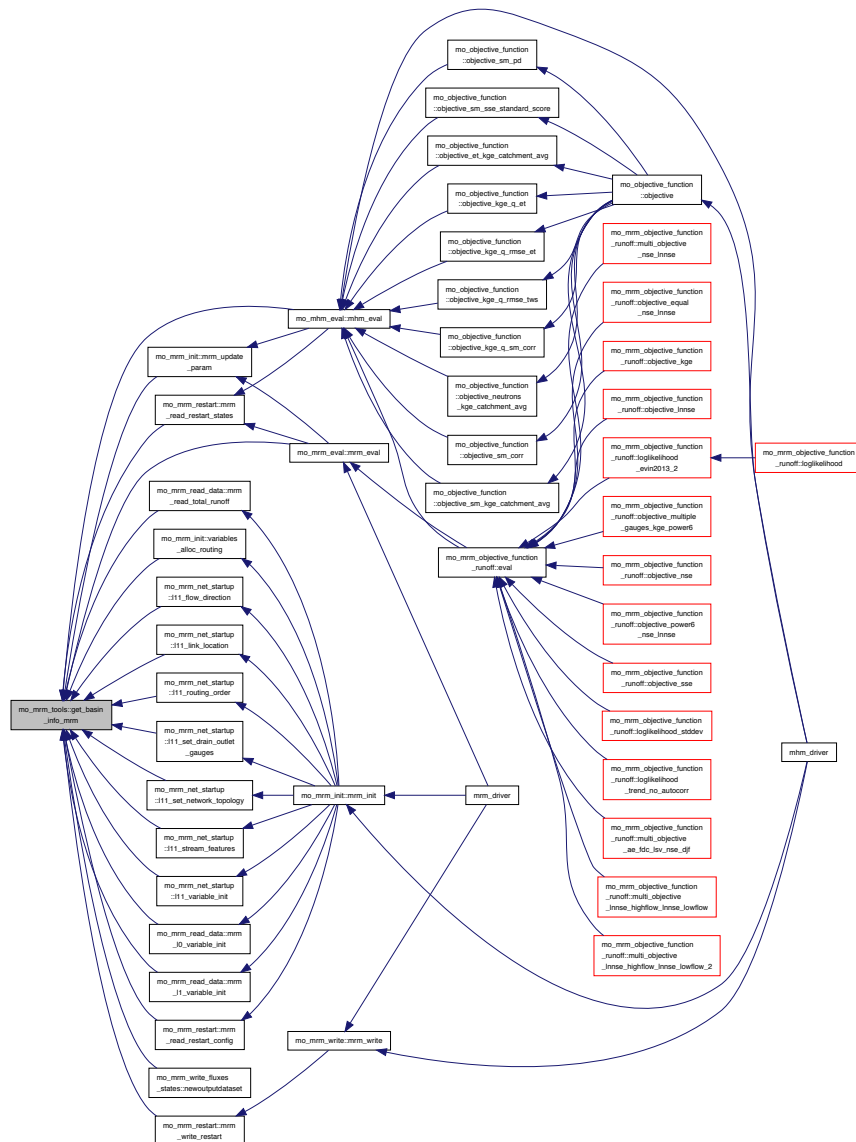
References `mo_mrm_global_variables::basin_mrm`, `mo_kind::dp`, `mo_kind::i4`, `mo_mrm_global_variables::level0`, `mo_mrm_global_variables::level1`, `mo_mrm_global_variables::level11`, and `mo_message::message()`.

Referenced by `mo_mrm_net_startup::l11_flow_direction()`, `mo_mrm_net_startup::l11_link_location()`, `mo_mrm_net_startup::l11_routing_order()`, `mo_mrm_net_startup::l11_set_drain_outlet_gauges()`, `mo_mrm_net_startup::l11_set_network_topology()`, `mo_mrm_net_startup::l11_stream_features()`, `mo_mrm_net_startup::l11_variable_init()`, `mo_mhm_eval::mhm_eval()`, `mo_mrm_eval::mrm_eval()`, `mo_mrm_read_data::mrm_l0_variable_init()`, `mo_mrm_read_data::mrm_l1_variable_init()`, `mo_mrm_restart::mrm_read_restart_config()`, `mo_mrm_restart::mrm_read_restart_states()`, `mo_mrm_read_data::mrm_read_total_runoff()`, `mo_mrm_init::mrm_update_param()`, `mo_mrm_restart::mrm_write_restart()`, `mo_mrm_write_fluxes_states::newoutputdataset()`, and `mo_mrm_init::variables_alloc_routing()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.43 mo_mrm_write Module Reference

write of discharge and restart files

Functions/Subroutines

- subroutine, public [mrm_write](#) ()
write discharge and restart files
- subroutine [write_configfile](#) ()
This modules writes the results of the configuration into an ASCII-file.
- subroutine [write_daily_obs_sim_discharge](#) (Qobs, Qsim)
Write a file for the daily observed and simulated discharge timeseries during the evaluation period for each gauging station.

- subroutine, public [mrm_write_output_fluxes](#) (iBasin, timeStep_model_outputs, warmingDays_mrm, new←Time, nTimeSteps, nTStepDay, tt, day, month, year, timestep, mask11, L11_qmod)
write fluxes to netcdf output files
- subroutine, public [mrm_write_optifile](#) (best_OF, best_paramSet, param_names)
Write briefly final optimization results.
- subroutine, public [mrm_write_optinamelist](#) (parameters, maskpara, parameters_name)
Write final, optimized parameter set in a namelist format.

Variables

- integer(i4) [day_counter](#)
- integer(i4) [month_counter](#)
- integer(i4) [year_counter](#)
- integer(i4) [average_counter](#)
- type(outputdataset) [nc](#)

15.43.1 Detailed Description

write of discharge and restart files

This module contains the subroutines for writing the discharge files and optionally the restart files.

Author

Stephan Thober

Date

Aug 2015

15.43.2 Function/Subroutine Documentation

15.43.2.1 mrm_write()

```
subroutine, public mo_mrm_write::mrm_write ( )
```

write discharge and restart files

First, this subroutine calls the writing or restart files that only succeeds if it happens after the write of mHM restart files because mHM restart files must exist. Second, simulated discharge is aggregated to the daily scale and then written to file jointly with observed discharge

Author

Juliane Mai, Rohini Kumar & Stephan Thober

Aug 2015

Referenced by `mhm_driver()`, and `mrm_driver()`.

```

graph LR
    A[mo_mrm_write:mrm_write] --> B[mo_mrm_restart:mrm_write_restart]
    A --> C[mo_mrm_write:write_configfile]
    A --> D[mo_mrm_write:write_daily_obs_sim_discharge]
    B --> E[mo_mrm_tools:get_basin_info_mrm]
    B --> F[mo_message:message]
    C --> F
    D --> E
    D --> G[mo_julian:dec2date]
    D --> H[mo_julian:dec2datejulian]
    D --> I[mo_julian:julday]
    D --> J[mo_julian:caldat]
    D --> K[mo_julian:dec2date365]
    D --> L[mo_julian:dec2date360]
    E --> F
    G --> M[mo_julian:julday365]
    G --> N[mo_julian:caldat365]
    G --> O[mo_julian:juldayjulian]
    G --> P[mo_julian:selectcalendar]
    G --> Q[mo_julian:julday360]
    G --> R[mo_julian:caldatjulian]
    G --> S[mo_julian:caldat360]
    H --> M
    H --> N
    H --> O
    H --> P
    H --> Q
    H --> R
    H --> S
    I --> M
    I --> N
    I --> O
    I --> P
    I --> Q
    I --> R
    I --> S
    J --> M
    J --> N
    J --> O
    J --> P
    J --> Q
    J --> R
    J --> S
    K --> M
    K --> N
    K --> O
    K --> P
    K --> Q
    K --> R
    K --> S
    L --> M
    L --> N
    L --> O
    L --> P
    L --> Q
    L --> R
    L --> S
  
```

```

graph LR
    mhm_driver --> mo_mrm_write
    mrm_driver --> mo_mrm_write
    subgraph mo_mrm_write [mo_mrm_write::mrm_write]
    end

```

```

subroutine, public mo_mrm_write::mrm_write_optifile (
    real(dp), intent(in) best_OF,
    real(dp), dimension(:), intent(in) best_paramSet,
    character(len=*), dimension(:), intent(in) param_names )

```

Write overall best objective function and the best optimized parameter set to a file_opti.

Parameters

in	<i>real(dp) :: best_OF</i>	best objective function value as returned by the optimization routine
in	<i>real(dp), dimension(:) :: best_paramSet</i>	best associated global parameter set Called only when optimize is .TRUE.

Author

David Schaefer

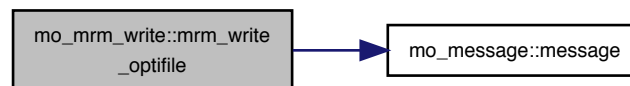
Date

July 2013

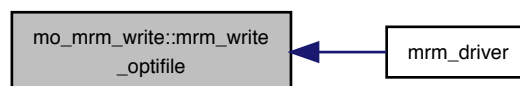
References `mo_mrm_global_variables::dirconfigout`, `mo_mrm_file::file_opti`, `mo_message::message()`, and `mo_mrm_file::uopti`.

Referenced by `mrm_driver()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**15.43.2.3 mrm_write_optinamelist()**

```

subroutine, public mo_mrm_write::mrm_write_optinamelist (
    real(dp), dimension(:, :), intent(in) parameters,
    logical, dimension(size(parameters,1)), intent(in) maskpara,
    character(len=*), dimension(size(parameters,1)), intent(in) parameters_name )
  
```

Write final, optimized parameter set in a namelist format.

Write final, optimized parameter set in a namelist format.

Parameters

in	<i>real(dp) :: parameters(:,:)</i>	information about parameter (min, max, opti)
in	<i>logical :: maskpara(:)</i>	information which parameter where optimized
in	<i>character(len=*) :: parameters_name(:)</i>	clear names of parameters Called only when optimize is .TRUE.

Author

Juliane Mai

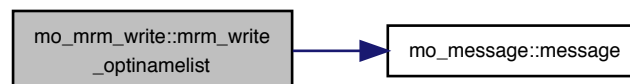
Date

Dec 2013

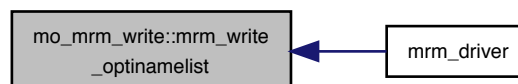
References mo_mrm_global_variables::dirconfigout, mo_mrm_file::file_opti_nml, mo_message::message(), mo_common_variables::processmatrix, and mo_mrm_file::uopti_nml.

Referenced by mrm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.43.2.4 mrm_write_output_fluxes()

```

subroutine, public mo_mrm_write::mrm_write_output_fluxes (
    integer(i4), intent(in) iBasin,
    integer(i4), intent(in) timeStep_model_outputs,
    integer(i4), intent(in) warmingDays_mrm,
    real(dp), intent(in) newTime,
    integer(i4), intent(in) nTimeSteps,

```

```

integer(i4), intent(in) nTStepDay,
integer(i4), intent(in) tt,
integer(i4), intent(in) day,
integer(i4), intent(in) month,
integer(i4), intent(in) year,
integer(i4), intent(in) timestep,
logical, dimension(:,:), intent(in) mask11,
real(dp), dimension(:), intent(in) L11_qmod )

```

write fluxes to netcdf output files

This subroutine creates a netcdf data set for writing L11_QTIN for different time averages.

Author

Stephan Thober

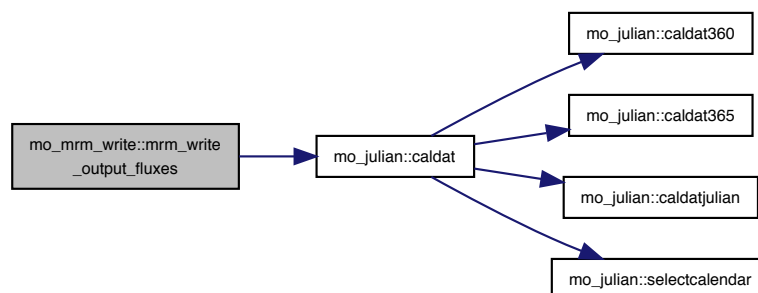
Date

Aug 2015

References `average_counter`, `mo_julian::caldat()`, `day_counter`, `mo_kind::dp`, `mo_kind::i4`, `month_counter`, `nc`, and `year_counter`.

Referenced by `mo_mhm_eval::mhm_eval()`, and `mo_mrm_eval::mrm_eval()`.

Here is the call graph for this function:



[illegible]

```
subroutine mo_mrm_write::write_configfile ( )
```

Author
Christoph Schneider

```
References mo_mrm_global_variables::basin_mrm, mo_mrm_global_variables::dirconfigout, mo_mrm_global_
_variables::dirgauges, mo_mrm_global_variables::dirlcover, mo_mrm_global_variables::dirmorpho, mo_mrm_
global_variables::dirout, mo_mrm_global_variables::dirrestartout, mo_mrm_global_variables::dirtotalrunoff, mo_
kind::dp, mo_mrm_global_variables::evalper, mo_mrm_file::file_config, mo_mrm_global_variables::gauge, mo_
_common_variables::global_parameters, mo_common_variables::global_parameters_name, mo_kind::i4, mo_
mrm_global_variables::inflowgauge, mo_mrm_global_variables::l0_ncells, mo_mrm_global_variables::l11_
fromn, mo_mrm_global_variables::l11_id, mo_mrm_global_variables::l11_l1_id, mo_mrm_global_variables::
```

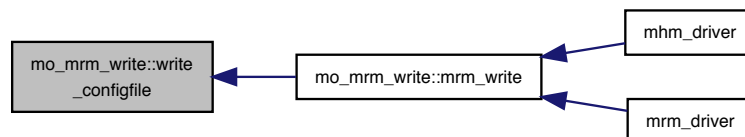
::l11_label, mo_mrm_global_variables::l11_length, mo_mrm_global_variables::l11_ncells, mo_mrm_global_variables::l11_netperm, mo_mrm_global_variables::l11_rorder, mo_mrm_global_variables::l11_slope, mo_mrm_global_variables::l11_ton, mo_mrm_global_variables::l1_areacell, mo_mrm_global_variables::l1_l11_id, mo_mrm_global_variables::l1_ncells, mo_mrm_global_variables::lcfilename, mo_mrm_global_variables::lcyearid, mo_message::message(), mo_mrm_global_variables::mrm_coupling_mode, mo_mrm_global_variables::nbasins, mo_mrm_global_variables::ngaugestotal, mo_mrm_global_variables::ninflowgaugestotal, mo_mrm_constants::nodata_dp, mo_common_variables::processmatrix, mo_mrm_global_variables::read_restart, mo_mrm_global_variables::resolutionhydrology, mo_mrm_global_variables::resolutionrouting, mo_mrm_global_variables::simper, mo_mrm_global_variables::timestep, mo_mrm_file::uconfig, mo_mrm_file::version, mo_mrm_global_variables::warmper, and mo_mrm_global_variables::write_restart.

Referenced by mrm_write().

Here is the call graph for this function:



Here is the caller graph for this function:



15.43.2.6 write_daily_obs_sim_discharge()

```

subroutine mo_mrm_write::write_daily_obs_sim_discharge (
    real(dp), dimension(:,:), intent(in) Qobs,
    real(dp), dimension(:,:), intent(in) Qsim )
  
```

Write a file for the daily observed and simulated discharge timeseries during the evaluation period for each gauging station.

Write a file for the daily observed and simulated discharge timeseries during the evaluation period for each gauging station

Parameters

in	<i>real(dp), dimension(:, :) :: Qobs</i>	daily time series of observed discharge dims = (nModeling_days , nGauges_total)
----	--	---

Parameters

in	<i>real(dp), dimension(:,:) :: Qsim</i>	daily time series of modeled discharge dims = (nModeling_days , nGauges_total)
----	---	--

Author

Rohini Kumar

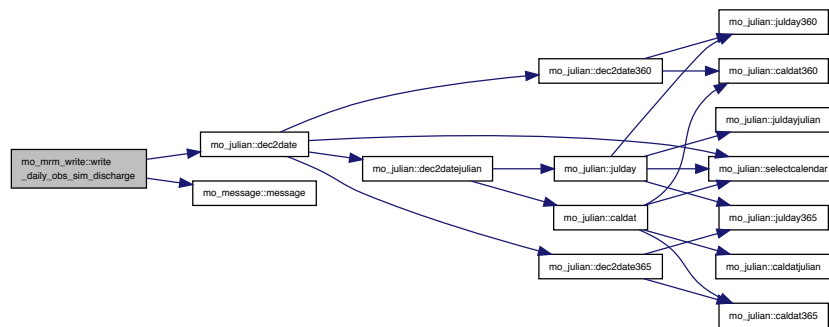
Date

August 2013

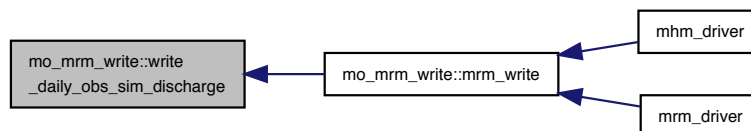
References mo_mrm_global_variables::basin_mrm, mo_julian::dec2date(), mo_mrm_global_variables::dirout, mo_mrm_global_variables::evalper, mo_mrm_file::file_daily_discharge, mo_mrm_global_variables::gauge, mo_message::message(), mo_mrm_global_variables::nbasins, mo_mrm_file::ncfile_discharge, and mo_mrm_file::udaily_discharge.

Referenced by mrm_write().

Here is the call graph for this function:



Here is the caller graph for this function:



15.43.3 Variable Documentation

15.43.3.1 average_counter

```
integer(i4) mo_mrm_write::average_counter [private]
```

Referenced by `mrm_write_output_fluxes()`.

15.43.3.2 day_counter

```
integer(i4) mo_mrm_write::day_counter [private]
```

Referenced by `mrm_write_output_fluxes()`.

15.43.3.3 month_counter

```
integer(i4) mo_mrm_write::month_counter [private]
```

Referenced by `mrm_write_output_fluxes()`.

15.43.3.4 nc

```
type(outputdataset) mo_mrm_write::nc [private]
```

Referenced by `mrm_write_output_fluxes()`.

15.43.3.5 year_counter

```
integer(i4) mo_mrm_write::year_counter [private]
```

Referenced by `mrm_write_output_fluxes()`.

15.44 mo_mrm_write_fluxes_states Module Reference

Creates NetCDF output for different fluxes and state variables of mHM.

Data Types

- interface [outputdataset](#)
- interface [outputvariable](#)

Functions/Subroutines

- type([outputvariable](#)) function [newoutputvariable](#) (nc, name, dtype, dims, ncells, mask, avg)
Initialize OutputVariable.
- subroutine [updatevariable](#) (self, data)
Update OutputVariable.
- subroutine [writevariabletimestep](#) (self, timestep)
Write timestep to file.

- type([outputdataset](#)) function [newoutputdataset](#) (ibasin, mask)
Initialize OutputDataset.
- subroutine [updatedataset](#) (self, sidx, eid, L11_Qmod)
Update all variables.
- subroutine [writetimestep](#) (self, timestep)
Write all accumulated data.
- subroutine [close](#) (self)
Close the file.
- type([ncdataset](#)) function [createoutputfile](#) (ibasin)
Create and initialize output file.
- subroutine [writevariableattributes](#) (var, long_name, unit)
Write output variable attributes.
- subroutine [mapcoordinates](#) (ibasin, level, y, x)
Generate map coordinates.
- subroutine [geocoordinates](#) (ibasin, level, lat, lon)
Generate geographic coordinates.
- character(16) function [fluxesunit](#) (ibasin)
Generate a unit string.

15.44.1 Detailed Description

Creates NetCDF output for different fluxes and state variables of mHM.

NetCDF is first initialized and later on variables are put to the NetCDF.

Authors

Matthias Zink

Date

Apr 2013

15.44.2 Function/Subroutine Documentation

15.44.2.1 [close\(\)](#)

```
subroutine mo_mrm_write_fluxes_states::close (
    class(outputdataset) self ) [private]
```

Close the file.

Close the file associated with variable of type([OutputDataset](#))

Author

Rohini Kumar & Stephan Thober

Date

August 2013

References `mo_mrm_global_variables::dirout`, and `mo_message::message()`.

Here is the call graph for this function:



15.44.2.2 createoutputfile()

```
type(ncdataset) function mo_mrm_write_fluxes_states::createoutputfile (
    integer(i4), intent(in) ibasin )
```

Create and initialize output file.

Create output file, write all non-dynamic variables and global attributes for the given basin.

Parameters

in	<i>integer(i4) :: ibasin</i>	-> basin id
in	<i>logical, target :: mask1(:, :)</i>	-> level11 mask

Author

David Schaefer

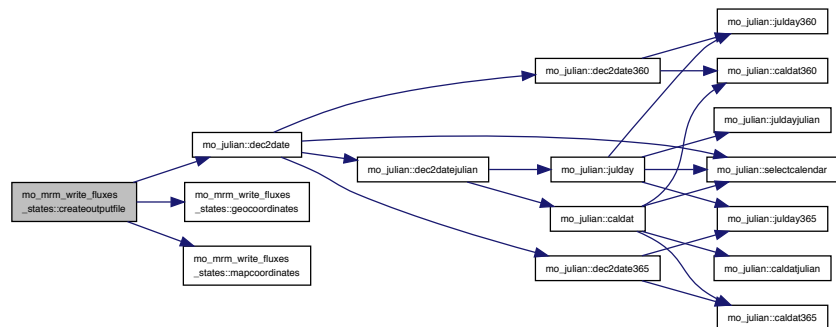
Date

June 2015

References mo_julian::dec2date(), mo_mrm_global_variables::dirout, mo_mrm_global_variables::evalper, mo_mrm_file::file_mrm_output, geocoordinates(), mo_mrm_global_variables::level11, mapcoordinates(), and mo_mrm_file::version.

Referenced by newoutputdataset().

Here is the call graph for this function:



Here is the caller graph for this function:



15.44.2.3 fluxesunit()

```
character(16) function mo_mrm_write_fluxes_states::fluxesunit (
    integer(i4), intent(in) ibasin )
```

Generate a unit string.

Generate the unit string for the output variable netcdf attribute based on modeling timestep

Parameters

in	integer(i4) :: iBasin	-> basin id
----	-----------------------	-------------

Author

David Schaefer

Date

June 2015

References `mo_mrm_global_variables::ntstepday`, `mo_mrm_global_variables::simper`, `mo_mrm_global_variables::timestep`, and `mo_mrm_global_variables::timestep_model_outputs_mrm`.

15.44.2.4 geocoordinates()

```
subroutine mo_mrm_write_fluxes_states::geocoordinates (
    integer(i4), intent(in) ibasin,
    type(gridgeoref), intent(in) level,
    real(dp), dimension(:, :), intent(out), allocatable lat,
    real(dp), dimension(:, :), intent(out), allocatable lon ) [private]
```

Generate geographic coordinates.

Generate geographic coordinate arrays for given basin and level

Parameters

in	<i>integer(i4) :: iBasin</i>	-> basin number
in	<i>type(gridGeoRef) :: level</i>	-> grid reference
out	<i>real(dp) :: lat(:, :)</i>	-> lat-coordinates
out	<i>real(dp) :: lon(:, :)</i>	-> lon-coorindates

Author

Matthias Zink

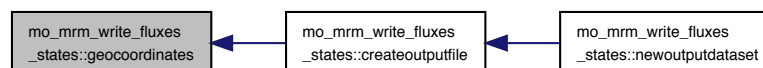
Date

Apr 2013

References `mo_mrm_global_variables::l11_rect_latitude`, and `mo_mrm_global_variables::l11_rect_longitude`.

Referenced by `createoutputfile()`.

Here is the caller graph for this function:



15.44.2.5 mapcoordinates()

```

subroutine mo_mrm_write_fluxes_states::mapcoordinates (
    integer(i4), intent(in) ibasin,
    type(gridgeoref), intent(in) level,
    real(dp), dimension(:), intent(out), allocatable y,
    real(dp), dimension(:), intent(out), allocatable x ) [private]

```

Generate map coordinates.

Generate map coordinate arrays for given basin and level

Parameters

in	<i>integer(i4) :: ibasin</i>	-> basin number
in	<i>type(geoGridRef) :: level</i>	-> grid reference
out	<i>real(:) :: y(:)</i>	-> y-coordinates
out	<i>real(dp) :: x(:)</i>	-> x-coorindates

Author

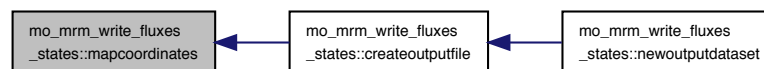
Matthias Zink

Date

Apr 2013

Referenced by createoutputfile().

Here is the caller graph for this function:



15.44.2.6 newoutputdataset()

```

type(outputdataset) function mo_mrm_write_fluxes_states::newoutputdataset (
    integer(i4), intent(in) ibasin,
    logical, dimension(:,:), target mask ) [private]

```

Initialize OutputDataset.

Create and initialize the output file. If new a new output variable needs to be written, this is the first of two procedures to change (second: updateDataset)

Parameters

in	<i>integer(i4) :: ibasin</i>	-> basin id
in	<i>logical :: mask1</i>	-> L1 mask to reconstruct the data

Author

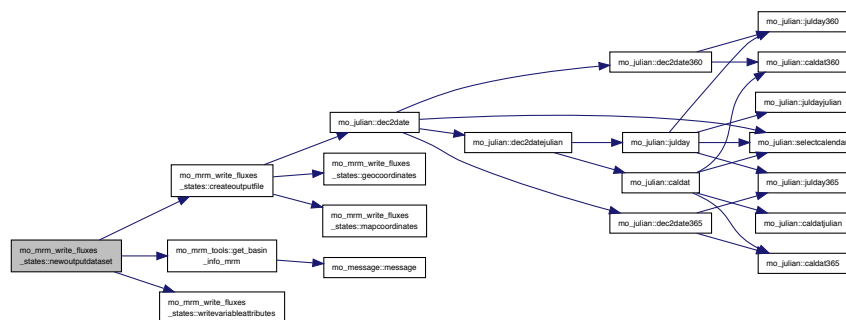
Matthias Zink

Date

Apr 2013

References `createoutputfile()`, `mo_mrm_tools::get_basin_info_mrm()`, `mo_mrm_global_variables::outputflxstate_`↔
`mrm`, and `writevariableattributes()`.

Here is the call graph for this function:



15.44.2.7 newoutputvariable()

```

type(outputvariable) function mo_mrm_write_fluxes_states::newoutputvariable (
    type(ncdataset), intent(in) nc,
    character(*), intent(in) name,
    character(*), intent(in) dtype,
    character(16), dimension(3), intent(in) dims,
    integer(i4), intent(in) ncells,
    logical, dimension(:,:), intent(in), target mask,
    logical, intent(in), optional avg ) [private]

```

Initialize OutputVariable.

Parameters

in	<i>type(NcDataset) :: nc</i>	-> NcDataset which contains the variable
in	<i>integer(i4) :: ncells</i>	-> number of cells in basin
in	<i>logical, target :: mask(:, :)</i>	-> mask to reconstruct data
in	<i>logical, optional :: avg</i>	-> average the data before writing

Author

David Schaefer

Date

June 2015

15.44.2.8 updatedataset()

```

subroutine mo_mrm_write_fluxes_states::updatedataset (
    class(outputdataset), intent(inout), target self,
    integer(i4), intent(in)  sidx,
    integer(i4), intent(in)  eidx,
    real(dp), dimension(:), intent(in) L11_Qmod )

```

Update all variables.

Call the type bound procedure updateVariable for all output variables. If a new output variable needs to be written, this is the second of two procedures to change (first: newOutputDataset)

Parameters

in	<i>sidx</i>	-> start index of the basin related data in L1_* arguments
in	<i>eidx</i>	-> end index of the basin related data in L1_* arguments
in	<i>L11_qMod</i>	

Author

Matthias Zink

Date

Apr 2013

References mo_mrm_global_variables::outputfluxstate_mrm, and updatevariable().

Here is the call graph for this function:

**15.44.2.9 updatevariable()**

```

subroutine mo_mrm_write_fluxes_states::updatevariable (
    class(outputvariable), intent(inout) self,
    real(dp), dimension(:), intent(in) data ) [private]

```

Update OutputVariable.

Add the array given as actual argument to the derived type's component 'data'

Parameters

in	<i>type(NcDataset) :: nc</i>	-> NcDataset which contains the variable
in	<i>integer(i4) :: ncells</i>	-> number of cells in basin
in	<i>logical, target :: mask(:, :)</i>	-> mask to reconstruct data
in	<i>logical, optional :: avg</i>	-> average the data before writing

Author

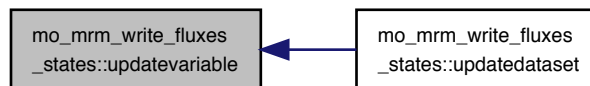
David Schaefer

Date

June 2015

Referenced by `updateddataset()`.

Here is the caller graph for this function:



15.44.2.10 writetimestep()

```

subroutine mo_mrm_write_fluxes_states::writetimestep (
    class(outputdataset), intent(inout), target self,
    integer(i4), intent(in) timestep )
  
```

Write all accumulated data.

Write all accumulated and potentially averaged data to disk.

Parameters

in	<i>integer(i4) :: timestep</i>	The model timestep to write
----	--------------------------------	-----------------------------

Author

David Schaefer

Date

June 2015

15.44.2.11 writevariableattributes()

```

subroutine mo_mrm_write_fluxes_states::writevariableattributes (
    type(outputvariable), intent(in) var,
    character(*), intent(in) long_name,
    character(*), intent(in) unit )

```

Write output variable attributes.

Parameters

in	<i>type(OutputVariable) :: var</i>	
in	<i>character(*) :: long_name</i>	-> variable name
in	<i>character(*) :: unit</i>	-> physical unit

Author

David Schaefer

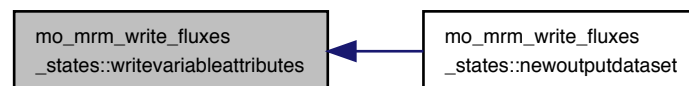
Date

June 2015

References mo_mrm_constants::nodata_dp.

Referenced by newoutputdataset().

Here is the caller graph for this function:



15.44.2.12 writevariabletimestep()

```

subroutine mo_mrm_write_fluxes_states::writevariabletimestep (
    class(outputvariable), intent(inout) self,
    integer(i4), intent(in) timestep ) [private]

```

Write timestep to file.

Write the content of the derived types's component 'data' to file, average if necessary

Parameters

in	<i>integer(i4) :: timestep</i>	-> index along the time dimension of the netcdf variable
----	--------------------------------	--

Author

David Schafer

Date

June 2015

References mo_mrm_constants::nodata_dp.

15.45 mo_multi_param_reg Module Reference

Multiscale parameter regionalization (MPR).

Functions/Subroutines

- subroutine, public [mpr](#) (proc_Mat, iFlag_soil, param, nodata, mask0, geoUnit0, geoUnitList, GeoUnitKar, LAI_LUT, LAI_UnitList, SDB_is_present, SDB_nHorizons, SDB_nTillHorizons, SDB_sand, SDB_clay, SDB_↔ DbM, SDB_Wd, SDB_RZdepth, nHorizons_mHM, horizon_depth, c2TSTu, fForest1, flperm1, fPerm1, soilld0, Asp0, LCover_LAI0, LCover0, slope_emp0, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, n↔ L0_in_L1, latitude, alpha1, IDDP1, DDmax1, DD1, fAsp1, HarSamCoeff1, PrieTayAlpha1, aeroResist1, surf↔ Resist1, fRoots1, K0_1, K1_1, K2_1, Kp1, karstic_loss, FC1, SMs1, beta1, jarvis_thresh_c1, TT1, HL1_1, HL3, PW1)
Regionalizing and Upscaling process parameters.
- subroutine [baseflow_param](#) (param, geoUnit0, geoUnitList, nodata, k2_0)
baseflow recession parameter
- subroutine [snow_acc_melt_param](#) (param, c2TSTu, fForest1, flperm1, fPerm1, TT1, DD1, IDDP1, DDmax1)
Calculates the snow parameters.
- subroutine [iper_thres_runoff](#) (param, HL3)
sets the impervious layer threshold parameter for runoff generation
- subroutine [karstic_layer](#) (param, geoUnitKar, geoUnit0, geoUnitlist, mask0, nodata, SMs_FC0, KsVar_V0, cell_id0, nL0_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, c2TSTu, karstic_loss, L1_Kp)
calculates the Karstic percolation loss
- subroutine, public [canopy_intercept_param](#) (proc_Mat, param, LAI0, nL0_in_L1, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, cell_id0, mask0, nodata, max_intercept1)
estimate effective maximum interception capacity at L1
- subroutine [aerodynamical_resistance](#) (LCover0, LAI_LUT, param, mask0, nodata, cell_id0, nL0_in_L1, Upp↔ _row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, aerodyn_resistance1)
Regionalization of aerodynamic resistance.

15.45.1 Detailed Description

Multiscale parameter regionalization (MPR).

This module provides the routines for multiscale parameter regionalization (MPR).

Authors

Stephan Thober, Rohini Kumar

Date

Dec 2012

15.45.2 Function/Subroutine Documentation

15.45.2.1 aerodynamical_resistance()

```

subroutine mo_multi_param_reg::aerodynamical_resistance (
    integer(i4), dimension(:), intent(in) LCover0,
    real(dp), dimension(:, :), intent(in) LAILUT,
    real(dp), dimension(6), intent(in) param,
    logical, dimension(:, :), intent(in) mask0,
    real(dp), intent(in) nodata,
    integer(i4), dimension(:), intent(in) cell_id0,
    integer(i4), dimension(:), intent(in) nL0_in_L1,
    integer(i4), dimension(:), intent(in) Upp_row_L1,
    integer(i4), dimension(:), intent(in) Low_row_L1,
    integer(i4), dimension(:), intent(in) Lef_col_L1,
    integer(i4), dimension(:), intent(in) Rig_col_L1,
    real(dp), dimension(:, :), intent(out) aerodyn_resistance1 )

```

Regionalization of aerodynamic resistance.

estimation of aerodynamical resistance Global parameters needed (see mhm_parameter.nml):

- param(1) = canopyheight_forest
- param(2) = canopyheight_impervious
- param(3) = canopyheight_pervious
- param(4) = displacementheight_coeff
- param(5) = roughnesslength_momentum_coeff
- param(6) = roughnesslength_heat_coeff

Parameters

in	<i>integer(i4) :: LCover0(:)</i>	- land cover at level 0
in	<i>real(dp) :: LAILUT(:)</i>	- LUT of LAi values
in	<i>integer(i4) :: LAIUnitList(:)</i>	- List of ids of each LAI class in LAILUT
in	<i>real(dp) :: param(:)</i>	- vector with global parameters
in	<i>logical :: mask0(:, :)</i>	- mask at level 0 field
in	<i>real(dp) :: nodata</i>	- nodata value
in	<i>integer(i4) :: cell_id0 (:)</i>	- Cell ids at level 0
in	<i>integer(i4) :: nL0_in_L1 (:)</i>	- Number of L0 cells within a L1 cell
in	<i>integer(i4) :: Upp_row_L1(:)</i>	- Upper row of high resolution block
in	<i>integer(i4) :: Low_row_L1(:)</i>	- Lower row of high resolution block
in	<i>integer(i4) :: Lef_col_L1(:)</i>	- Left column of high resolution block
in	<i>integer(i4) :: Rig_col_L1(:)</i>	- Right column of high resolution block
out	<i>real(dp) :: aerodyn_resistance1(:)</i>	- [s m-1] aerodynamical resistance

15.45.2.2 baseflow_param()

```

subroutine mo_multi_param_reg::baseflow_param (
    real(dp), dimension(:), intent(in) param,
    integer(i4), dimension(:), intent(in) geoUnit0,
    integer(i4), dimension(:), intent(in) geoUnitList,
    real(dp), intent(in) nodata,
    real(dp), dimension(:), intent(out) k2_0 )

```

baseflow recession parameter

This subroutine calculates the baseflow recession parameter based on the geological units at the Level 0 scale. For each level 0 cell, it assigns the value specified in the parameter array param for the geological unit in this cell. Global parameters needed (see mhm_parameter.nml):

- param(1) = GeoParam(1,:)
- param(2) = GeoParam(2,:)
- ...

Parameters

in	<i>real(dp) :: param(:)</i>	- array of global baseflow recession parameters
in	<i>integer(i4) :: geoUnit0(:, :)</i>	- array of geological units at Level 0
in	<i>integer(i4) :: geoUnitList(:)</i>	- array of indices for geological units.
in	<i>real(dp) :: nodata</i>	- no data value
out	<i>real(dp) :: k2_0</i>	- baseflow recession parameter at Level 0

Author

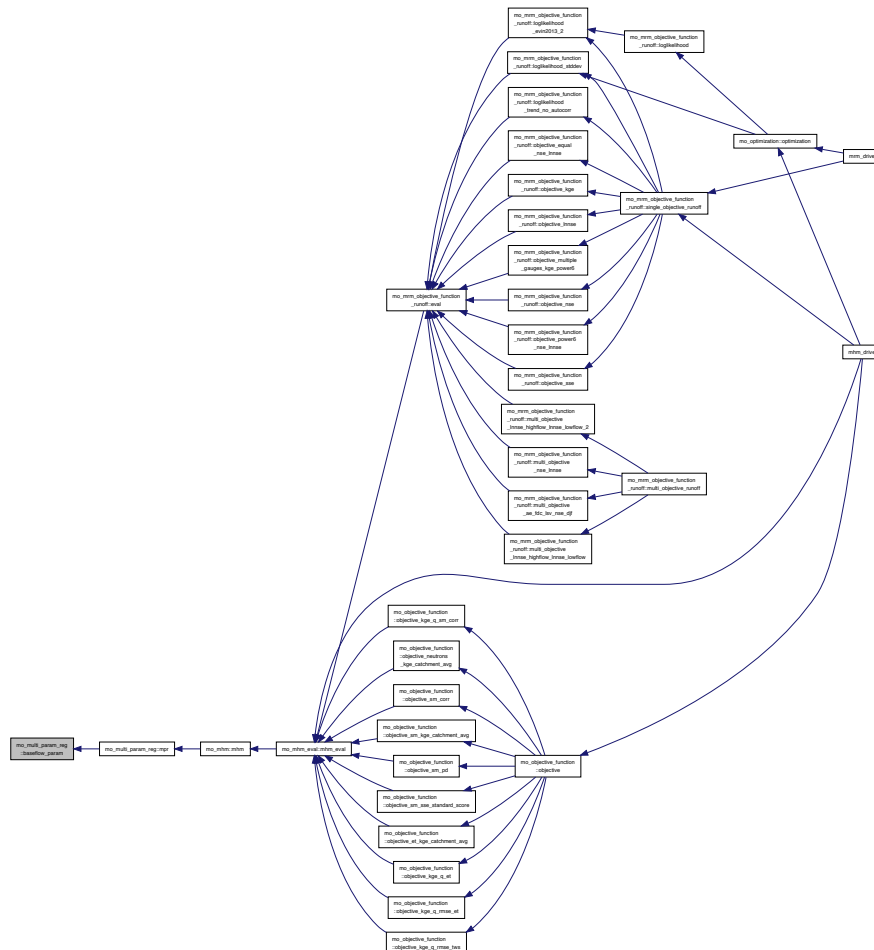
Stephan Thober, Rohini Kumar

Date

Dec 2012

Referenced by mpr().

Here is the caller graph for this function:



15.45.2.3 canopy_intercept_param()

```

subroutine, public mo_multi_param_reg::canopy_intercept_param (
  integer(i4), dimension(:, :), intent(in) proc_Mat,
  real(dp), dimension(:), intent(in) param,
  real(dp), dimension(:), intent(in) LAIO,
  integer(i4), dimension(:), intent(in) nLO_in_L1,
  integer(i4), dimension(:), intent(in) upp_row_L1,
  integer(i4), dimension(:), intent(in) low_row_L1,
  integer(i4), dimension(:), intent(in) lef_col_L1,
  integer(i4), dimension(:), intent(in) rig_col_L1,
  integer(i4), dimension(:), intent(in) cell_id0,
  logical, dimension(:, :), intent(in) mask0,
  real(dp), intent(in) nodata,
  real(dp), dimension(:), intent(out) max_intercept1 )

```

estimate effective maximum interception capacity at L1

estimate effective maximum interception capacity at L1 for a given Leaf Area Index field.

Global parameters needed (see mhm_parameter.nml):

Process Case 1:

- param(1) = canopyInterceptionFactor

Parameters

in	<i>integer(i4) :: proc_Mat(:, :)</i>	- process matrix
in	<i>real(dp) :: param(:)</i>	- array of global parameters
in	<i>real(dp) :: LAI0(:)</i>	- LAI at level-0
in	<i>integer(i4) :: nL0_in_L1 (:)</i>	- Number of L0 cells within a L1 cell
in	<i>integer(i4) :: upp_row_L1(:)</i>	- Upper row of high resolution block
in	<i>integer(i4) :: low_row_L1(:)</i>	- Lower row of high resolution block
in	<i>integer(i4) :: lef_col_L1(:)</i>	- Left column of high resolution block
in	<i>integer(i4) :: rig_col_L1(:)</i>	- Right column of high resolution block
in	<i>integer(i4) :: cell_id0 (:)</i>	- Cell ids at level 0
in	<i>logical :: mask0(:, :)</i>	- mask at level 0 field
in	<i>real(dp) :: nodata</i>	- nodata value
out	<i>real(dp) :: max_intercept1(:)</i>	- maximum canopy interception at Level-1

Author

Rohini Kumar

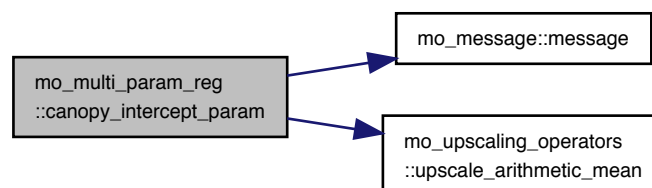
Date

Aug. 2013

References mo_message::message(), and mo_upscaling_operators::upscale_arithmetic_mean().

Referenced by mo_mhm::mhm().

Here is the call graph for this function:



```

subroutine mo_multi_param_reg::iper_thres_runoff (
    real(dp), dimension(1), intent(in) param,
    real(dp), dimension(:), intent(out) HL3 ) [private]

```

to be done by Kumar

Global parameters needed (see mhm parameter.nml):

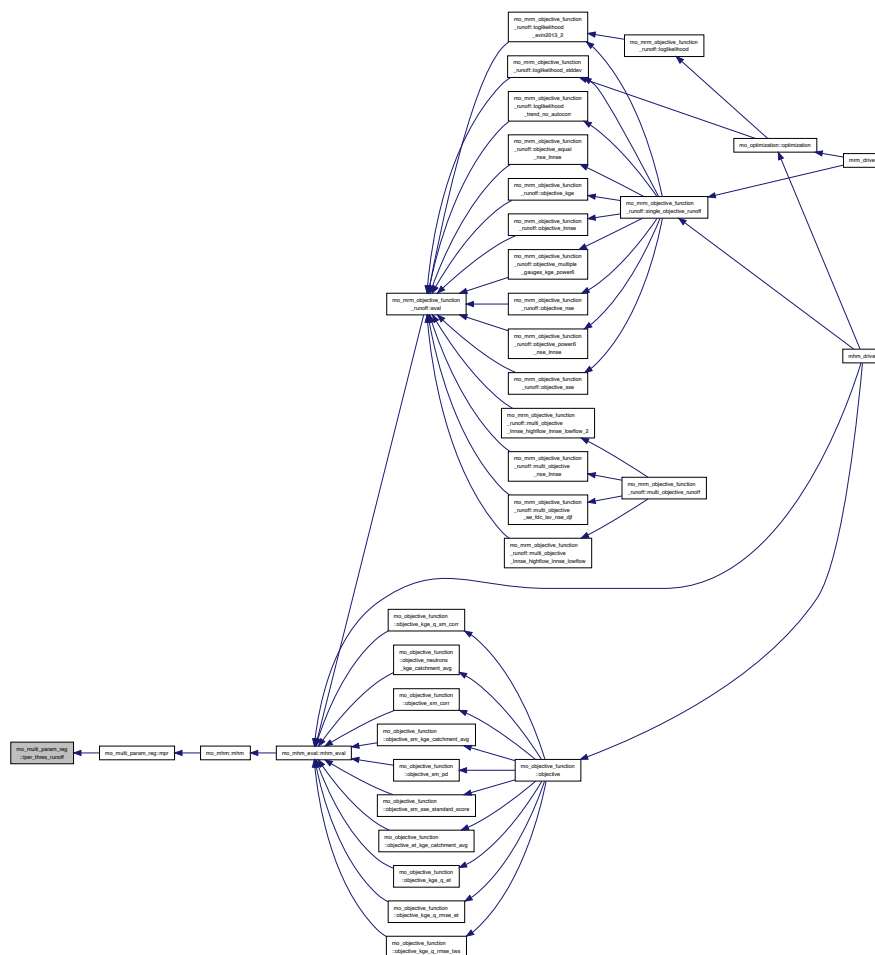
- Generated on December 1, 2017

in	$real(dp) :: param$	- given threshold parameter
out	$real(dp) ::$ $HL3(:)$	- distributed parameter field

Stephan Thober, Rohini Kumar

Dec 2012

Here is the caller graph for this function:



```
subroutine mo_multi_param_reg::karstic_layer (
    real(dp), dimension(3), intent(in) param,
```

```

integer(i4), dimension(:), intent(in) geoUnitKar,
integer(i4), dimension(:), intent(in) geoUnit0,
integer(i4), dimension(:), intent(in) geoUnitlist,
logical, dimension(:,:), intent(in) mask0,
real(dp), intent(in) nodata,
real(dp), dimension(:), intent(in) SMs_FC0,
real(dp), dimension(:), intent(in) KsVar_V0,
integer(i4), dimension(:), intent(in) cell_id0,
integer(i4), dimension(:), intent(in) nL0_in_L1,
integer(i4), dimension(:), intent(in) Upp_row_L1,
integer(i4), dimension(:), intent(in) Low_row_L1,
integer(i4), dimension(:), intent(in) Lef_col_L1,
integer(i4), dimension(:), intent(in) Rig_col_L1,
real(dp), intent(in) c2TSTu,
real(dp), dimension(:), intent(out) karstic_loss,
real(dp), dimension(:), intent(out) L1_Kp ) [private]

```

calculates the Karstic percolation loss

This subroutine calls first the karstic_fraction upscaling routine for determine the karstic fraction area for every Level 1 cell. Then, the karstic percolation loss is estimated given two shape parameters by

$$karstic_{loss} = 1 + (fKarArea * param(1)) * ((-1) ** INT(param(2), i4))$$

where $karstic_{loss}$ is the karstic percolation loss and $fKarArea$ ist the fraction of karstic area at level 1
Global parameters needed (see mhm_parameter.nml):

- param(1) = rechargeCoefficient
- param(2) = rechargeFactor_karstic
- param(3) = gain_loss_GWreservoir_karstic

Parameters

in	<i>integer(i4) :: nGeoUnits</i>	- number of geological formations
in	<i>integer(i4) :: geoUnitKar(:)</i>	- number of Karstic formation
in	<i>integer(i4) :: geoUnit0(:)</i>	- id of the Karstic formation
in	<i>logical :: mask0(:, :)</i>	- mask at level 0
in	<i>real(dp) :: nodata</i>	- given nodata value
in	<i>integer(i4) :: nL0_in_L1(:)</i>	- number of l0 cells within a l1 cell
in	<i>integer(i4) :: Upp_row_L1(:)</i>	- upper row of a l1 cell in l0 grid
in	<i>integer(i4) :: Low_row_L1(:)</i>	- lower row of a l1 cell in l0 grid
in	<i>integer(i4) :: Lef_col_L1(:)</i>	- left col of a l1 cell in l0 grid
in	<i>integer(i4) :: Rig_col_L1(:)</i>	- right col of a l1 cell in l0 grid
out	<i>real(dp) :: karstic_loss(:)</i>	[-] Karstic percolation loss
out	<i>real(dp) :: L1_Kp(:)</i>	[d-1] percolation coefficient

Author

Rohini Kumar, Stephan Thober

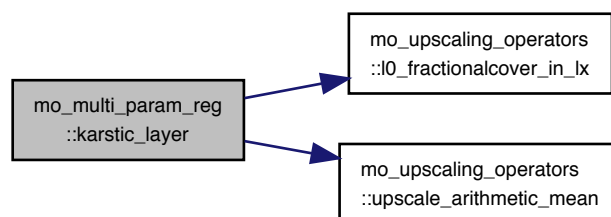
Date

Feb 2013

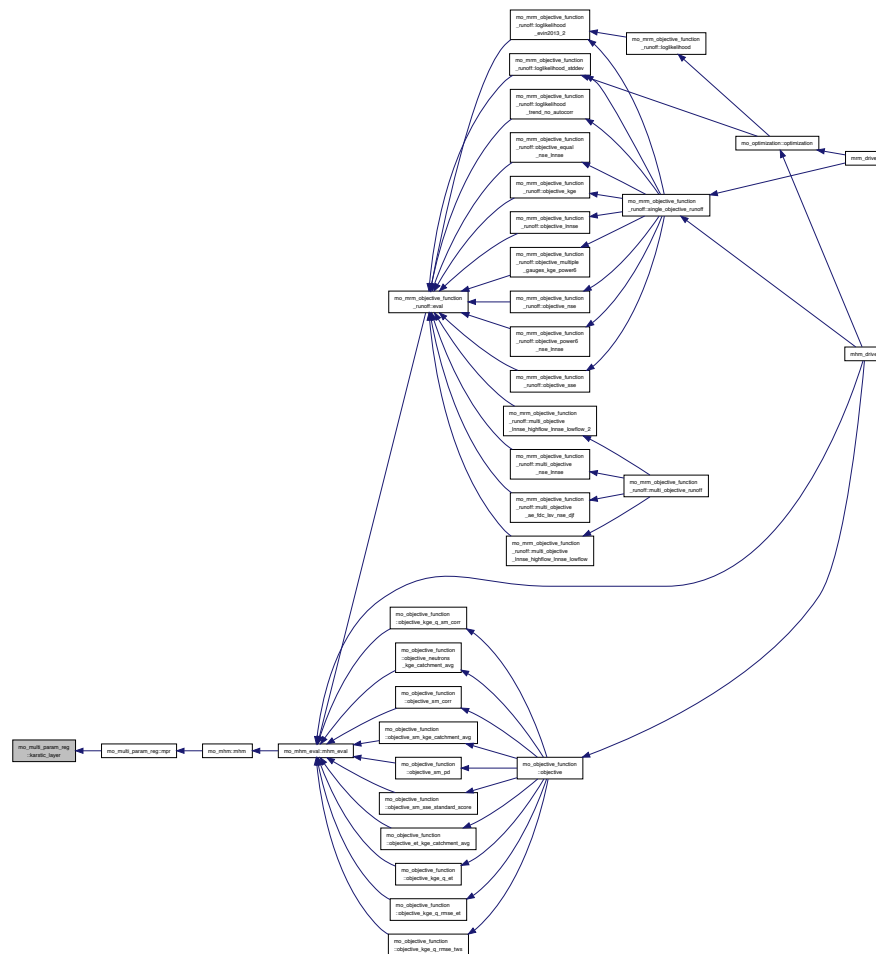
References `mo_kind::i4`, `mo_upscaling_operators::l0_fractionalcover_in_lx()`, and `mo_upscaling_operators::upscale_arithmetic_mean()`.

Referenced by `mpr()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.45.2.6 mpr()

```
subroutine, public mo_multi_param_reg::mpr (
  integer(i4), dimension(:,:), intent(in) proc_Mat,
  integer(i4), intent(in) iFlag_soil,
  real(dp), dimension(:), intent(in) param,
  real(dp), intent(in) nodata,
  logical, dimension(:,:), intent(in) mask0,
  integer(i4), dimension(:), intent(in) geoUnit0,
  integer(i4), dimension(:), intent(in) geoUnitList,
  integer(i4), dimension(:), intent(in) GeoUnitKar,
  real(dp), dimension(:,:), intent(in) LAILUT,
  integer(i4), dimension(:), intent(in) LAIUnitList,
  integer(i4), dimension(:), intent(in) SDB_is_present,
  integer(i4), dimension(:), intent(in) SDB_nHorizons,
  integer(i4), dimension(:), intent(in) SDB_nTillHorizons,
  real(dp), dimension(:,:), intent(in) SDB_sand,
  real(dp), dimension(:,:), intent(in) SDB_clay,
  real(dp), dimension(:,:), intent(in) SDB_DbM,
  real(dp), dimension(:,:,:), intent(in) SDB_Wd,
```

```

real(dp), dimension(:), intent(in) SDB_RZdepth,
integer(i4), intent(in) nHorizons_mhm,
real(dp), dimension(nhorizons_mhm), intent(in) horizon_depth,
real(dp), intent(in) c2TSTu,
real(dp), dimension(:), intent(in) fForest1,
real(dp), dimension(:), intent(in) fIperm1,
real(dp), dimension(:), intent(in) fPerm1,
integer(i4), dimension(:,:), intent(in) soilId0,
real(dp), dimension(:), intent(in) Asp0,
integer(i4), dimension(:), intent(in) LCover_LAI0,
integer(i4), dimension(:), intent(in) LCover0,
real(dp), dimension(:), intent(in) slope_emp0,
integer(i4), dimension(:), intent(in) cell_id0,
integer(i4), dimension(:), intent(in) upp_row_L1,
integer(i4), dimension(:), intent(in) low_row_L1,
integer(i4), dimension(:), intent(in) lef_col_L1,
integer(i4), dimension(:), intent(in) rig_col_L1,
integer(i4), dimension(:), intent(in) nL0_in_L1,
real(dp), dimension(:), intent(in) latitude,
real(dp), dimension(:), intent(inout) alphas,
real(dp), dimension(:), intent(inout) IDDP1,
real(dp), dimension(:), intent(inout) DDmax1,
real(dp), dimension(:), intent(inout) DD1,
real(dp), dimension(:), intent(inout) fAsp1,
real(dp), dimension(:), intent(inout) HarSamCoeff1,
real(dp), dimension(:,:), intent(inout) PrieTayAlpha1,
real(dp), dimension(:,:), intent(inout) aeroResist1,
real(dp), dimension(:,:), intent(inout) surfResist1,
real(dp), dimension(:,:), intent(inout) fRoots1,
real(dp), dimension(:), intent(inout) K0_1,
real(dp), dimension(:), intent(inout) K1_1,
real(dp), dimension(size(upp_row_L1,1)), intent(inout) K2_1,
real(dp), dimension(:), intent(inout) Kp1,
real(dp), dimension(:), intent(inout) karstic_loss,
real(dp), dimension(:,:), intent(inout) FC1,
real(dp), dimension(:,:), intent(inout) SMs1,
real(dp), dimension(:,:), intent(inout) beta1,
real(dp), dimension(:), intent(inout) jarvis_thresh_c1,
real(dp), dimension(:), intent(inout) TT1,
real(dp), dimension(:), intent(inout) HL1_1,
real(dp), dimension(:), intent(inout) HL3,
real(dp), dimension(:,:), intent(inout) PW1 )

```

Regionalizing and Upscaling process parameters.

calculating process parameters at L0 scale (Regionalization), like:

- Baseflow recession parameter
- Soil moisture parameters
- PET correction for aspect
and upscale these parameters to retrieve effective parameters at scale L1.
Further parameter regionalizations are done for:
- snow accumulation and melting parameters

- threshold parameter for runoff generation on impervious layer
- karstic percolation loss
- setting up the Regionalized Routing Parameters

Parameters

in	<i>integer(i4) :: tt</i>	- simulation time step
in	<i>integer(i4) :: newYear</i>	- new year
in	<i>integer(i4) :: LCyearId</i>	- mapping of landcover scenes
in	<i>integer(i4) :: proc_flag(:,:)</i>	- indicates which process shall be run the shape is number of processes times 3, the first column indicates which kind of process is run, 0 indicating do not run, the second column indicates the number of parameters required for this process and the third column indicates the position of the first parameter for this process in the array param
in	<i>real(dp) :: param(:,:)</i>	- given global parameter array
in	<i>integer(i4) :: iFlag_soil</i>	- flags for handling multiple soil databases
in	<i>real(dp) :: nodata</i>	- given nodata value
in	<i>integer(i4) :: geoUnit0(:,:)</i>	- geological units at Level 0
in	<i>integer(i4) :: geo_unit_list(:,:)</i>	- index list of geological units
in	<i>real(dp) :: LAIUT(:,:)</i>	- [1] Leaf area index for LAIUnit
in	<i>integer(i4) :: LAIUnitList(:,:)</i>	- [1] List of ids of each LAI class in LAIUT
in	<i>integer(i4) :: is_present(:,:)</i>	- indicates whether soiltype exists
in	<i>integer(i4) :: nHorizons(:,:)</i>	- Number of Horizons per soiltype
in	<i>integer(i4) :: nTillHorizons(:,:)</i>	- Number of Tillage Horizons
in	<i>real(dp) :: sand(:,:)</i>	- sand content
in	<i>real(dp) :: clay(:,:)</i>	- clay content
in	<i>real(dp) :: DbM(:,:)</i>	- mineral Bulk density
in	<i>real(dp) :: Wd(:,:)</i>	- weights of mHM horizons
in	<i>real(dp) :: RZdepth(:,:)</i>	- [mm] Total soil depth
in	<i>integer(i4) :: nHorizons_mHM</i>	- Number of horizons in mHM
in	<i>integer(i4) :: horizon_depth(:,:)</i>	- depth of each horizon
in	<i>real(dp) :: c2TSTu</i>	- unit transformation coefficient
in	<i>real(dp) :: fForest1(:,:)</i>	- fraction of forest cover at scale L1
in	<i>real(dp) :: flperm1(:,:)</i>	- fraction of sealed area at scale L1
in	<i>real(dp) :: fPerm1(:,:)</i>	- fraction of permeable area at scale L1
in	<i>integer(i4) :: soilID0(:,:)</i>	- [1] soil IDs at level 0
in	<i>real(dp) :: Asp0(:,:)</i>	- [degree] Aspect at Level 0
in	<i>real(dp) :: LCover_LAI0(:,:)</i>	- [1] land cover ID for LAI estimation
in	<i>integer(i4) :: LCover0(:,:)</i>	- [1] land use cover at level 0
in	<i>real(dp) :: length(:,:)</i>	- [m] total length
in	<i>real(dp) :: slope(:,:)</i>	- average slope
in	<i>real(dp) :: fFPimp(:,:)</i>	- fraction of the flood plain with impervious layer
in	<i>real(dp) :: TS</i>	- [h] time step in
in	<i>integer(i4) :: cell_id0(:,:)</i>	- cell ids of high resolution field, Number of rows times Number of columns of high resolution field

Parameters

in	<i>integer(i4) :: upp_row_L1(:)</i>	- Upper row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: low_row_L1(:)</i>	- Lower row id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: lef_col_L1(:)</i>	- Left column id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: rig_col_L1(:)</i>	- Right column id in high resolution field (L0) of low resolution cell (L1 cell)
in	<i>integer(i4) :: nL0_in_L1(:)</i>	- Number of high resolution cells (L0) in low resolution cell (L1 cell)
in, out	<i>real(dp) :: k2_1(:, :)</i>	- baseflow recession parameter at L1
in, out	<i>real(dp) :: KsVar_H0(:, :)</i>	- relative variability of saturated hydraulic cond. for Horizontal flow
in, out	<i>real(dp) :: KsVar_V0(:, :)</i>	- relative variability of saturated hydraulic cond. for Vertical flow
in, out	<i>real(dp) :: SMs_tot0(:, :)</i>	- total saturated soil moisture content
in, out	<i>real(dp) :: SMs_FC0(:, :)</i>	- soil moisture deficit from field cap. w.r.t to saturation
in, out	<i>real(dp) :: beta1(:, :)</i>	- Parameter that determines the relative contribution to SM, upscaled Bulk density. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: SMs1(:, :)</i>	- [10 ⁻³ m] depth of saturated SM cont Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: FC1(:, :)</i>	- [10 ⁻³ m] field capacity. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: PW1(:, :)</i>	- [10 ⁻³ m] permanent wilting point. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: jarvis_thresh_c1(:),-</i>	[1] jarvis critical value for normalized soil water content
in, out	<i>real(dp) :: fRoots1(:, :)</i>	- fraction of roots in soil horizons. Number of cells at L1 times number of horizons in mHM
in, out	<i>real(dp) :: TT1(:)</i>	- threshold temperature for snow rain
in, out	<i>real(dp) :: DD1(:)</i>	- Degree-day factor
in, out	<i>real(dp) :: DDmax1(:)</i>	- Maximum Degree-day factor
in, out	<i>real(dp) :: IDDP1(:)</i>	- increase of the degree-day factor per mm of increase in precipitation
in, out	<i>real(dp) :: fAsp1(:)</i>	- [1] PET correction for Aspect at level 1
in, out	<i>real(dp) :: HarSamCoeff1(:)</i>	- [1] PET Hargreaves Samani coefficient at level 1
in, out	<i>real(dp) :: PrieTayAlpha1(:, :)</i>	- [1] PET Priestley Taylor coefficient at level 1
in, out	<i>real(dp) :: aeroResist1(:, :)</i>	- [s m-1] PET aerodynamical resistance at level 1
in, out	<i>real(dp) :: surfResist1(:, :)</i>	- [s m-1] PET bulk surface resistance at level 1
in, out	<i>real(dp) :: HL3(:)</i>	- threshold parameter for runoff generation on impervious layer
in, out	<i>real(dp) :: K(:)</i>	- [d] Muskingum travel time parameters
in, out	<i>real(dp) :: xi(:)</i>	- [1] Muskingum diffusion parameter (attenuation)
in, out	<i>real(dp) :: C1(:)</i>	- routing parameter C1 (Chow, 25-41)
in, out	<i>real(dp) :: C2(:)</i>	- routing parameter C2 (")

Author

Stephan Thober, Rohini Kumar

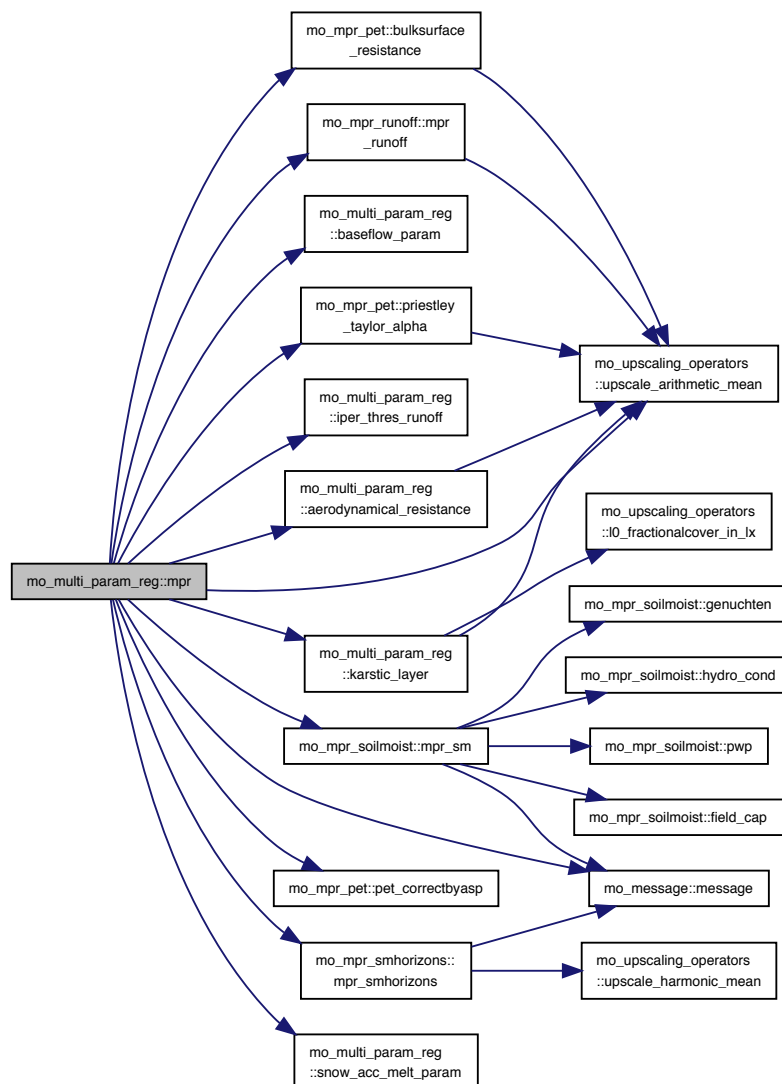
Date

Dec 2012

References `aerodynamical_resistance()`, `baseflow_param()`, `mo_mpr_pet::bulksurface_resistance()`, `mo_kind::i4`, `iper_thres_runoff()`, `karstic_layer()`, `mo_message::message()`, `mo_mpr_runoff::mpr_runoff()`, `mo_mpr_soilmoist::mpr_sm()`, `mo_mpr_smhorizons::mpr_smhorizons()`, `mo_mpr_pet::pet_correctbyasp()`, `mo_mpr_pet::priestley_taylor_alpha()`, `snow_acc_melt_param()`, and `mo_upscaling_operators::upscale_arithmetic_mean()`.

Referenced by `mo_mhm::mhm()`.

Here is the call graph for this function:



[illegible]

```

subroutine mo_multi_param_reg::snow_acc_melt_param (
    real(dp), dimension(8), intent(in)  param,
    real(dp), intent(in)  c2TSTu,
    real(dp), dimension(:), intent(in)  fForest1,
    real(dp), dimension(:), intent(in)  fIperm1,
    real(dp), dimension(:), intent(in)  fPerm1,
    real(dp), dimension(:), intent(out) TT1,
    real(dp), dimension(:), intent(out) DD1,
    real(dp), dimension(:), intent(out) IDDP1,
    real(dp), dimension(:), intent(out) DDmax1 ) [private]

```

This subroutine calculates the snow parameters threshold temperature (TT), degree-day factor without precipitation (DD) and maximum degree-day factor (DDmax) as well as increase of degree-day factor per mm of increase in

precipitation (IDDP).

Global parameters needed (see mhm_parameter.nml):

- param(1) = snowTreshholdTemperature
- param(2) = degreeDayFactor_forest
- param(3) = degreeDayFactor_impervious
- param(4) = degreeDayFactor_pervious
- param(5) = increaseDegreeDayFactorByPrecip
- param(6) = maxDegreeDayFactor_forest
- param(7) = maxDegreeDayFactor_impervious
- param(8) = maxDegreeDayFactor_pervious
INTENT(IN)

Parameters

in	<i>real(dp) :: param(8)</i>	- There are eight snow parameters required
in	<i>real(dp) :: c2TSTu</i>	- unit transformation coefficient
in	<i>real(dp) :: fForest1(:)</i>	- fraction of forest cover at scale L1
in	<i>real(dp) :: flperm1(:)</i>	- fraction of sealed area at scale L1
in	<i>real(dp) :: fPerm1(:)</i>	- fraction of permeable area at scale L1
out	<i>real(dp) :: TT1(:)</i>	- threshold temperature for snow rain
out	<i>real(dp) :: DD1(:)</i>	- Degree-day factor
out	<i>real(dp) :: DDmax1(:)</i>	- Maximum Degree-day factor
out	<i>real(dp) :: IDDP1(:)</i>	- increase of the degree-day factor per mm of increase in precipitation

Author

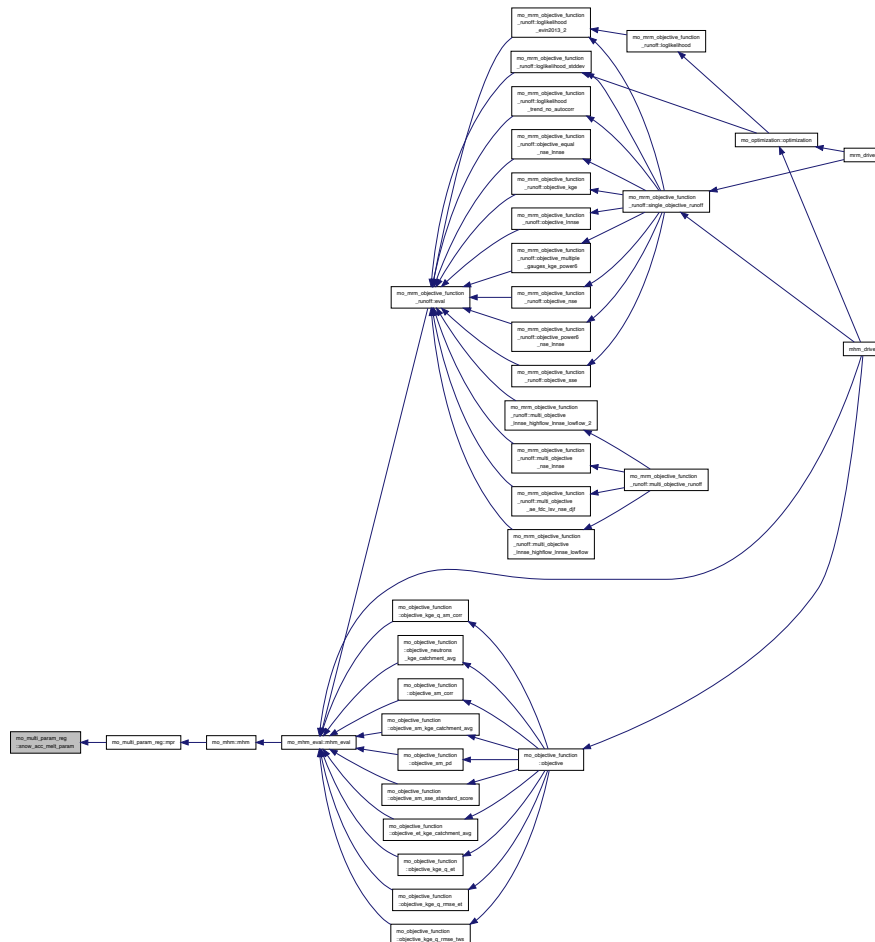
Stephan Thober, Rohini Kumar

Date

Dec 2012

Referenced by mpr().

Here is the caller graph for this function:



15.46 mo_ncread Module Reference

Data Types

- interface [get_ncvar](#)

Functions/Subroutines

- integer(i4) function, dimension(5), public [get_ncdim](#) (Filename, Variable, PrintInfo, ndims)
- subroutine, public [get_ncdimatt](#) (Filename, Variable, DimName, DimLen)
- subroutine, public [get_ncvaratt](#) (FileName, VarName, AttName, AttValues, fid, dtype)
- subroutine [get_ncvar_0d_sp](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_0d_dp](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_1d_dp](#) (Filename, VarName, Dat, start, a_count, fid)

- subroutine [get_ncvar_2d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_0d_i4](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_0d_i1](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- integer(i4) function, public [ncopen](#) (Fname)
- subroutine, public [ncclose](#) (ncid)
- subroutine [get_info](#) (Varname, ncid, varid, xtype, dl, Info, ndims)
- subroutine [check](#) (status)

15.46.1 Function/Subroutine Documentation

15.46.1.1 [check\(\)](#)

```
subroutine mo_ncread::check (
    integer(i4), intent(in) status ) [private]
```

Referenced by [get_info\(\)](#), [get_ncdim\(\)](#), [get_ncdimatt\(\)](#), [get_ncvar_0d_dp\(\)](#), [get_ncvar_0d_i1\(\)](#), [get_ncvar_0d_i4\(\)](#), [get_ncvar_0d_sp\(\)](#), [get_ncvar_1d_dp\(\)](#), [get_ncvar_1d_i1\(\)](#), [get_ncvar_1d_i4\(\)](#), [get_ncvar_1d_sp\(\)](#), [get_ncvar_2d_dp\(\)](#), [get_ncvar_2d_i1\(\)](#), [get_ncvar_2d_i4\(\)](#), [get_ncvar_2d_sp\(\)](#), [get_ncvar_3d_dp\(\)](#), [get_ncvar_3d_i1\(\)](#), [get_ncvar_3d_i4\(\)](#), [get_ncvar_3d_sp\(\)](#), [get_ncvar_4d_dp\(\)](#), [get_ncvar_4d_i1\(\)](#), [get_ncvar_4d_i4\(\)](#), [get_ncvar_4d_sp\(\)](#), [get_ncvar_5d_dp\(\)](#), [get_ncvar_5d_i1\(\)](#), [get_ncvar_5d_i4\(\)](#), [get_ncvar_5d_sp\(\)](#), [get_ncvaratt\(\)](#), [ncclose\(\)](#), and [ncopen\(\)](#).

15.46.1.2 get_info()

```

subroutine mo_ncread::get_info (
    character(len=*), intent(in) Varname,
    integer(i4), intent(in) ncid,
    integer(i4), intent(out) varid,
    integer(i4), intent(out) xtype,
    integer(i4), dimension(:), intent(inout), optional dl,

```

```
logical, intent(in), optional Info,
integer(i4), intent(out), optional ndims ) [private]
```

References check().

Referenced by `get_ncdim()`, `get_ncdimatt()`, `get_ncvar_0d_dp()`, `get_ncvar_0d_i1()`, `get_ncvar_0d_i4()`, `get_ncvar_0d_sp()`, `get_ncvar_1d_dp()`, `get_ncvar_1d_i1()`, `get_ncvar_1d_i4()`, `get_ncvar_1d_sp()`, `get_ncvar_2d_dp()`, `get_ncvar_2d_i1()`, `get_ncvar_2d_i4()`, `get_ncvar_2d_sp()`, `get_ncvar_3d_dp()`, `get_ncvar_3d_i1()`, `get_ncvar_3d_i4()`, `get_ncvar_3d_sp()`, `get_ncvar_4d_dp()`, `get_ncvar_4d_i1()`, `get_ncvar_4d_i4()`, `get_ncvar_4d_sp()`, `get_ncvar_5d_dp()`, `get_ncvar_5d_i1()`, `get_ncvar_5d_i4()`, and `get_ncvar_5d_sp()`.

Here is the call graph for this function:

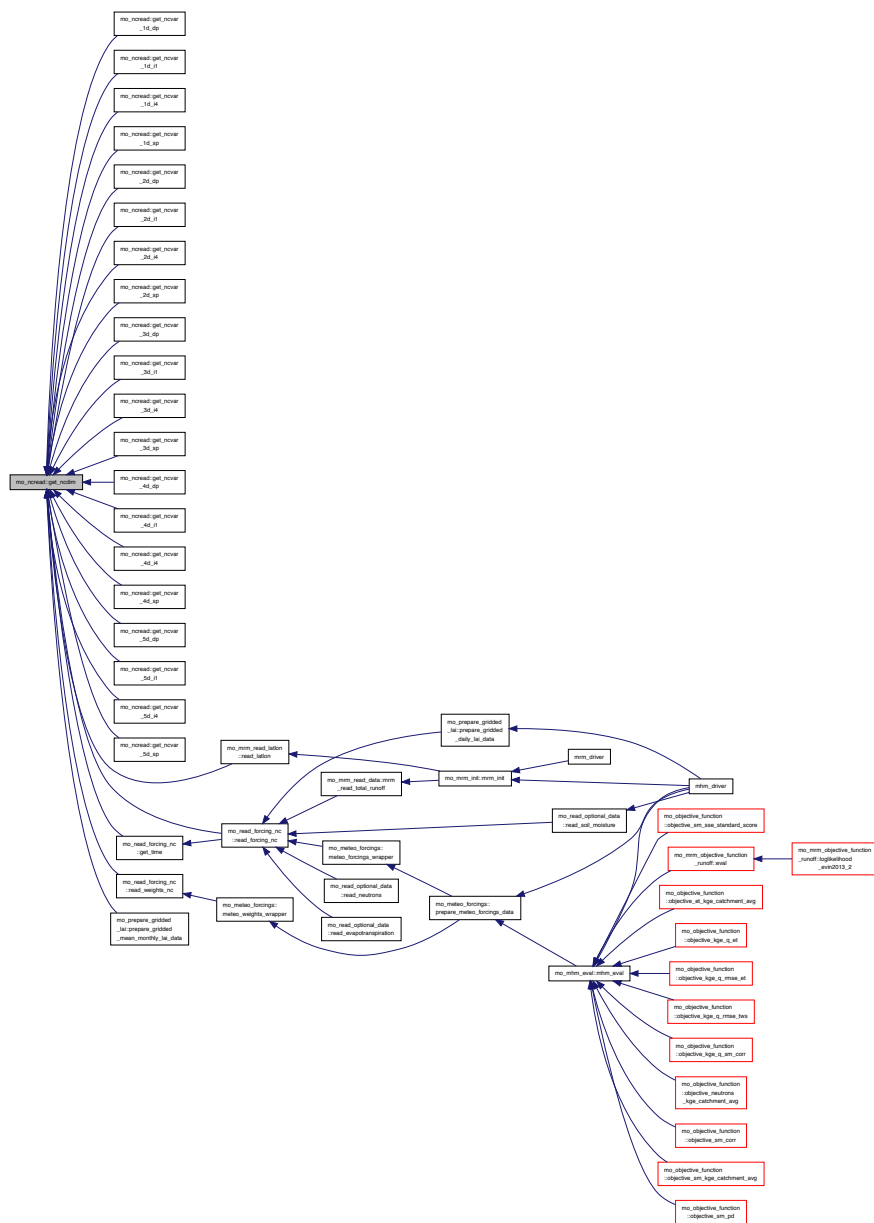


[illegible]

```
integer(i4) function, dimension(5), public mo_ncread::get_ncdim (
    character(len=*) , intent(in) Filename,
    character(len=*) , intent(in) Variable,
    logical, intent(in), optional PrintInfo,
    integer(i4), intent(out), optional ndims )
```

Referenced by `get_ncvar_1d_dp()`, `get_ncvar_1d_i1()`, `get_ncvar_1d_i4()`, `get_ncvar_1d_sp()`, `get_ncvar_2d_dp()`, `get_ncvar_2d_i1()`, `get_ncvar_2d_i4()`, `get_ncvar_2d_sp()`, `get_ncvar_3d_dp()`, `get_ncvar_3d_i1()`, `get_ncvar_3d_i4()`, `get_ncvar_3d_sp()`, `get_ncvar_4d_dp()`, `get_ncvar_4d_i1()`, `get_ncvar_4d_i4()`, `get_ncvar_4d_sp()`, `get_ncvar_5d_dp()`, `get_ncvar_5d_i1()`, `get_ncvar_5d_i4()`, `get_ncvar_5d_sp()`, `mo_read_forcing_nc::get_time()`, `mo_read_forcing_nc::prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data()`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_read_forcing_nc::mrm_read_latlon::read_latlon()`, and `mo_read_forcing_nc::read_weights_nc()`.

Here is the caller graph for this function:



15.46.1.4 get_ncdimatt()

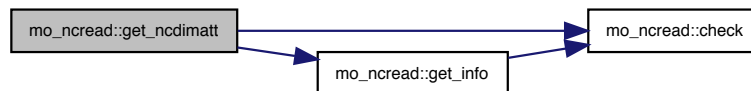
```

subroutine, public mo_ncread::get_ncdimatt (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) Variable,
    character(len=*), dimension(:), intent(out), allocatable DimName,
    integer(i4), dimension(:), intent(out), optional, allocatable DimLen )

```

References check(), and get_info().

Here is the call graph for this function:



15.46.1.5 get_ncvar_0d_dp()

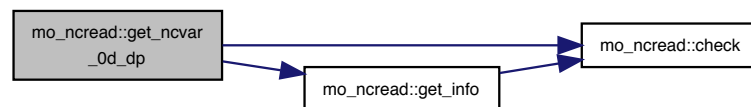
```

subroutine mo_ncread::get_ncvar_0d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), intent(inout) Dat,
    integer(i4), intent(in), optional fid ) [private]

```

References check(), and get_info().

Here is the call graph for this function:



15.46.1.6 get_ncvar_0d_i1()

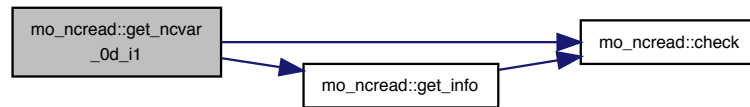
```

subroutine mo_ncread::get_ncvar_0d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), intent(inout) Dat,
    integer(i4), intent(in), optional fid ) [private]

```

References check(), and get_info().

Here is the call graph for this function:



15.46.1.7 `get_ncvar_0d_i4()`

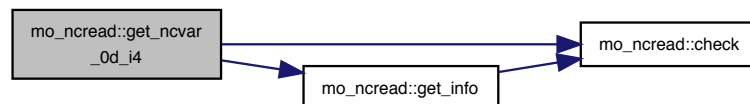
```

subroutine mo_ncread::get_ncvar_0d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), intent(inout) Dat,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, and `get_info()`.

Here is the call graph for this function:



15.46.1.8 `get_ncvar_0d_sp()`

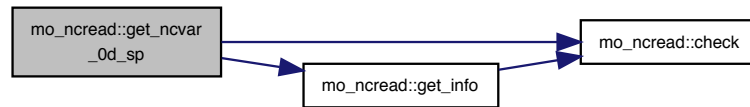
```

subroutine mo_ncread::get_ncvar_0d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), intent(inout) Dat,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, and `get_info()`.

Here is the call graph for this function:



15.46.1.9 get_ncvar_1d_dp()

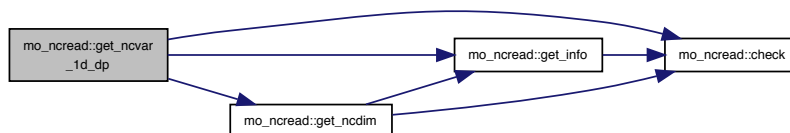
```

subroutine mo_ncread::get_ncvar_1d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.10 get_ncvar_1d_i1()

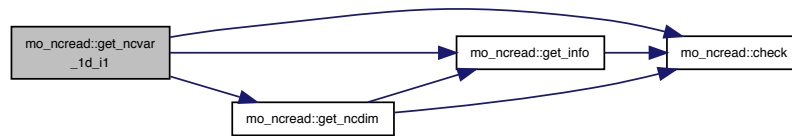
```

subroutine mo_ncread::get_ncvar_1d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.11 `get_ncvar_1d_i4()`

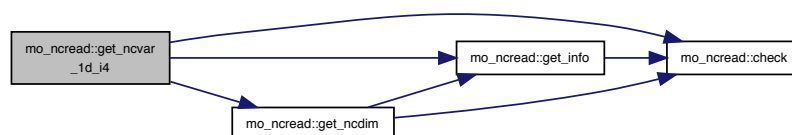
```

subroutine mo_ncread::get_ncvar_1d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.12 `get_ncvar_1d_sp()`

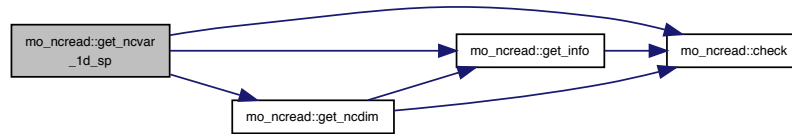
```

subroutine mo_ncread::get_ncvar_1d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.13 get_ncvar_2d_dp()

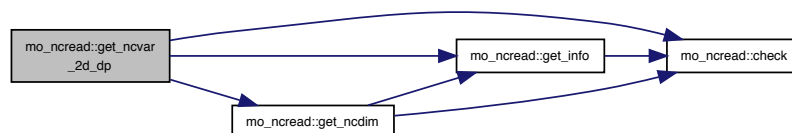
```

subroutine mo_ncread::get_ncvar_2d_dp (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  real(dp), dimension(:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.14 get_ncvar_2d_i1()

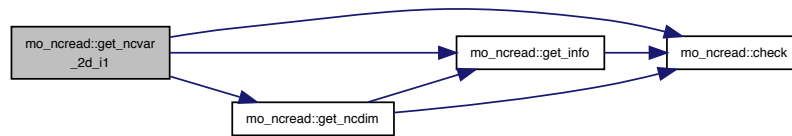
```

subroutine mo_ncread::get_ncvar_2d_i1 (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  integer(1), dimension(:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.15 `get_ncvar_2d_i4()`

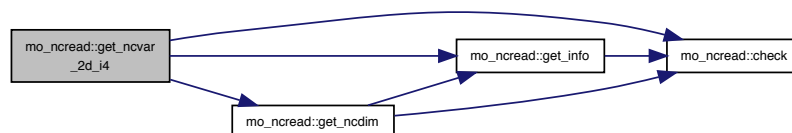
```

subroutine mo_ncread::get_ncvar_2d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.16 `get_ncvar_2d_sp()`

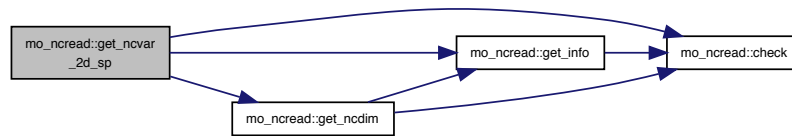
```

subroutine mo_ncread::get_ncvar_2d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.17 get_ncvar_3d_dp()

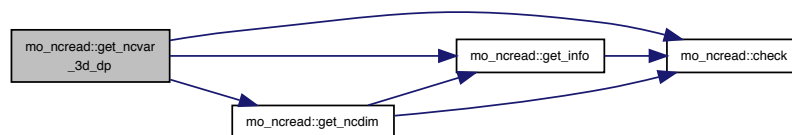
```

subroutine mo_ncread::get_ncvar_3d_dp (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  real(dp), dimension(:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.18 get_ncvar_3d_i1()

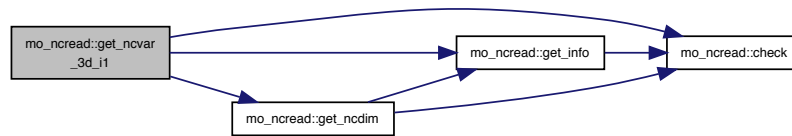
```

subroutine mo_ncread::get_ncvar_3d_i1 (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  integer(1), dimension(:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.19 `get_ncvar_3d_i4()`

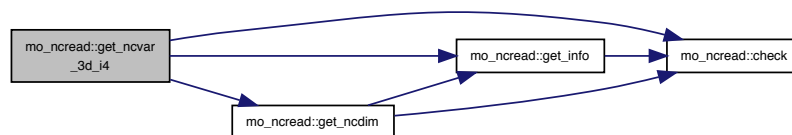
```

subroutine mo_ncread::get_ncvar_3d_i4 (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  integer(i4), dimension(:, :, :), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.20 `get_ncvar_3d_sp()`

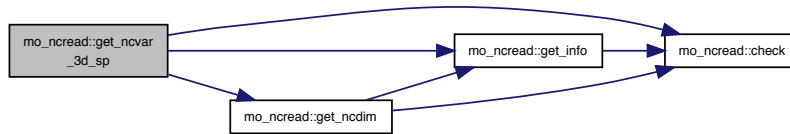
```

subroutine mo_ncread::get_ncvar_3d_sp (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  real(sp), dimension(:, :, :), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.21 get_ncvar_4d_dp()

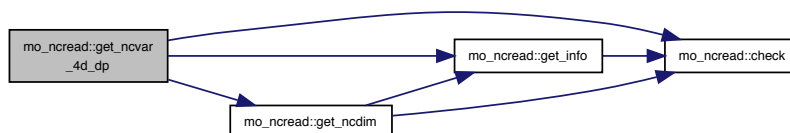
```

subroutine mo_ncread::get_ncvar_4d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.22 get_ncvar_4d_i1()

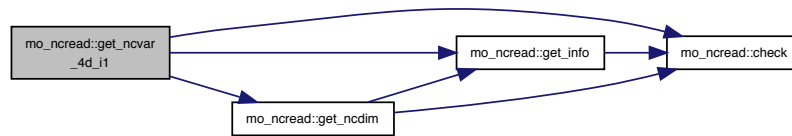
```

subroutine mo_ncread::get_ncvar_4d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.23 `get_ncvar_4d_i4()`

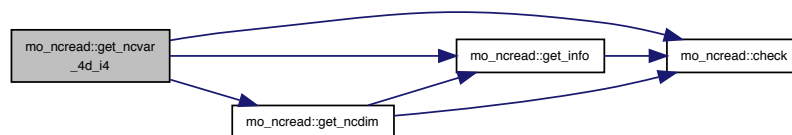
```

subroutine mo_ncread::get_ncvar_4d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.24 `get_ncvar_4d_sp()`

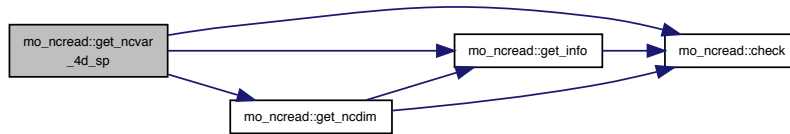
```

subroutine mo_ncread::get_ncvar_4d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.25 get_ncvar_5d_dp()

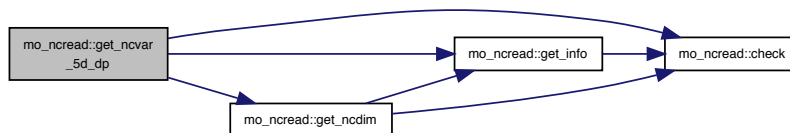
```

subroutine mo_ncread::get_ncvar_5d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:,:,:,,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.26 get_ncvar_5d_i1()

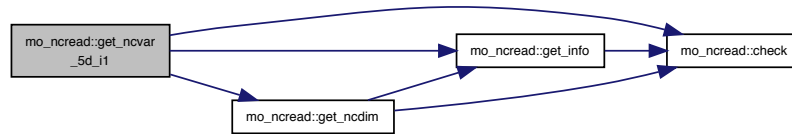
```

subroutine mo_ncread::get_ncvar_5d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:,:,:,,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.27 `get_ncvar_5d_i4()`

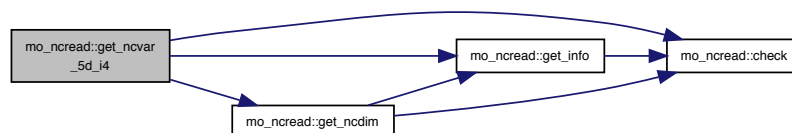
```

subroutine mo_ncread::get_ncvar_5d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:,:,:,,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.28 `get_ncvar_5d_sp()`

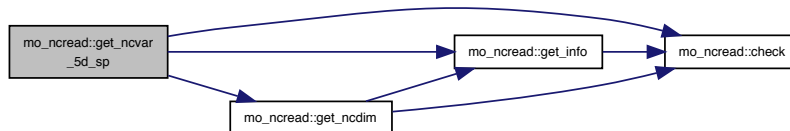
```

subroutine mo_ncread::get_ncvar_5d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:,:,:,,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid ) [private]

```

References `check()`, `get_info()`, and `get_ncdim()`.

Here is the call graph for this function:



15.46.1.29 get_ncvaratt()

```

subroutine, public mo_ncread::get_ncvaratt (
    character(len=*), intent(in) FileName,
    character(len=*), intent(in) VarName,
    character(len=*), intent(in) AttName,
    character(len=*), intent(out) AttValues,
    integer(i4), intent(in), optional fid,
    integer(i4), intent(out), optional dtype )

```

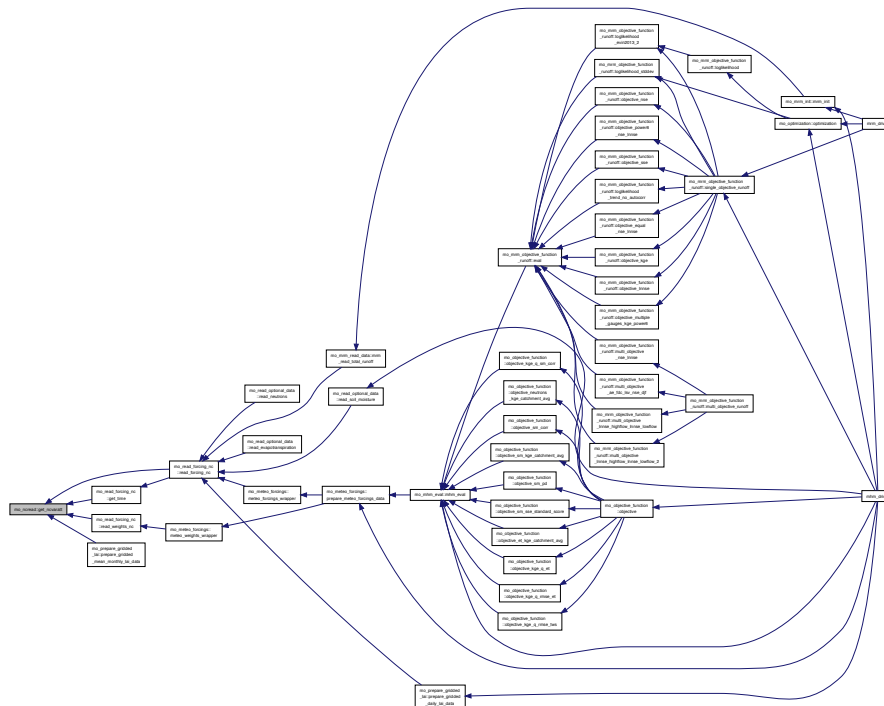
References `check()`.

Referenced by `mo_read_forcing_nc::get_time()`, `mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai` ↔ `data()`, `mo_read_forcing_nc::read_forcing_nc()`, and `mo_read_forcing_nc::read_weights_nc()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.46.1.30 nccclose()

```
subroutine, public mo_nccread::nccclose (
    integer(i4), intent(in) ncid )
```

References check().

Here is the call graph for this function:



15.46.1.31 ncopen()

```
integer(i4) function, public mo_nccread::ncopen (
    character(len=*), intent(in) Fname )
```

References check().

Here is the call graph for this function:



15.47 mo_ncwrite Module Reference

Data Types

- type [attribute](#)
- type [dims](#)
- interface [dump_netcdf](#)
- interface [var2nc](#)

Extended [dump_netcdf](#) for multiple variables.

- type [variable](#)

Functions/Subroutines

- subroutine, public [close_netcdf](#) (ncId)
- subroutine, public [create_netcdf](#) (Filename, ncId, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_1d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_1d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_1d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [var2nc_1d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncId, nrec)
- subroutine [var2nc_1d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncId, nrec)
- subroutine [var2nc_1d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncId, nrec)
- subroutine [var2nc_2d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncId, nrec)
- subroutine [var2nc_2d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncId, nrec)

- subroutine [var2nc_2d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine, public [write_dynamic_netcdf](#) (ncid, irec)
- subroutine, public [write_static_netcdf](#) (ncid)
- integer(i4) function [open_netcdf](#) (f_name, create)
- subroutine [check](#) (status)

Variables

- integer(i4), parameter, public [nmaxdim](#) = 5
- integer(i4), parameter, public [nmaxatt](#) = 20
- integer(i4), parameter, public [maxlen](#) = 256
- integer(i4), parameter, public [ngatt](#) = 20
- integer(i4), parameter, public [nattdim](#) = 2
- integer(i4), public [nvars](#)
- integer(i4), public [ndims](#)
- type([dims](#)), dimension(:), allocatable, public [dnc](#)
- type([variable](#)), dimension(:), allocatable, public [v](#)
- type([attribute](#)), dimension([ngatt](#)), public [gatt](#)

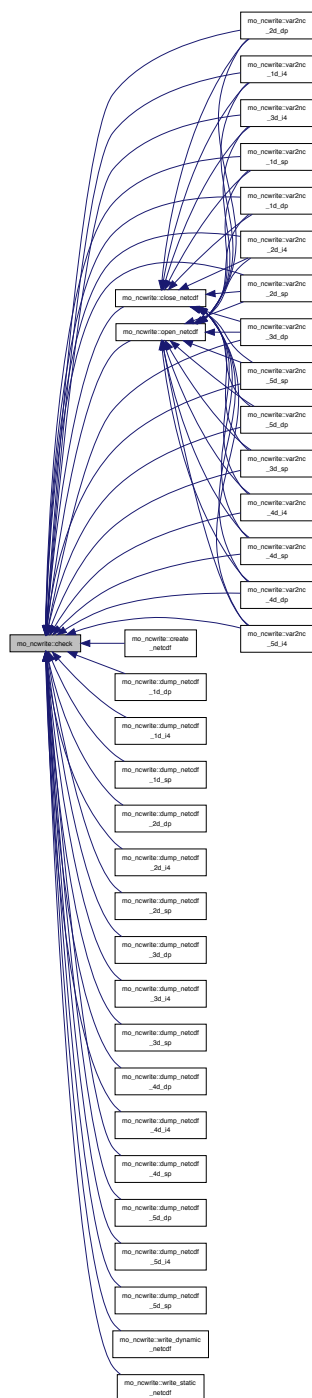
15.47.1 Function/Subroutine Documentation

15.47.1.1 [check\(\)](#)

```
subroutine mo_ncwrite::check (
    integer(i4), intent(in) status ) [private]
```

Referenced by [close_netcdf\(\)](#), [create_netcdf\(\)](#), [dump_netcdf_1d_dp\(\)](#), [dump_netcdf_1d_i4\(\)](#), [dump_netcdf_1d_sp\(\)](#), [dump_netcdf_2d_dp\(\)](#), [dump_netcdf_2d_i4\(\)](#), [dump_netcdf_2d_sp\(\)](#), [dump_netcdf_3d_dp\(\)](#), [dump_netcdf_3d_i4\(\)](#), [dump_netcdf_3d_sp\(\)](#), [dump_netcdf_4d_dp\(\)](#), [dump_netcdf_4d_i4\(\)](#), [dump_netcdf_4d_sp\(\)](#), [dump_netcdf_5d_dp\(\)](#), [dump_netcdf_5d_i4\(\)](#), [dump_netcdf_5d_sp\(\)](#), [open_netcdf\(\)](#), [var2nc_1d_dp\(\)](#), [var2nc_1d_i4\(\)](#), [var2nc_1d_sp\(\)](#), [var2nc_2d_dp\(\)](#), [var2nc_2d_i4\(\)](#), [var2nc_2d_sp\(\)](#), [var2nc_3d_dp\(\)](#), [var2nc_3d_i4\(\)](#), [var2nc_3d_sp\(\)](#), [var2nc_4d_dp\(\)](#), [var2nc_4d_i4\(\)](#), [var2nc_4d_sp\(\)](#), [var2nc_5d_dp\(\)](#), [var2nc_5d_i4\(\)](#), [var2nc_5d_sp\(\)](#), [write_dynamic_netcdf\(\)](#), and [write_static_netcdf\(\)](#).

Here is the caller graph for this function:



15.47.1.2 close_netcdf()

```
subroutine, public mo_ncwrite::close_netcdf (
    integer(i4), intent(in) ncId )
```

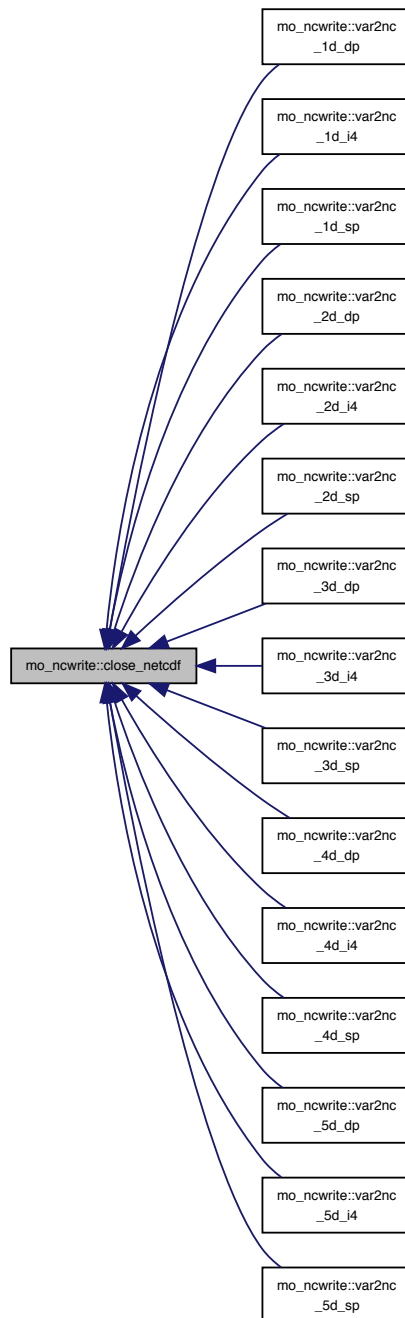
References check().

Referenced by `var2nc_1d_dp()`, `var2nc_1d_i4()`, `var2nc_1d_sp()`, `var2nc_2d_dp()`, `var2nc_2d_i4()`, `var2nc_2d_sp()`, `var2nc_3d_dp()`, `var2nc_3d_i4()`, `var2nc_3d_sp()`, `var2nc_4d_dp()`, `var2nc_4d_i4()`, `var2nc_4d_sp()`, `var2nc_5d_dp()`, `var2nc_5d_i4()`, and `var2nc_5d_sp()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.47.1.3 create_netcdf()

```

subroutine, public mo_ncwrite::create_netcdf (
    character(len=*), intent(in) Filename,
    integer(i4), intent(out) ncid,

```

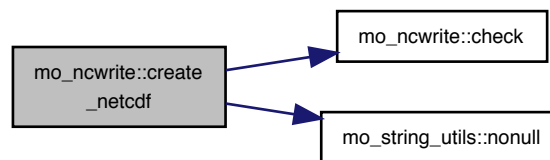
```

logical, intent(in), optional lfs,
logical, intent(in), optional netcdf4,
integer(i4), intent(in), optional deflate_level )

```

References `check()`, `dnc`, `gatt`, `ndims`, `ngatt`, `mo_string_utils::nonnull()`, `nvars`, and `v`.

Here is the call graph for this function:



15.47.1.4 dump_netcdf_1d_dp()

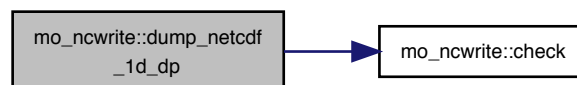
```

subroutine mo_ncwrite::dump_netcdf_1d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]

```

References `check()`, and `ndims`.

Here is the call graph for this function:



15.47.1.5 dump_netcdf_1d_i4()

```

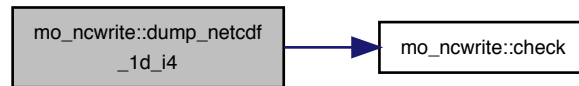
subroutine mo_ncwrite::dump_netcdf_1d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,

```

```
logical, intent(in), optional netcdf4,
integer(i4), intent(in), optional deflate_level ) [private]
```

References `check()`, and `ndims`.

Here is the call graph for this function:



15.47.1.6 dump_netcdf_1d_sp()

```
subroutine mo_ncwrite::dump_netcdf_1d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
```

References `check()`, and `ndims`.

Here is the call graph for this function:

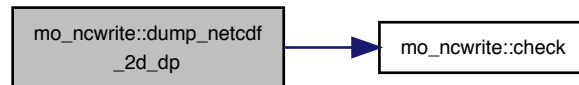


15.47.1.7 dump_netcdf_2d_dp()

```
subroutine mo_ncwrite::dump_netcdf_2d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:, :), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
```

References `check()`, and `ndims`.

Here is the call graph for this function:



15.47.1.8 dump_netcdf_2d_i4()

```

subroutine mo_ncwrite::dump_netcdf_2d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



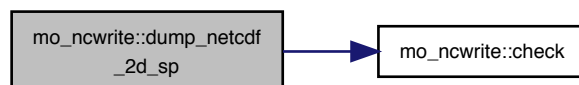
15.47.1.9 dump_netcdf_2d_sp()

```

subroutine mo_ncwrite::dump_netcdf_2d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



15.47.1.10 dump_netcdf_3d_dp()

```

subroutine mo_ncwrite::dump_netcdf_3d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



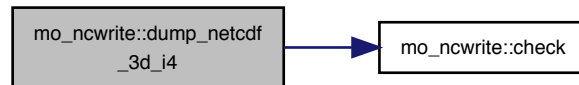
15.47.1.11 dump_netcdf_3d_i4()

```

subroutine mo_ncwrite::dump_netcdf_3d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



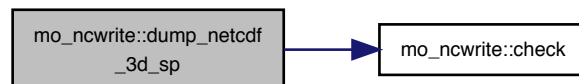
15.47.1.12 dump_netcdf_3d_sp()

```

subroutine mo_ncwrite::dump_netcdf_3d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References check(), and ndims.

Here is the call graph for this function:



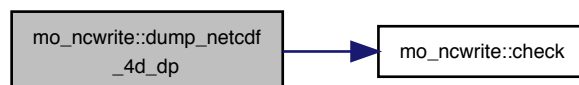
15.47.1.13 dump_netcdf_4d_dp()

```

subroutine mo_ncwrite::dump_netcdf_4d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:,:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References check(), and ndims.

Here is the call graph for this function:



15.47.1.14 dump_netcdf_4d_i4()

```

subroutine mo_ncwrite::dump_netcdf_4d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References check(), and ndims.

Here is the call graph for this function:



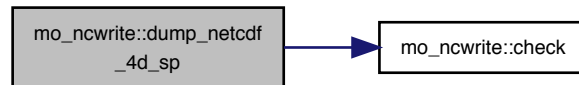
15.47.1.15 dump_netcdf_4d_sp()

```

subroutine mo_ncwrite::dump_netcdf_4d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:,:,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References check(), and ndims.

Here is the call graph for this function:



15.47.1.16 dump_netcdf_5d_dp()

```

subroutine mo_ncwrite::dump_netcdf_5d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



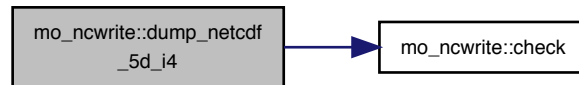
15.47.1.17 dump_netcdf_5d_i4()

```

subroutine mo_ncwrite::dump_netcdf_5d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References `check()`, and `ndims`.

Here is the call graph for this function:



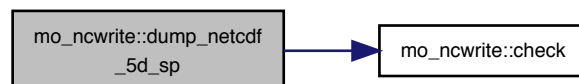
15.47.1.18 dump_netcdf_5d_sp()

```

subroutine mo_ncwrite::dump_netcdf_5d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level ) [private]
  
```

References check(), and ndims.

Here is the call graph for this function:



15.47.1.19 open_netcdf()

```

integer(i4) function mo_ncwrite::open_netcdf (
    character(len=*), intent(in) f_name,
    logical, intent(in) create ) [private]
  
```

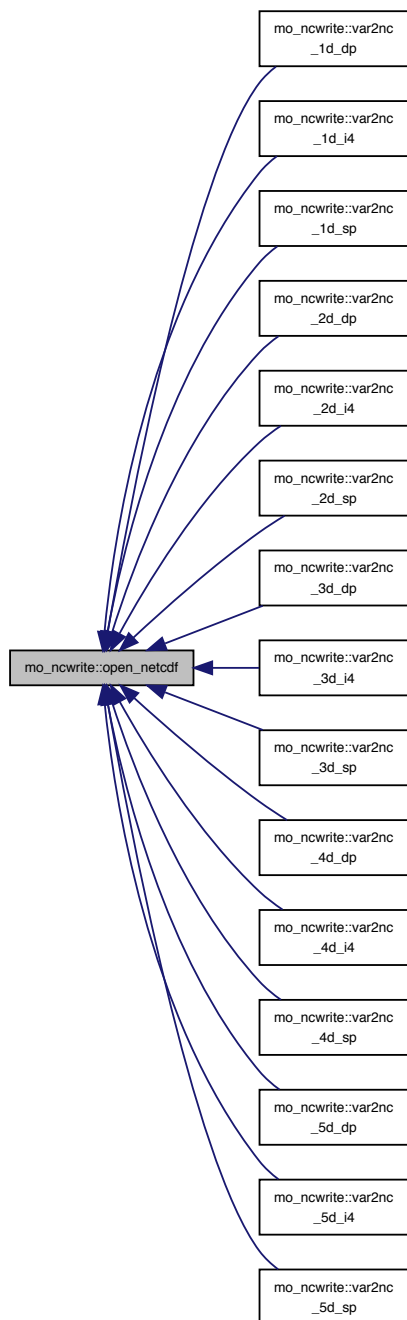
References check().

Referenced by var2nc_1d_dp(), var2nc_1d_i4(), var2nc_1d_sp(), var2nc_2d_dp(), var2nc_2d_i4(), var2nc_2d_sp(), var2nc_3d_dp(), var2nc_3d_i4(), var2nc_3d_sp(), var2nc_4d_dp(), var2nc_4d_i4(), var2nc_4d_sp(), var2nc_5d_dp(), var2nc_5d_i4(), and var2nc_5d_sp().

Here is the call graph for this function:



Here is the caller graph for this function:



15.47.1.20 var2nc_1d_dp()

```

subroutine mo_ncwrite::var2nc_1d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:), intent(in) arr,

```

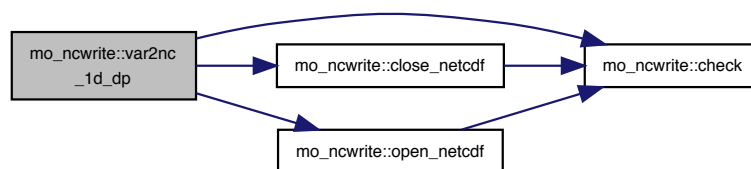
```

character(len=*), dimension(:), intent(in) dnames,
character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.21 var2nc_1d_i4()

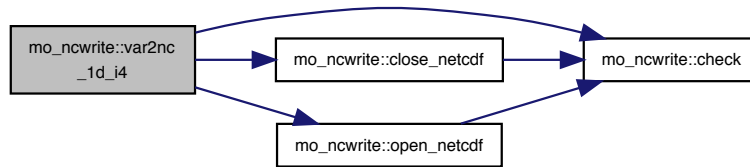
```

subroutine mo_ncwrite::var2nc_1d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



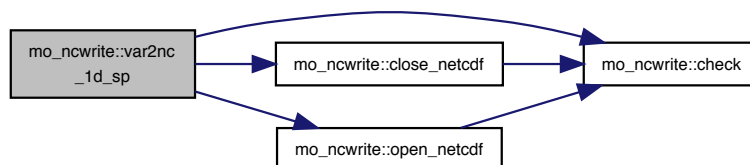
15.47.1.22 var2nc_1d_sp()

```

subroutine mo_ncwrite::var2nc_1d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:, :), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]
  
```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.23 var2nc_2d_dp()

```

subroutine mo_ncwrite::var2nc_2d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:, :), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
  
```

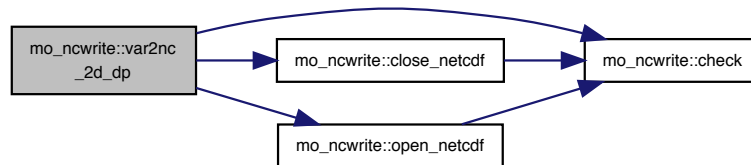
```

character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.24 var2nc_2d_i4()

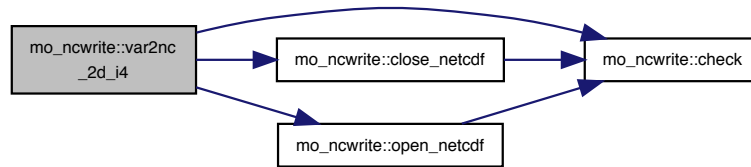
```

subroutine mo_ncwrite::var2nc_2d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



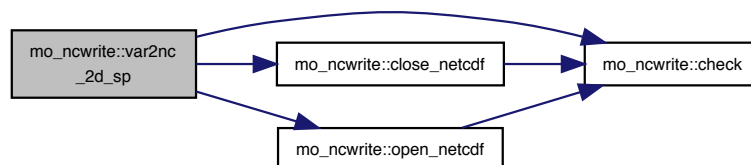
15.47.1.25 var2nc_2d_sp()

```

subroutine mo_ncwrite::var2nc_2d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:, :), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:, :), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]
  
```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.26 var2nc_3d_dp()

```

subroutine mo_ncwrite::var2nc_3d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:, :, :), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
  
```

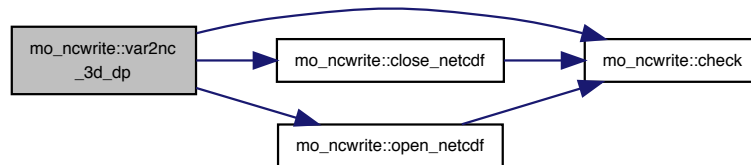
```

character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.27 var2nc_3d_i4()

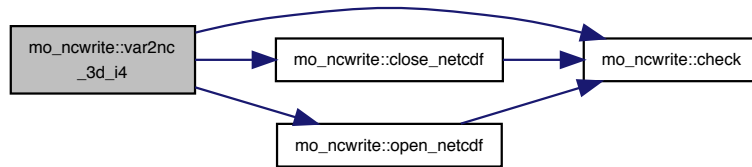
```

subroutine mo_ncwrite::var2nc_3d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



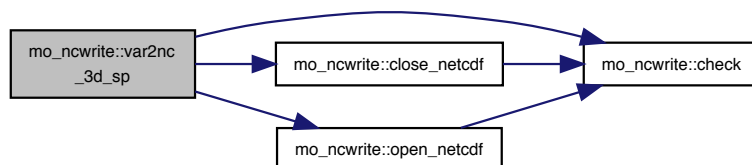
15.47.1.28 var2nc_3d_sp()

```

subroutine mo_ncwrite::var2nc_3d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]
  
```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.29 var2nc_4d_dp()

```

subroutine mo_ncwrite::var2nc_4d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:,:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
  
```

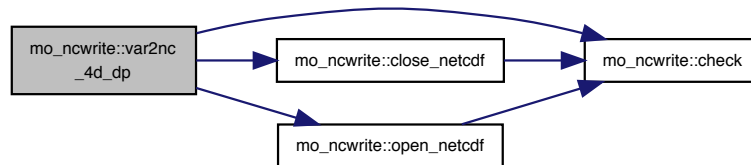
```

character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.30 var2nc_4d_i4()

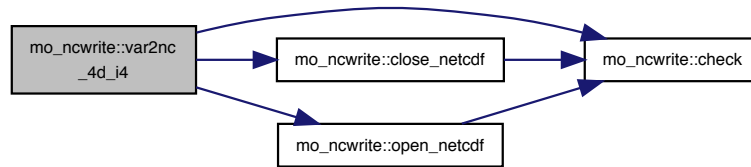
```

subroutine mo_ncwrite::var2nc_4d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



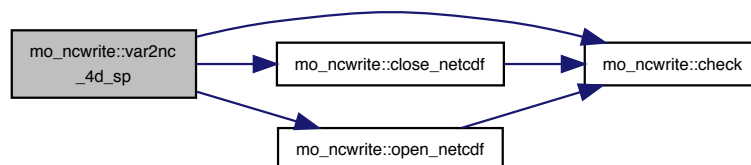
15.47.1.31 var2nc_4d_sp()

```

subroutine mo_ncwrite::var2nc_4d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]
  
```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.32 var2nc_5d_dp()

```

subroutine mo_ncwrite::var2nc_5d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
  
```

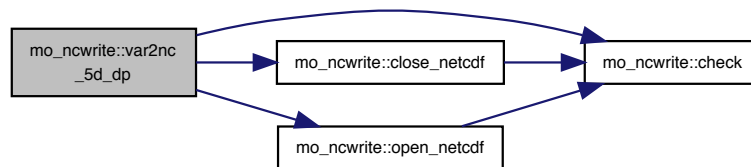
```

character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.33 var2nc_5d_i4()

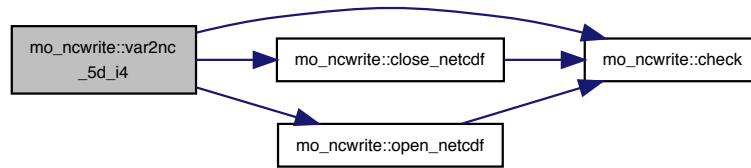
```

subroutine mo_ncwrite::var2nc_5d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]

```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



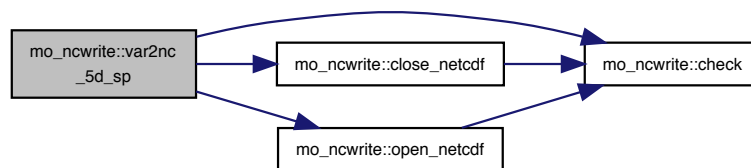
15.47.1.34 var2nc_5d_sp()

```

subroutine mo_ncwrite::var2nc_5d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec ) [private]
  
```

References `check()`, `close_netcdf()`, `ndims`, and `open_netcdf()`.

Here is the call graph for this function:



15.47.1.35 write_dynamic_netcdf()

```

subroutine, public mo_ncwrite::write_dynamic_netcdf (
    integer(i4), intent(in) ncId,
    integer(i4), intent(in) irec )
  
```

References `check()`, `nvars`, and `v`.

Here is the call graph for this function:



15.47.1.36 write_static_netcdf()

```

subroutine, public mo_ncwrite::write_static_netcdf (
    integer(i4), intent(in) ncId )
  
```

References `check()`, `nvars`, and `v`.

Here is the call graph for this function:



15.47.2 Variable Documentation

15.47.2.1 dnc

```

type (dims), dimension(:), allocatable, public mo_ncwrite::dnc
  
```

Referenced by `create_netcdf()`, and `mo_set_netcdf_outputs::set_netcdf()`.

15.47.2.2 gatt

```

type(attribute), dimension(ngatt), public mo_ncwrite::gatt
  
```

Referenced by `create_netcdf()`.

15.47.2.3 maxlen

```
integer(i4), parameter, public mo_ncwrite::maxlen = 256
```

15.47.2.4 nattdim

```
integer(i4), parameter, public mo_ncwrite::nattdim = 2
```

15.47.2.5 ndims

```
integer(i4), public mo_ncwrite::ndims
```

Referenced by `create_netcdf()`, `dump_netcdf_1d_dp()`, `dump_netcdf_1d_i4()`, `dump_netcdf_1d_sp()`, `dump_netcdf_2d_dp()`, `dump_netcdf_2d_i4()`, `dump_netcdf_2d_sp()`, `dump_netcdf_3d_dp()`, `dump_netcdf_3d_i4()`, `dump_netcdf_3d_sp()`, `dump_netcdf_4d_dp()`, `dump_netcdf_4d_i4()`, `dump_netcdf_4d_sp()`, `dump_netcdf_5d_dp()`, `dump_netcdf_5d_i4()`, `dump_netcdf_5d_sp()`, `mo_set_netcdf_outputs::set_netcdf()`, `var2nc_1d_dp()`, `var2nc_1d_i4()`, `var2nc_1d_sp()`, `var2nc_2d_dp()`, `var2nc_2d_i4()`, `var2nc_2d_sp()`, `var2nc_3d_dp()`, `var2nc_3d_i4()`, `var2nc_3d_sp()`, `var2nc_4d_dp()`, `var2nc_4d_i4()`, `var2nc_4d_sp()`, `var2nc_5d_dp()`, `var2nc_5d_i4()`, and `var2nc_5d_sp()`.

15.47.2.6 ngatt

```
integer(i4), parameter, public mo_ncwrite::ngatt = 20
```

Referenced by `create_netcdf()`.

15.47.2.7 nmaxatt

```
integer(i4), parameter, public mo_ncwrite::nmaxatt = 20
```

15.47.2.8 nmaxdim

```
integer(i4), parameter, public mo_ncwrite::nmaxdim = 5
```

15.47.2.9 nvars

```
integer(i4), public mo_ncwrite::nvars
```

Referenced by `create_netcdf()`, `mo_set_netcdf_outputs::set_netcdf()`, `write_dynamic_netcdf()`, and `write_static_netcdf()`.

15.47.2.10 v

```
type(variable), dimension(:), allocatable, public mo_ncwrite::v
```

Referenced by `create_netcdf()`, `mo_set_netcdf_outputs::set_netcdf()`, `write_dynamic_netcdf()`, and `write_static_netcdf()`.

15.48 mo_netcdf Module Reference

NetCDF Fortran 90 interface wrapper.

Data Types

- interface `ncdataset`
Provides basic file modification functionality.
- type `ncdimension`
Provides the dimension access functionality.
- interface `ncvariable`

Functions/Subroutines

- subroutine `initncvariable` (self, id, parent)
- subroutine `initncdimension` (self, id, parent)
- subroutine `initncdataset` (self, fname, mode)
- type(`ncvariable`) function `newncvariable` (id, parent)
- type(`ncdimension`) function `newncdimension` (id, parent)
- type(`ncdataset`) function `newncdataset` (fname, mode)
- subroutine `close` (self)
- integer(i4) function `getnvariables` (self)
- integer(i4) function, dimension(:), allocatable `getvariableids` (self)
- type(`ncvariable`) function, dimension(:), allocatable `getvariables` (self)
- character(len=256) function `getdimensionname` (self)
- integer(i4) function `getdimensionlength` (self)
- logical function `isdatasetunlimited` (self)
- type(`ncdimension`) function `getunlimitedddimension` (self)
- logical function `equalncdimensions` (dim1, dim2)
- logical function `isunlimitedddimension` (self)
- type(`ncdimension`) function `setdimension` (self, name, length)
- logical function `hasvariable` (self, name)
- logical function `hasdimension` (self, name)
- type(`ncvariable`) function `setvariablewithids` (self, name, dtype, dimensions, contiguous, chunksizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(`ncvariable`) function `setvariablewithnames` (self, name, dtype, dimensions, contiguous, chunksizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(`ncvariable`) function `setvariablewithtypes` (self, name, dtype, dimensions, contiguous, chunksizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(`ncdimension`) function `getdimensionbyid` (self, id)
- type(`ncdimension`) function `getdimensionbyname` (self, name)
- type(`ncvariable`) function `getvariablebyname` (self, name)
- character(len=256) function `getvariablename` (self)
- integer(i4) function `getnodimensions` (self)
- type(`ncdimension`) function, dimension(:), allocatable `getvariabledimensions` (self)
- integer(i4) function, dimension(:), allocatable `getvariablesshape` (self)
- character(3) function `getvariabledtype` (self)
- logical function `isunlimitedvariable` (self)
- logical function `hasattribute` (self, name)

- subroutine [setglobalattributechar](#) (self, name, data)
- subroutine [setglobalattributei8](#) (self, name, data)
- subroutine [setglobalattributei16](#) (self, name, data)
- subroutine [setglobalattributei32](#) (self, name, data)
- subroutine [setglobalattributef32](#) (self, name, data)
- subroutine [setglobalattributef64](#) (self, name, data)
- subroutine [getglobalattributechar](#) (self, name, avalue)
- subroutine [getglobalattributei8](#) (self, name, avalue)
- subroutine [getglobalattributei16](#) (self, name, avalue)
- subroutine [getglobalattributei32](#) (self, name, avalue)
- subroutine [getglobalattributef32](#) (self, name, avalue)
- subroutine [getglobalattributef64](#) (self, name, avalue)
- subroutine [setvariableattributechar](#) (self, name, data)
- subroutine [setvariableattributei8](#) (self, name, data)
- subroutine [setvariableattributei16](#) (self, name, data)
- subroutine [setvariableattributei32](#) (self, name, data)
- subroutine [setvariableattributef32](#) (self, name, data)
- subroutine [setvariableattributef64](#) (self, name, data)
- subroutine [getvariableattributechar](#) (self, name, avalue)
- subroutine [getvariableattributei8](#) (self, name, avalue)
- subroutine [getvariableattributei16](#) (self, name, avalue)
- subroutine [getvariableattributei32](#) (self, name, avalue)
- subroutine [getvariableattributef32](#) (self, name, avalue)
- subroutine [getvariableattributef64](#) (self, name, avalue)
- subroutine [setvariablefillvaluei8](#) (self, fvalue)
- subroutine [setvariablefillvaluei16](#) (self, fvalue)
- subroutine [setvariablefillvaluei32](#) (self, fvalue)
- subroutine [setvariablefillvaluef32](#) (self, fvalue)
- subroutine [setvariablefillvaluef64](#) (self, fvalue)
- subroutine [getvariablefillvaluei8](#) (self, fvalue)
- subroutine [getvariablefillvaluei16](#) (self, fvalue)
- subroutine [getvariablefillvaluei32](#) (self, fvalue)
- subroutine [getvariablefillvaluef32](#) (self, fvalue)
- subroutine [getvariablefillvaluef64](#) (self, fvalue)
- subroutine [setdatascalar_i8](#) (self, values, start)
- subroutine [setdata1di8](#) (self, values, start, cnt, stride, map)
- subroutine [setdata2di8](#) (self, values, start, cnt, stride, map)
- subroutine [setdata3di8](#) (self, values, start, cnt, stride, map)
- subroutine [setdata4di8](#) (self, values, start, cnt, stride, map)
- subroutine [setdata5di8](#) (self, values, start, cnt, stride, map)
- subroutine [setdatascalar_i16](#) (self, values, start)
- subroutine [setdata1di16](#) (self, values, start, cnt, stride, map)
- subroutine [setdata2di16](#) (self, values, start, cnt, stride, map)
- subroutine [setdata3di16](#) (self, values, start, cnt, stride, map)
- subroutine [setdata4di16](#) (self, values, start, cnt, stride, map)
- subroutine [setdata5di16](#) (self, values, start, cnt, stride, map)
- subroutine [setdatascalar_i32](#) (self, values, start)
- subroutine [setdata1di32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata2di32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata3di32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata4di32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata5di32](#) (self, values, start, cnt, stride, map)
- subroutine [setdatascalar_f32](#) (self, values, start)
- subroutine [setdata1df32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata2df32](#) (self, values, start, cnt, stride, map)

- subroutine [setdata3df32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata4df32](#) (self, values, start, cnt, stride, map)
- subroutine [setdata5df32](#) (self, values, start, cnt, stride, map)
- subroutine [setdatascalarf64](#) (self, values, start)
- subroutine [setdata1df64](#) (self, values, start, cnt, stride, map)
- subroutine [setdata2df64](#) (self, values, start, cnt, stride, map)
- subroutine [setdata3df64](#) (self, values, start, cnt, stride, map)
- subroutine [setdata4df64](#) (self, values, start, cnt, stride, map)
- subroutine [setdata5df64](#) (self, values, start, cnt, stride, map)
- subroutine [getdatascalarf8](#) (self, data, start, cnt, stride, map)
- subroutine [getdata1df8](#) (self, data, start, cnt, stride, map)
- subroutine [getdata2df8](#) (self, data, start, cnt, stride, map)
- subroutine [getdata3df8](#) (self, data, start, cnt, stride, map)
- subroutine [getdata4df8](#) (self, data, start, cnt, stride, map)
- subroutine [getdata5df8](#) (self, data, start, cnt, stride, map)
- subroutine [getdatascalarf16](#) (self, data, start, cnt, stride, map)
- subroutine [getdata1df16](#) (self, data, start, cnt, stride, map)
- subroutine [getdata2df16](#) (self, data, start, cnt, stride, map)
- subroutine [getdata3df16](#) (self, data, start, cnt, stride, map)
- subroutine [getdata4df16](#) (self, data, start, cnt, stride, map)
- subroutine [getdata5df16](#) (self, data, start, cnt, stride, map)
- subroutine [getdatascalarf32](#) (self, data, start, cnt, stride, map)
- subroutine [getdata1df32](#) (self, data, start, cnt, stride, map)
- subroutine [getdata2df32](#) (self, data, start, cnt, stride, map)
- subroutine [getdata3df32](#) (self, data, start, cnt, stride, map)
- subroutine [getdata4df32](#) (self, data, start, cnt, stride, map)
- subroutine [getdata5df32](#) (self, data, start, cnt, stride, map)
- subroutine [getdatascalarf64](#) (self, data, start, cnt, stride, map)
- subroutine [getdata1df64](#) (self, data, start, cnt, stride, map)
- subroutine [getdata2df64](#) (self, data, start, cnt, stride, map)
- subroutine [getdata3df64](#) (self, data, start, cnt, stride, map)
- subroutine [getdata4df64](#) (self, data, start, cnt, stride, map)
- subroutine [getdata5df64](#) (self, data, start, cnt, stride, map)
- integer(i4) function, dimension(datarank) [getreaddatashape](#) (var, datarank, instart, incnt, instride)
- integer(i4) function [getdtypefromstring](#) (dtype)
- character(3) function [getdtypefrominteger](#) (dtype)
- subroutine [check](#) (status, msg)

15.48.1 Detailed Description

NetCDF Fortran 90 interface wrapper.

A thin wrapper around the NetCDF Fortran 90 interface. Provided are currently 3 user facing derived Types:

1. NcDataset
2. NcDimension
3. NcVariable

Authors

David Schaefer

Date

Jun 2015

15.48.2 Function/Subroutine Documentation**15.48.2.1 check()**

```
subroutine mo_netcdf::check (
    integer(i4), intent(in) status,
    character(*), intent(in) msg )
```

Referenced by `close()`, `getdata1df32()`, `getdata1df64()`, `getdata1di16()`, `getdata1di32()`, `getdata1di8()`, `getdata2df32()`, `getdata2df64()`, `getdata2di16()`, `getdata2di32()`, `getdata2di8()`, `getdata3df32()`, `getdata3df64()`, `getdata3di16()`, `getdata3di32()`, `getdata3di8()`, `getdata4df32()`, `getdata4df64()`, `getdata4di16()`, `getdata4di32()`, `getdata4di8()`, `getdata5df32()`, `getdata5df64()`, `getdata5di16()`, `getdata5di32()`, `getdata5di8()`, `getdatascalarf32()`, `getdatascalarf64()`, `getdatascalar16()`, `getdatascalar32()`, `getdatascalar8()`, `getdimensionbyid()`, `getdimensionbyname()`, `getdimensionlength()`, `getdimensionname()`, `getglobalattributechar()`, `getglobalattributef32()`, `getglobalattributef64()`, `getglobalattributei16()`, `getglobalattributei32()`, `getglobalattributei8()`, `getnodimensions()`, `getnvariables()`, `getunlimiteddimension()`, `getvariableattributechar()`, `getvariableattributef32()`, `getvariableattributef64()`, `getvariableattributei16()`, `getvariableattributei32()`, `getvariableattributei8()`, `getvariablebyname()`, `getvariabledimensions()`, `getvariabledtype()`, `getvariableids()`, `getvariablename()`, `initncdataset()`, `isdatasetunlimited()`, `setdata1df32()`, `setdata1df64()`, `setdata1di16()`, `setdata1di32()`, `setdata1di8()`, `setdata2df32()`, `setdata2df64()`, `setdata2di16()`, `setdata2di32()`, `setdata2di8()`, `setdata3df32()`, `setdata3df64()`, `setdata3di16()`, `setdata3di32()`, `setdata3di8()`, `setdata4df32()`, `setdata4df64()`, `setdata4di16()`, `setdata4di32()`, `setdata4di8()`, `setdata5df32()`, `setdata5df64()`, `setdata5di16()`, `setdata5di32()`, `setdata5di8()`, `setdatascalarf32()`, `setdatascalarf64()`, `setdatascalar16()`, `setdatascalar32()`, `setdatascalar8()`, `setdimension()`, `setglobalattributechar()`, `setglobalattributef32()`, `setglobalattributef64()`, `setglobalattributei16()`, `setglobalattributei32()`, `setglobalattributei8()`, `setvariableattributechar()`, `setvariableattributef32()`, `setvariableattributef64()`, `setvariableattributei16()`, `setvariableattributei32()`, `setvariableattributei8()`, and `setvariablewithids()`.

15.48.2.2 close()

```
subroutine mo_netcdf::close (
    class(ncdataset) self )
```

References `check()`.

Here is the call graph for this function:



15.48.2.3 equalncdimensions()

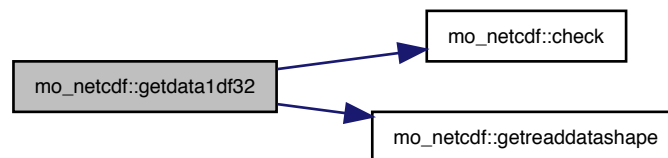
```
logical function mo_netcdf::equalncdimensions (
    type(ncdimension), intent(in) dim1,
    type(ncdimension), intent(in) dim2 )
```

15.48.2.4 getdata1df32()

```
subroutine mo_netcdf::getdata1df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:

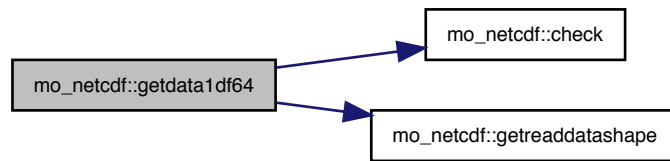


15.48.2.5 getdata1df64()

```
subroutine mo_netcdf::getdata1df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



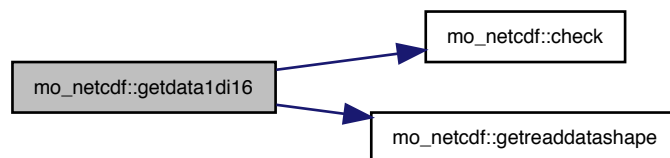
15.48.2.6 getdata1di16()

```

subroutine mo_netcdf::getdata1di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



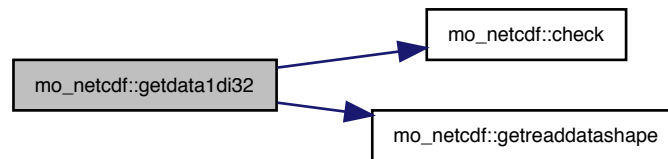
15.48.2.7 getdata1di32()

```

subroutine mo_netcdf::getdata1di32 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



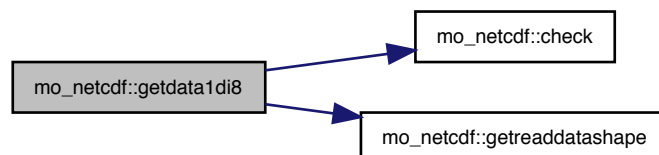
15.48.2.8 getdata1di8()

```

subroutine mo_netcdf::getdata1di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



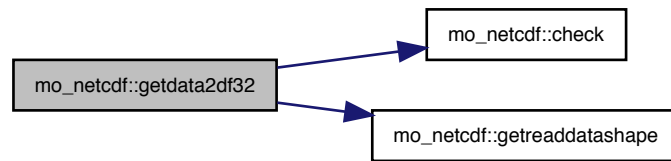
15.48.2.9 getdata2df32()

```

subroutine mo_netcdf::getdata2df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:, :), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



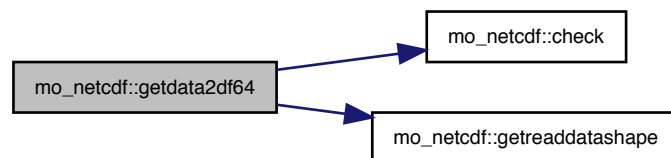
15.48.2.10 getdata2df64()

```

subroutine mo_netcdf::getdata2df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:, :), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



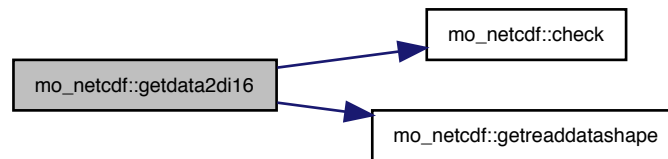
15.48.2.11 getdata2di16()

```

subroutine mo_netcdf::getdata2di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:, :), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.12 getdata2di32()

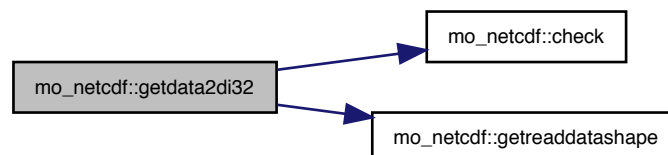
```

subroutine mo_netcdf::getdata2di32 (
  class(ncvariable), intent(in) self,
  integer(i4), dimension(:, :), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.13 getdata2di8()

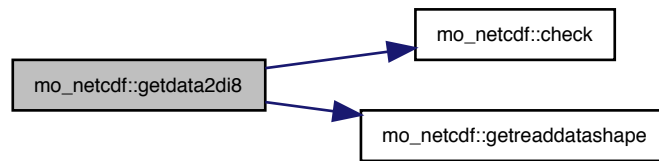
```

subroutine mo_netcdf::getdata2di8 (
  class(ncvariable), intent(in) self,
  integer(i1), dimension(:, :), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



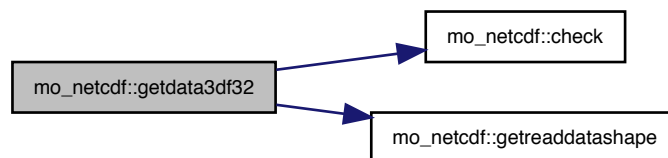
15.48.2.14 getdata3df32()

```

subroutine mo_netcdf::getdata3df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



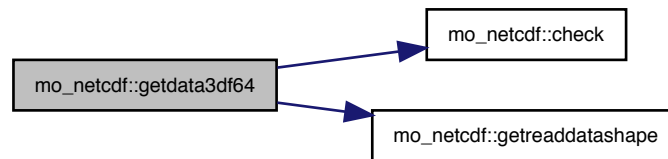
15.48.2.15 getdata3df64()

```

subroutine mo_netcdf::getdata3df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:,:,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.16 getdata3di16()

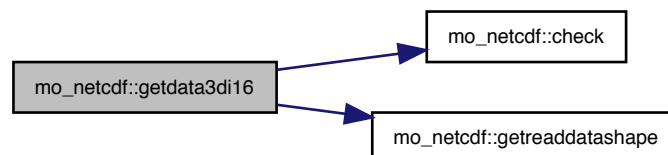
```

subroutine mo_netcdf::getdata3di16 (
  class(ncvariable), intent(in) self,
  integer(i2), dimension(:,:,:), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.17 getdata3di32()

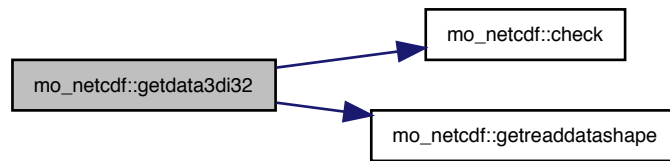
```

subroutine mo_netcdf::getdata3di32 (
  class(ncvariable), intent(in) self,
  integer(i4), dimension(:,:,:), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.18 getdata3di8()

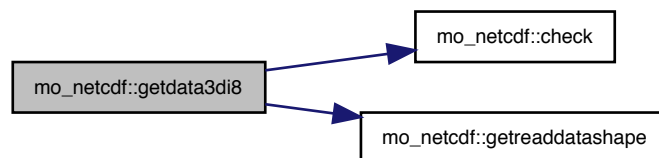
```

subroutine mo_netcdf::getdata3di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:,:,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.19 getdata4df32()

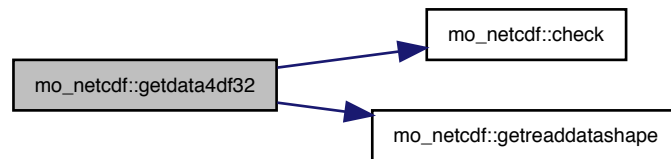
```

subroutine mo_netcdf::getdata4df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:,:,), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.20 `getdata4df64()`

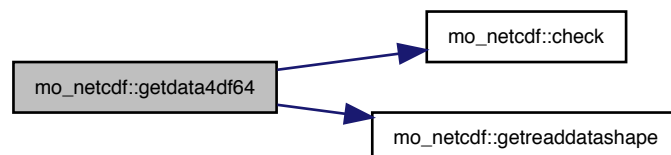
```

subroutine mo_netcdf::getdata4df64 (
  class(ncvariable), intent(in) self,
  real(dp), dimension(:,:,:,:), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.21 `getdata4di16()`

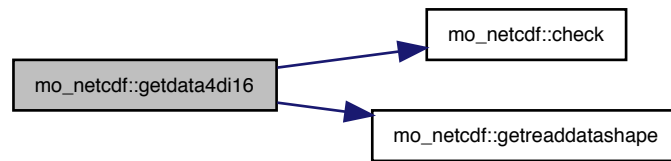
```

subroutine mo_netcdf::getdata4di16 (
  class(ncvariable), intent(in) self,
  integer(i2), dimension(:,:,:,:), intent(out), allocatable data,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



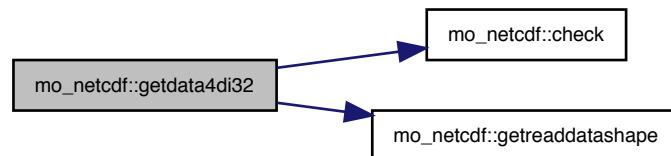
15.48.2.22 getdata4di32()

```

subroutine mo_netcdf::getdata4di32 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:,:,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



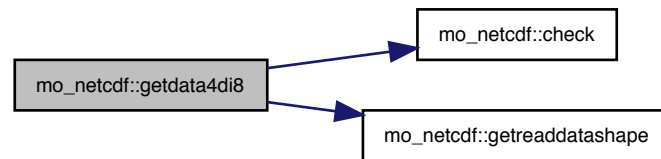
15.48.2.23 getdata4di8()

```

subroutine mo_netcdf::getdata4di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:,:,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



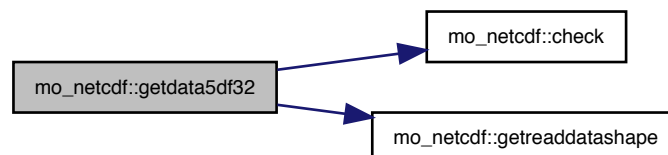
15.48.2.24 getdata5df32()

```

subroutine mo_netcdf::getdata5df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:,:,,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



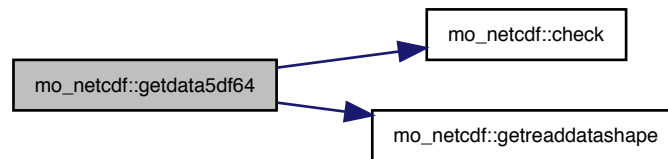
15.48.2.25 getdata5df64()

```

subroutine mo_netcdf::getdata5df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:,:,:,,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.26 getdata5di16()

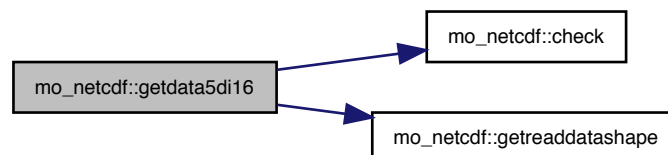
```

subroutine mo_netcdf::getdata5di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:,:,:,,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.27 getdata5di32()

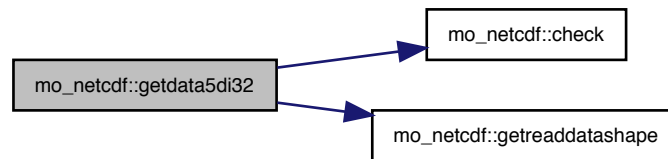
```

subroutine mo_netcdf::getdata5di32 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:,:,:,,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.28 getdata5di8()

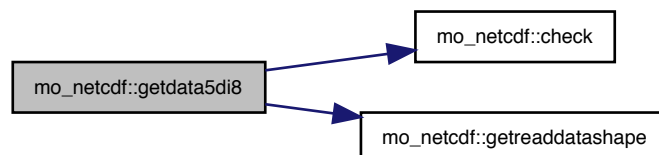
```

subroutine mo_netcdf::getdata5di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:,:,:,,:), intent(out), allocatable data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`, and `getreaddatashape()`.

Here is the call graph for this function:



15.48.2.29 getdatascalarf32()

```

subroutine mo_netcdf::getdatascalarf32 (
    class(ncvariable), intent(in) self,
    real(sp), intent(out) data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`.

Here is the call graph for this function:



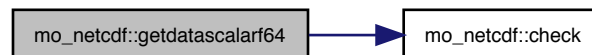
15.48.2.30 getdatascalarf64()

```

subroutine mo_netcdf::getdatascalarf64 (
    class(ncvariable), intent(in) self,
    real(dp), intent(out) data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.31 getdatascalarl16()

```

subroutine mo_netcdf::getdatascalarl16 (
    class(ncvariable), intent(in) self,
    integer(i2), intent(out) data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



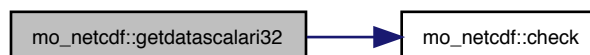
15.48.2.32 getdatascalari32()

```

subroutine mo_netcdf::getdatascalari32 (
    class(ncvariable), intent(in) self,
    integer(i4), intent(out) data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



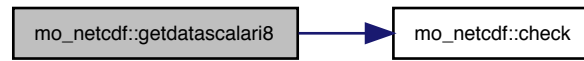
15.48.2.33 getdatascalari8()

```

subroutine mo_netcdf::getdatascalari8 (
    class(ncvariable), intent(in) self,
    integer(i1), intent(out) data,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.34 getdimensionbyid()

```
type(ncdimension) function mo_netcdf::getdimensionbyid (  
    class(ncdataset), intent(in) self,  
    integer(i4) id )
```

References check().

Here is the call graph for this function:

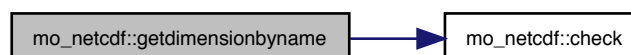


15.48.2.35 getdimensionbyname()

```
type(ncdimension) function mo_netcdf::getdimensionbyname (  
    class(ncdataset), intent(in) self,  
    character(*) name )
```

References check().

Here is the call graph for this function:

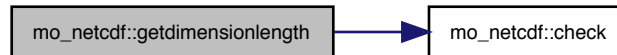


15.48.2.36 getdimensionlength()

```
integer(i4) function mo_netcdf::getdimensionlength (  
    class(ncdimension), intent(in) self )
```

References check().

Here is the call graph for this function:

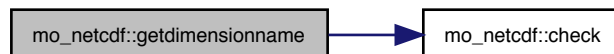


15.48.2.37 getdimensionname()

```
character(len=256) function mo_netcdf::getdimensionname (  
    class(ncdimension), intent(in) self )
```

References check().

Here is the call graph for this function:

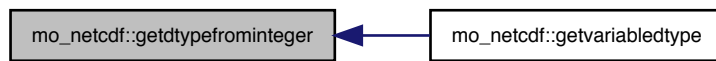


15.48.2.38 getdtypefrominteger()

```
character(3) function mo_netcdf::getdtypefrominteger (  
    integer(i4) dtype )
```

Referenced by getvariabledtype().

Here is the caller graph for this function:



15.48.2.39 getdtypefromstring()

```
integer(i4) function mo_netcdf::getdtypefromstring (
    character(*) dtype )
```

Referenced by `setvariablewithids()`.

Here is the caller graph for this function:

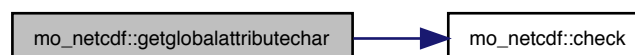


15.48.2.40 getglobalattributechar()

```
subroutine mo_netcdf::getglobalattributechar (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name,
    character(*), intent(out) avalue )
```

References `check()`.

Here is the call graph for this function:

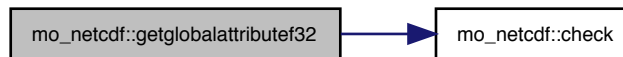


15.48.2.41 getglobalattribuf32()

```
subroutine mo_netcdf::getglobalattribuf32 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    real(sp), intent(out) avalue )
```

References check().

Here is the call graph for this function:

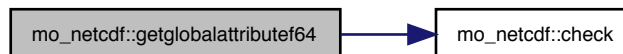


15.48.2.42 getglobalattribuf64()

```
subroutine mo_netcdf::getglobalattribuf64 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    real(dp), intent(out) avalue )
```

References check().

Here is the call graph for this function:

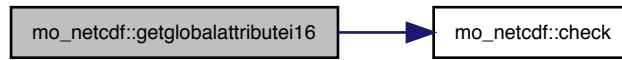


15.48.2.43 getglobalattributei16()

```
subroutine mo_netcdf::getglobalattributei16 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i2), intent(out) avalue )
```

References check().

Here is the call graph for this function:

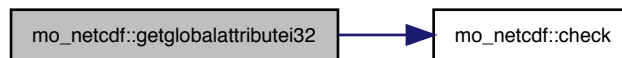


15.48.2.44 getglobalattributei32()

```
subroutine mo_netcdf::getglobalattributei32 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i4), intent(out) avalue )
```

References check().

Here is the call graph for this function:



15.48.2.45 getglobalattributei8()

```
subroutine mo_netcdf::getglobalattributei8 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i1), intent(out) avalue )
```

References check().

Here is the call graph for this function:



15.48.2.46 getnodimensions()

```
integer(i4) function mo_netcdf::getnodimensions (
    class(ncvariable), intent(in) self )
```

References check().

Here is the call graph for this function:



15.48.2.47 getnovariables()

```
integer(i4) function mo_netcdf::getnovariables (
    class(ncdataset), intent(in) self )
```

References check().

Here is the call graph for this function:

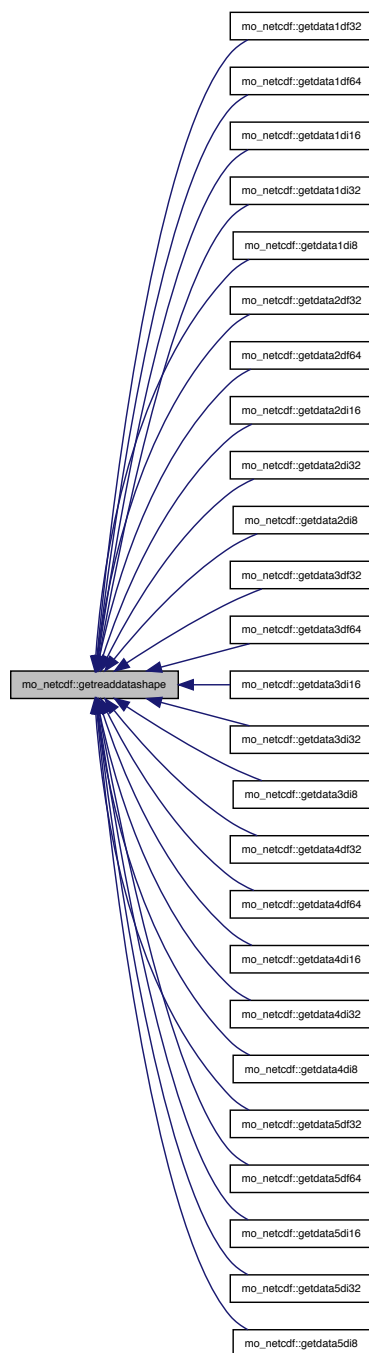


15.48.2.48 getreaddatashape()

```
integer(i4) function, dimension(datarank) mo_netcdf::getreaddatashape (
    type(ncvariable), intent(in) var,
    integer(i4), intent(in) datarank,
    integer(i4), dimension(:), intent(in), optional instart,
    integer(i4), dimension(:), intent(in), optional incnt,
    integer(i4), dimension(:), intent(in), optional instride )
```

Referenced by getdata1df32(), getdata1df64(), getdata1di16(), getdata1di32(), getdata1di8(), getdata2df32(), getdata2df64(), getdata2di16(), getdata2di32(), getdata2di8(), getdata3df32(), getdata3df64(), getdata3di16(), getdata3di32(), getdata3di8(), getdata4df32(), getdata4df64(), getdata4di16(), getdata4di32(), getdata4di8(), getdata5df32(), getdata5df64(), getdata5di16(), getdata5di32(), and getdata5di8().

Here is the caller graph for this function:



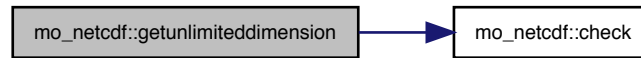
15.48.2.49 getunlimiteddimension()

```

type(ncdimension) function mo_netcdf::getunlimiteddimension (
    class(ncdataset), intent(in) self )
  
```

References check().

Here is the call graph for this function:

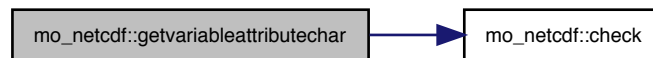


15.48.2.50 `getvariableattributechar()`

```
subroutine mo_netcdf::getvariableattributechar (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    character(*), intent(out) avalue )
```

References `check()`.

Here is the call graph for this function:

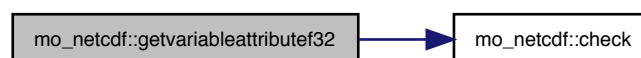


15.48.2.51 `getvariableattributef32()`

```
subroutine mo_netcdf::getvariableattributef32 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    real(sp), intent(out) avalue )
```

References `check()`.

Here is the call graph for this function:

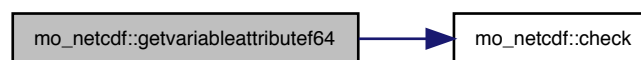


15.48.2.52 getvariableattributef64()

```
subroutine mo_netcdf::getvariableattributef64 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    real(dp), intent(out) avalue )
```

References check().

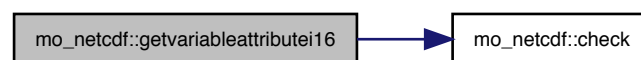
Here is the call graph for this function:

**15.48.2.53 getvariableattributei16()**

```
subroutine mo_netcdf::getvariableattributei16 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i2), intent(out) avalue )
```

References check().

Here is the call graph for this function:

**15.48.2.54 getvariableattributei32()**

```
subroutine mo_netcdf::getvariableattributei32 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i4), intent(out) avalue )
```

References check().

Here is the call graph for this function:

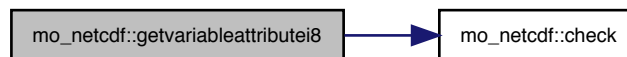


15.48.2.55 getvariableattributei8()

```
subroutine mo_netcdf::getvariableattributei8 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i1), intent(out) avalue )
```

References check().

Here is the call graph for this function:

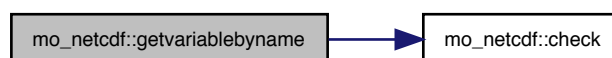


15.48.2.56 getvariablebyname()

```
type(ncvariable) function mo_netcdf::getvariablebyname (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name )
```

References check().

Here is the call graph for this function:

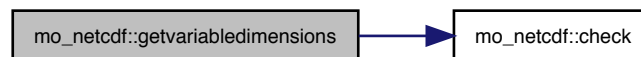


15.48.2.57 getvariablenumdimensions()

```
type(ncdimension) function, dimension(:), allocatable mo_netcdf::getvariablenumdimensions (
    class(ncvariable), intent(in) self )
```

References check().

Here is the call graph for this function:

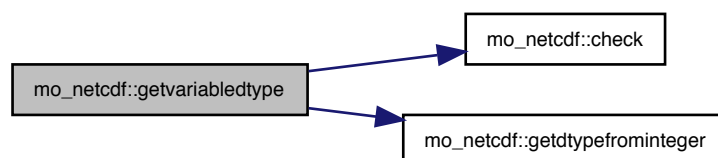


15.48.2.58 getvariabletype()

```
character(3) function mo_netcdf::getvariabletype (
    class(ncvariable), intent(in) self )
```

References check(), and getdtypefrominteger().

Here is the call graph for this function:



15.48.2.59 getvariablefillvaluef32()

```
subroutine mo_netcdf::getvariablefillvaluef32 (
    class(ncvariable), intent(in) self,
    real(sp), intent(out) fvalue )
```

15.48.2.60 getvariablefillvaluef64()

```
subroutine mo_netcdf::getvariablefillvaluef64 (  
    class(ncvariable), intent(in) self,  
    real(dp), intent(out) fvalue )
```

15.48.2.61 getvariablefillvaluei16()

```
subroutine mo_netcdf::getvariablefillvaluei16 (  
    class(ncvariable), intent(in) self,  
    integer(i2), intent(out) fvalue )
```

15.48.2.62 getvariablefillvaluei32()

```
subroutine mo_netcdf::getvariablefillvaluei32 (  
    class(ncvariable), intent(in) self,  
    integer(i4), intent(out) fvalue )
```

15.48.2.63 getvariablefillvaluei8()

```
subroutine mo_netcdf::getvariablefillvaluei8 (  
    class(ncvariable), intent(in) self,  
    integer(i1), intent(out) fvalue )
```

15.48.2.64 getvariableids()

```
integer(i4) function, dimension(:), allocatable mo_netcdf::getvariableids (  
    class(ncdataset), intent(in) self )
```

References check().

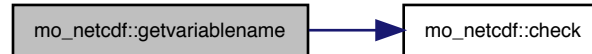
Here is the call graph for this function:

**15.48.2.65 getvariablename()**

```
character(len=256) function mo_netcdf::getvariablename (  
    class(ncvariable), intent(in) self )
```

References check().

Here is the call graph for this function:



15.48.2.66 getvariables()

```
type(ncvariable) function, dimension(:), allocatable mo_netcdf::getvariables (
    class(ncdataset), intent(in) self )
```

15.48.2.67 getvariablesshape()

```
integer(i4) function, dimension(:), allocatable mo_netcdf::getvariablesshape (
    class(ncvariable), intent(in) self )
```

15.48.2.68 hasattribute()

```
logical function mo_netcdf::hasattribute (
    class(ncvariable), intent(in) self,
    character(*), intent(in) name )
```

15.48.2.69 hasdimension()

```
logical function mo_netcdf::hasdimension (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name )
```

15.48.2.70 hasvariable()

```
logical function mo_netcdf::hasvariable (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name )
```

15.48.2.71 initncdataset()

```
subroutine mo_netcdf::initncdataset (  
    class(ncdataset), intent(inout) self,  
    character(*), intent(in) fname,  
    character(1), intent(in) mode )
```

References check().

Here is the call graph for this function:



15.48.2.72 initncdimension()

```
subroutine mo_netcdf::initncdimension (  
    class(ncdimension), intent(inout) self,  
    integer(i4), intent(in) id,  
    type(ncdataset), intent(in) parent )
```

15.48.2.73 initncvariable()

```
subroutine mo_netcdf::initncvariable (  
    class(ncvariable), intent(inout) self,  
    integer(i4), intent(in) id,  
    type(ncdataset), intent(in) parent )
```

15.48.2.74 isdatasetunlimited()

```
logical function mo_netcdf::isdatasetunlimited (  
    class(ncdataset), intent(in) self )
```

References check().

Here is the call graph for this function:



15.48.2.75 isunlimiteddimension()

```
logical function mo_netcdf::isunlimiteddimension (
    class(ncdimension), intent(in) self )
```

15.48.2.76 isunlimitedvariable()

```
logical function mo_netcdf::isunlimitedvariable (
    class(ncvariable), intent(in) self )
```

15.48.2.77 newncdataset()

```
type(ncdataset) function mo_netcdf::newncdataset (
    character(*), intent(in) fname,
    character(1), intent(in) mode )
```

15.48.2.78 newncdimension()

```
type(ncdimension) function mo_netcdf::newncdimension (
    integer(i4), intent(in) id,
    type(ncdataset), intent(in) parent )
```

15.48.2.79 newncvariable()

```
type(ncvariable) function mo_netcdf::newncvariable (
    integer(i4), intent(in) id,
    type(ncdataset), intent(in) parent )
```

15.48.2.80 setdata1df32()

```
subroutine mo_netcdf::setdata1df32 (
    class(ncvariable), intent(in) self,
```

```

real(sp), dimension(:), intent(in) values,
integer(i4), dimension(:), intent(in), optional start,
integer(i4), dimension(:), intent(in), optional cnt,
integer(i4), dimension(:), intent(in), optional stride,
integer(i4), dimension(:), intent(in), optional map )

```

References `check()`.

Here is the call graph for this function:



15.48.2.81 setdata1df64()

```

subroutine mo_netcdf::setdata1df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References `check()`.

Here is the call graph for this function:



15.48.2.82 setdata1di16()

```

subroutine mo_netcdf::setdata1di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.83 setdata1di32()

```

subroutine mo_netcdf::setdata1di32 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



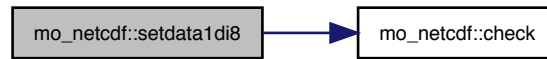
15.48.2.84 setdata1di8()

```

subroutine mo_netcdf::setdata1di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.85 setdata2df32()

```

subroutine mo_netcdf::setdata2df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.86 setdata2df64()

```

subroutine mo_netcdf::setdata2df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.87 setdata2di16()

```

subroutine mo_netcdf::setdata2di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



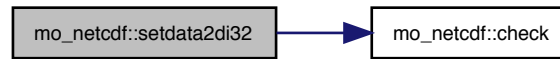
15.48.2.88 setdata2di32()

```

subroutine mo_netcdf::setdata2di32 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.89 setdata2di8()

```

subroutine mo_netcdf::setdata2di8 (
    class(ncvariable), intent(in) self,
    integer(i1), dimension(:, :), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

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

Here is the call graph for this function:



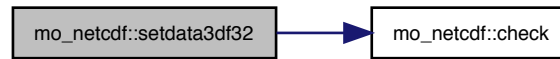
15.48.2.90 setdata3df32()

```

subroutine mo_netcdf::setdata3df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:, :, :), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

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

Here is the call graph for this function:



15.48.2.91 setdata3df64()

```

subroutine mo_netcdf::setdata3df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:,:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.92 setdata3di16()

```

subroutine mo_netcdf::setdata3di16 (
    class(ncvariable), intent(in) self,
    integer(i2), dimension(:,:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.93 setdata3di32()

```

subroutine mo_netcdf::setdata3di32 (
  class(ncvariable), intent(in) self,
  integer(i4), dimension(:,:,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.94 setdata3di8()

```

subroutine mo_netcdf::setdata3di8 (
  class(ncvariable), intent(in) self,
  integer(i1), dimension(:,:,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.95 setdata4df32()

```

subroutine mo_netcdf::setdata4df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



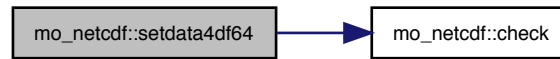
15.48.2.96 setdata4df64()

```

subroutine mo_netcdf::setdata4df64 (
    class(ncvariable), intent(in) self,
    real(dp), dimension(:,:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.97 setdata4di16()

```

subroutine mo_netcdf::setdata4di16 (
  class(ncvariable), intent(in) self,
  integer(i2), dimension(:,:,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.98 setdata4di32()

```

subroutine mo_netcdf::setdata4di32 (
  class(ncvariable), intent(in) self,
  integer(i4), dimension(:,:,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.99 setdata4di8()

```

subroutine mo_netcdf::setdata4di8 (
    class(ncvariable), intent(in) self,
    integer(i4), dimension(:,:,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



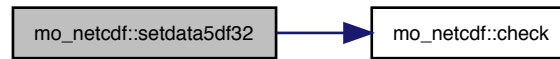
15.48.2.100 setdata5df32()

```

subroutine mo_netcdf::setdata5df32 (
    class(ncvariable), intent(in) self,
    real(sp), dimension(:,:,:,,:), intent(in) values,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional cnt,
    integer(i4), dimension(:), intent(in), optional stride,
    integer(i4), dimension(:), intent(in), optional map )
  
```

References check().

Here is the call graph for this function:



15.48.2.101 setdata5df64()

```

subroutine mo_netcdf::setdata5df64 (
  class(ncvariable), intent(in) self,
  real(dp), dimension(:,:,:,,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`.

Here is the call graph for this function:



15.48.2.102 setdata5di16()

```

subroutine mo_netcdf::setdata5di16 (
  class(ncvariable), intent(in) self,
  integer(i2), dimension(:,:,:,,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References `check()`.

Here is the call graph for this function:



15.48.2.103 setdata5di32()

```

subroutine mo_netcdf::setdata5di32 (
  class(ncvariable), intent(in) self,
  integer(i4), dimension(:,:,:,,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:



15.48.2.104 setdata5di8()

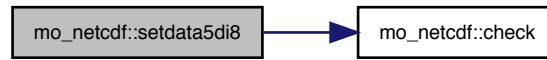
```

subroutine mo_netcdf::setdata5di8 (
  class(ncvariable), intent(in) self,
  integer(i1), dimension(:,:,:,,:), intent(in) values,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional cnt,
  integer(i4), dimension(:), intent(in), optional stride,
  integer(i4), dimension(:), intent(in), optional map )

```

References check().

Here is the call graph for this function:

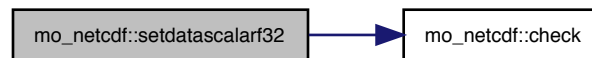


15.48.2.105 `setdatascalarf32()`

```
subroutine mo_netcdf::setdatascalarf32 (  
    class(ncvariable), intent(in) self,  
    real(sp), intent(in) values,  
    integer(i4), dimension(:), intent(in), optional start )
```

References `check()`.

Here is the call graph for this function:

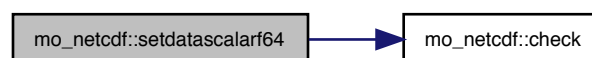


15.48.2.106 `setdatascalarf64()`

```
subroutine mo_netcdf::setdatascalarf64 (  
    class(ncvariable), intent(in) self,  
    real(dp), intent(in) values,  
    integer(i4), dimension(:), intent(in), optional start )
```

References `check()`.

Here is the call graph for this function:

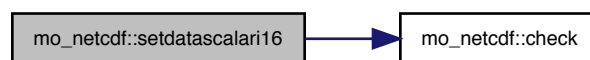


15.48.2.107 setdatascalari16()

```
subroutine mo_netcdf::setdatascalari16 (  
    class(ncvariable), intent(in) self,  
    integer(i2), intent(in) values,  
    integer(i4), dimension(:), intent(in), optional start )
```

References check().

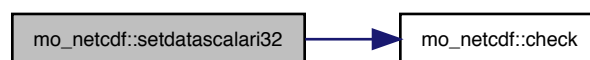
Here is the call graph for this function:

**15.48.2.108 setdatascalari32()**

```
subroutine mo_netcdf::setdatascalari32 (  
    class(ncvariable), intent(in) self,  
    integer(i4), intent(in) values,  
    integer(i4), dimension(:), intent(in), optional start )
```

References check().

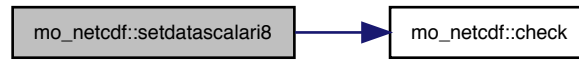
Here is the call graph for this function:

**15.48.2.109 setdatascalari8()**

```
subroutine mo_netcdf::setdatascalari8 (  
    class(ncvariable), intent(in) self,  
    integer(i1), intent(in) values,  
    integer(i4), dimension(:), intent(in), optional start )
```

References check().

Here is the call graph for this function:



15.48.2.110 `setdimension()`

```
type(ncdimension) function mo_netcdf::setdimension (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i4), intent(in) length )
```

References `check()`.

Here is the call graph for this function:

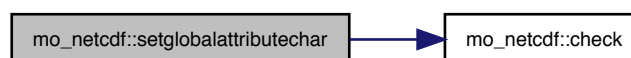


15.48.2.111 `setglobalattributechar()`

```
subroutine mo_netcdf::setglobalattributechar (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    character(*), intent(in) data )
```

References `check()`.

Here is the call graph for this function:

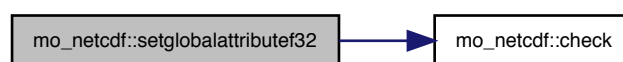


15.48.2.112 setglobalattribuf32()

```
subroutine mo_netcdf::setglobalattribuf32 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    real(sp), intent(in) data )
```

References check().

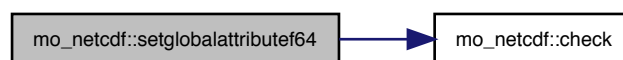
Here is the call graph for this function:

**15.48.2.113 setglobalattribuf64()**

```
subroutine mo_netcdf::setglobalattribuf64 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    real(dp), intent(in) data )
```

References check().

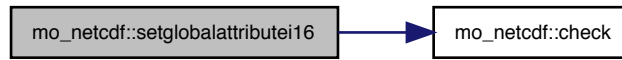
Here is the call graph for this function:

**15.48.2.114 setglobalattributei16()**

```
subroutine mo_netcdf::setglobalattributei16 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i2), intent(in) data )
```

References check().

Here is the call graph for this function:

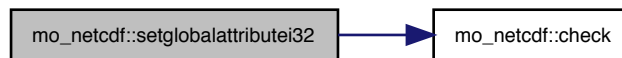


15.48.2.115 setglobalattributei32()

```
subroutine mo_netcdf::setglobalattributei32 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i4), intent(in) data )
```

References check().

Here is the call graph for this function:



15.48.2.116 setglobalattributei8()

```
subroutine mo_netcdf::setglobalattributei8 (  
    class(ncdataset), intent(in) self,  
    character(*), intent(in) name,  
    integer(i1), intent(in) data )
```

References check().

Here is the call graph for this function:

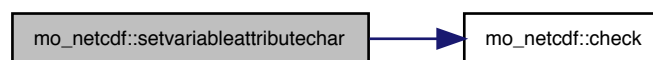


15.48.2.117 setvariableattributechar()

```
subroutine mo_netcdf::setvariableattributechar (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    character(*), intent(in) data )
```

References check().

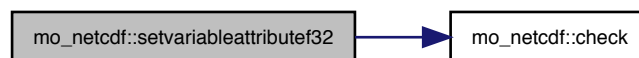
Here is the call graph for this function:

**15.48.2.118 setvariableattributef32()**

```
subroutine mo_netcdf::setvariableattributef32 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    real(sp), intent(in) data )
```

References check().

Here is the call graph for this function:

**15.48.2.119 setvariableattributef64()**

```
subroutine mo_netcdf::setvariableattributef64 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    real(dp), intent(in) data )
```

References check().

Here is the call graph for this function:

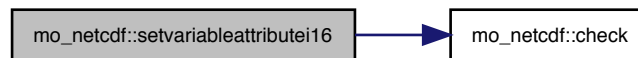


15.48.2.120 `setvariableattributei16()`

```
subroutine mo_netcdf::setvariableattributei16 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i2), intent(in) data )
```

References `check()`.

Here is the call graph for this function:

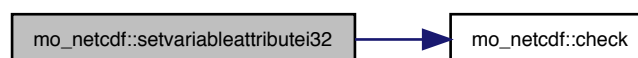


15.48.2.121 `setvariableattributei32()`

```
subroutine mo_netcdf::setvariableattributei32 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i4), intent(in) data )
```

References `check()`.

Here is the call graph for this function:

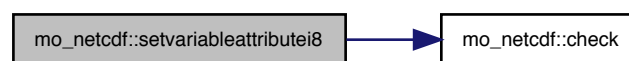


15.48.2.122 setvariableattributei8()

```
subroutine mo_netcdf::setvariableattributei8 (  
    class(ncvariable), intent(in) self,  
    character(*), intent(in) name,  
    integer(i1), intent(in) data )
```

References check().

Here is the call graph for this function:

**15.48.2.123 setvariablefillvaluef32()**

```
subroutine mo_netcdf::setvariablefillvaluef32 (  
    class(ncvariable), intent(in) self,  
    real(sp), intent(in) fvalue )
```

15.48.2.124 setvariablefillvaluef64()

```
subroutine mo_netcdf::setvariablefillvaluef64 (  
    class(ncvariable), intent(in) self,  
    real(dp), intent(in) fvalue )
```

15.48.2.125 setvariablefillvaluei16()

```
subroutine mo_netcdf::setvariablefillvaluei16 (  
    class(ncvariable), intent(in) self,  
    integer(i2), intent(in) fvalue )
```

15.48.2.126 setvariablefillvaluei32()

```
subroutine mo_netcdf::setvariablefillvaluei32 (  
    class(ncvariable), intent(in) self,  
    integer(i4), intent(in) fvalue )
```

15.48.2.127 setvariablefillvaluei8()

```

subroutine mo_netcdf::setvariablefillvaluei8 (
    class(ncvariable), intent(in) self,
    integer(i1), intent(in) fvalue )

```

15.48.2.128 setvariablewithids()

```

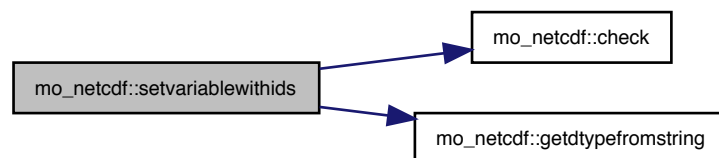
type(ncvariable) function mo_netcdf::setvariablewithids (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name,
    character(3), intent(in) dtype,
    integer(i4), dimension(:), intent(in) dimensions,
    logical, intent(in), optional contiguous,
    integer(i4), dimension(:), intent(in), optional chunksizes,
    integer(i4), intent(in), optional deflate_level,
    logical, intent(in), optional shuffle,
    logical, intent(in), optional fletcher32,
    integer(i4), intent(in), optional endianness,
    integer(i4), intent(in), optional cache_size,
    integer(i4), intent(in), optional cache_nelems,
    integer(i4), intent(in), optional cache_preemption )

```

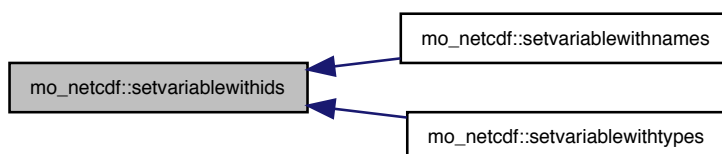
References `check()`, and `getdtypefromstring()`.

Referenced by `setvariablewithnames()`, and `setvariablewithtypes()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.48.2.129 setvariablewithnames()

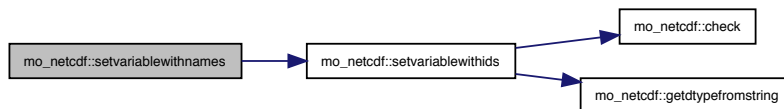
```

type(ncvariable) function mo_netcdf::setvariablewithnames (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name,
    character(3), intent(in) dtype,
    character(*), dimension(:), intent(in) dimensions,
    logical, intent(in), optional contiguous,
    integer(i4), dimension(:), intent(in), optional chunksizes,
    integer(i4), intent(in), optional deflate_level,
    logical, intent(in), optional shuffle,
    logical, intent(in), optional fletcher32,
    integer(i4), intent(in), optional endianness,
    integer(i4), intent(in), optional cache_size,
    integer(i4), intent(in), optional cache_nelems,
    integer(i4), intent(in), optional cache_preemption )

```

References setvariablewithids().

Here is the call graph for this function:



15.48.2.130 setvariablewithtypes()

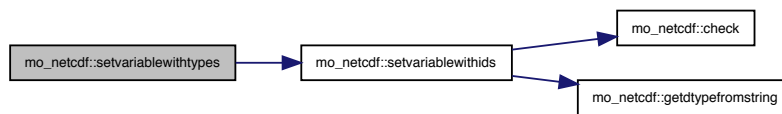
```

type(ncvariable) function mo_netcdf::setvariablewithtypes (
    class(ncdataset), intent(in) self,
    character(*), intent(in) name,
    character(3), intent(in) dtype,
    type(ncdimension), dimension(:), intent(in) dimensions,
    logical, intent(in), optional contiguous,
    integer(i4), dimension(:), intent(in), optional chunksizes,
    integer(i4), intent(in), optional deflate_level,
    logical, intent(in), optional shuffle,
    logical, intent(in), optional fletcher32,
    integer(i4), intent(in), optional endianness,
    integer(i4), intent(in), optional cache_size,
    integer(i4), intent(in), optional cache_nelems,
    integer(i4), intent(in), optional cache_preemption )

```

References setvariablewithids().

Here is the call graph for this function:



15.49 mo_neutrons Module Reference

THIS MODULE IS WORK IN PROGRESS, DO NOT USE FOR RESEARCH.

Functions/Subroutines

- subroutine, public [desiletsn0](#) (SoilMoisture, Horizons, N0, neutrons)
Calculate neutrons from soil moisture in the first layer.
- subroutine, public [cosmic](#) (SoilMoisture, Horizons, params, neutron_integral_AFast, neutrons)
Calculate neutrons from soil moisture in all layers.
- subroutine [oldintegration](#) (res, c)
- subroutine, public [tabularintegralafast](#) (integral, maxC)
Save approximation data for A_fast.
- subroutine [approx_mon_int](#) (res, f, c, xmin, xmax, eps, steps, fxmin, fxmax)
- recursive subroutine [approx_mon_int_steps](#) (res, f, c, xmin, xmax, eps, steps, fxmin, fxmax)
- recursive subroutine [approx_mon_int_eps](#) (res, f, c, xmin, xmax, eps, fxmin, fxmax)
- subroutine [lookupintegral](#) (res, integral, c)
- real(dp) function [intgrandfast](#) (c, phi)

15.49.1 Detailed Description

THIS MODULE IS WORK IN PROGRESS, DO NOT USE FOR RESEARCH.

The number of neutrons above the ground is directly related to the number soil water content in the ground, air, vegetation and/or snow. This module forward-models neutron abundance as a state variable for each cell.

Authors

Martin Schroen

Date

Mar 2015

15.49.2 Function/Subroutine Documentation

15.49.2.1 approx_mon_int()

```

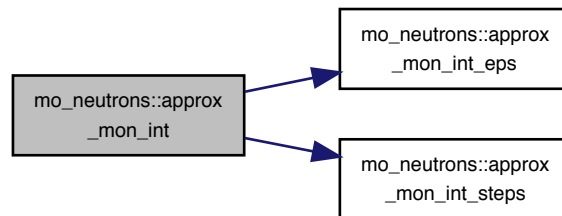
subroutine mo_neutrons::approx_mon_int (
    real(dp) res,
    real(dp), external f,
    real(dp), intent(in) c,
    real(dp), intent(in) xmin,
    real(dp), intent(in) xmax,
    real(dp), optional eps,
    integer(i4), optional steps,
    real(dp), optional fxmin,
    real(dp), optional fxmax )

```

References approx_mon_int_eps(), and approx_mon_int_steps().

Referenced by tabularintegralafast().

Here is the call graph for this function:



Here is the caller graph for this function:



15.49.2.2 approx_mon_int_eps()

```

recursive subroutine mo_neutrons::approx_mon_int_eps (
    real(dp) res,
    real(dp), external f,
    real(dp), intent(in) c,
    real(dp), intent(in) xmin,
    real(dp), intent(in) xmax,
    real(dp), intent(in) eps,
    real(dp), intent(in) fxmin,
    real(dp), intent(in) fxmax ) [private]

```

Referenced by approx_mon_int().

Here is the caller graph for this function:



15.49.2.3 approx_mon_int_steps()

```

recursive subroutine mo_neutrons::approx_mon_int_steps (
    real(dp) res,
    real(dp), external f,
    real(dp), intent(in) c,
    real(dp), intent(in) xmin,
    real(dp), intent(in) xmax,
    real(dp), intent(in) eps,
    integer(i4), intent(in) steps,
    real(dp), intent(in) fxmin,
    real(dp), intent(in) fxmax ) [private]
  
```

Referenced by approx_mon_int().

Here is the caller graph for this function:



15.49.2.4 cosmic()

```

subroutine, public mo_neutrons::cosmic (
    real(dp), dimension(:), intent(in) SoilMoisture,
    real(dp), dimension(:), intent(in) Horizons,
    real(dp), dimension(:), intent(in) params,
    real(dp), dimension(:), intent(in) neutron_integral_AFast,
    real(dp), intent(inout) neutrons )
  
```

Calculate neutrons from soil moisture in all layers.

Neutron counts above the ground (one value per cell in mHM) can be derived by a simplified physical neutron transport simulation. Fast cosmic-Ray neutrons are generated in the soil and attenuated differently in water and soil. The remaining neutrons that reached the surface relate to the profile of soil water content below. Variables like N, alpha and L3 are site-specific and need to be calibrated.

Parameters

in	<i>real(dp), dimension(:, :) :: SoilMoisture</i>	Soil Moisture
in	<i>real(dp), dimension(:) :: Horizons</i>	Horizon depths
in	<i>real(dp), dimension(:) :: params</i>	! N0, N1, N2, alpha0, alpha1, L30, L31
in	<i>real(dp), dimension(:) :: neutron_integral_AFast</i>	Tabular for Int Approx
out	<i>real(dp), dimension(size(SoilMoisture,1)) :: neutrons</i>	Neutron counts

Author

Martin Schroen, originally written by Rafael Rosolem

Date

Mar 2015

References mo_mhm_constants::cosmic_alpha, mo_mhm_constants::cosmic_bd, mo_mhm_constants::cosmic_l1, mo_mhm_constants::cosmic_l2, mo_mhm_constants::cosmic_l3, mo_mhm_constants::cosmic_l4, mo_mhm_constants::cosmic_n, mo_mhm_constants::cosmic_vwclat, mo_mhm_constants::h2odens, lookupintegral(), and mo_constants::pi_dp.

Referenced by mo_mhm::mhm().

Here is the call graph for this function:



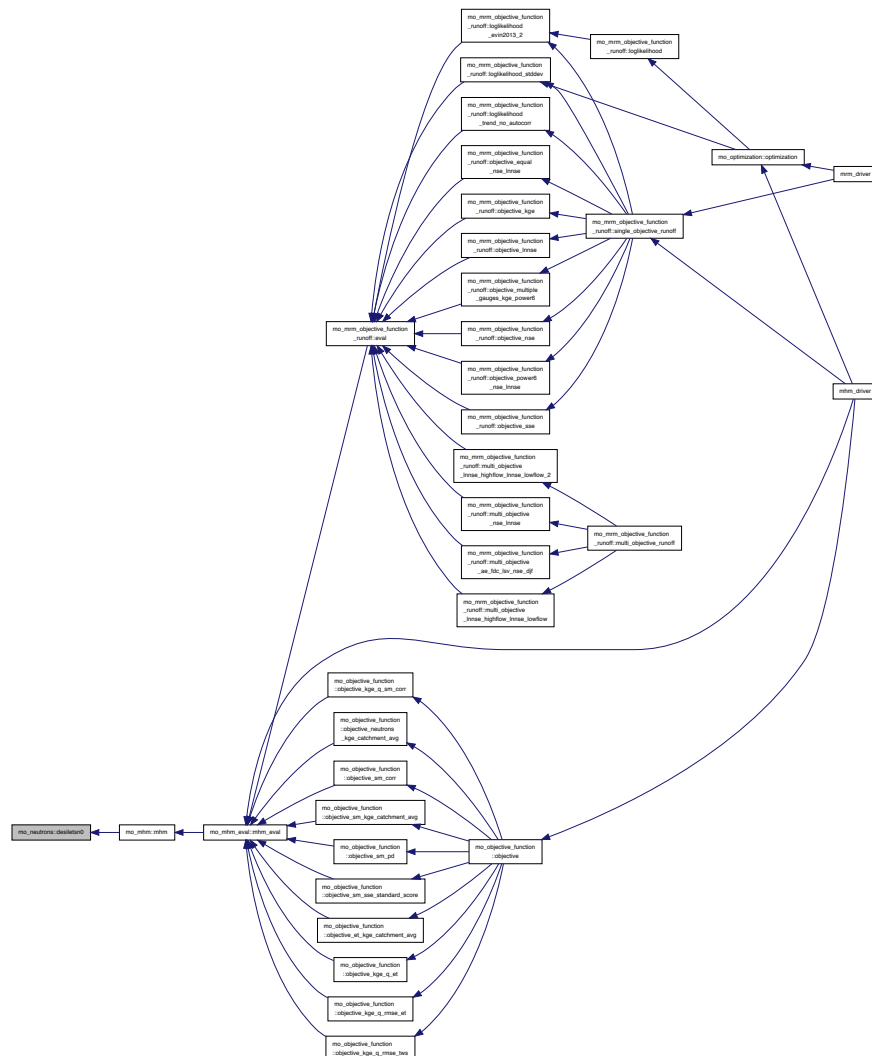
in	<i>real(dp), dimension(:) :: Horizons</i>	Horizon depths
in	<i>real(dp) :: N0</i>	dry neutron counts
out	<i>real(dp), dimension(size(SoilMoisture,1)) :: neutrons</i>	Neutron counts

Martin Schroen

Mar 2015

Referenced by `mo_mhm::mhm()`.

Here is the caller graph for this function:



15.49.2.6 `intgrandfast()`

```
real(dp) function mo_neutrons::intgrandfast (
    real(dp), intent(in) c,
    real(dp), intent(in) phi )
```

Referenced by `tabularintegralfast()`.

Here is the caller graph for this function:



15.49.2.7 `lookupintegral()`

```
subroutine mo_neutrons::lookupintegral (
    real(dp) res,
    real(dp), dimension(:), intent(in) integral,
    real(dp), intent(in) c ) [private]
```

References `mo_kind::i4`, and `mo_constants::pi_dp`.

Referenced by `cosmic()`.

[illegible]

```

subroutine mo_neutrons::oldintegration (
    real(dp) res,
    real(dp), intent(in) c )

```

```

subroutine, public mo_neutrons::tabularintegralafast (
    real(dp), dimension(:)  integral,
    real(dp), intent(in) maxC )

```

The COSMIC subroutine needs A_{fast} to be calculated. $A_{\text{fast}} = \int_0^{\pi/2} \exp(-\Lambda_{\text{fast}}(z)/\cos(\phi)) d\phi$. This subroutine stores data for intsize values for $c = \Lambda_{\text{fast}}(z)$ between 0 and maxC, and will be written into the global array variable neutron_integral_AFast. The calculation of the values is done with a very precise recursive

approximation subroutine. That recursive subroutine should not be used inside the time, cells and layer loops, because it is slow. Inside the loops in the module COSMIC the tabular is used to estimate A_fast, if $0 < c < \text{maxC}$, otherwise the recursive approximation is used.

Parameters

in	<i>real(dp), dimension(:, :) :: SoilMoisture</i>	Soil Moisture
in	<i>real(dp), dimension(:) :: Horizons</i>	Horizon depths
in	<i>real(dp), dimension(:) :: params</i>	! N0, N1, N2, alpha0, alpha1, L30, L31
in	<i>integer(i4) :: intsize</i>	! number of values for the approximation
in	<i>real(dp) :: maxC</i>	! maximum value for A_fast
out	<i>real(dp), dimension(intsize) :: neutron_integral_AFast</i>	approximation values

Author

Maren Kaluza

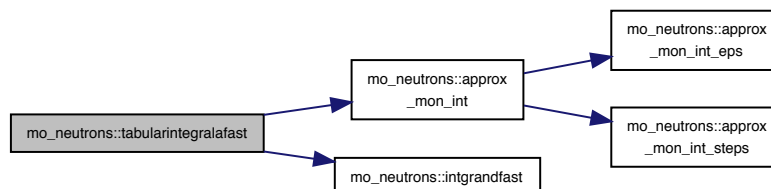
Date

Nov 2017

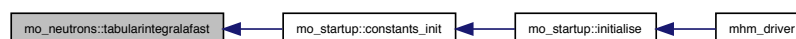
References approx_mon_int(), intgrandfast(), and mo_constants::pi_dp.

Referenced by mo_startup::constants_init().

Here is the call graph for this function:



Here is the caller graph for this function:



15.50 mo_nml Module Reference

Deal with namelist files.

Functions/Subroutines

- subroutine, public `open_nml` (file, unit, quiet)
Open a namelist file.
- subroutine, public `close_nml` (unit)
Close a namelist file.
- subroutine, public `position_nml` (name, unit, status, first)
Position a namelist file.

Variables

- integer(i4), parameter, public `positioned` = 0
Information: file pointer set to namelist group.
- integer(i4), parameter, public `missing` = 1
Error: namelist group is missing.
- integer(i4), parameter, public `length_error` = 2
Error: namelist group name too long.
- integer(i4), parameter, public `read_error` = 3
Error occurred during read of namelist file.
- integer, save, public `nunitnml` = -1

15.50.1 Detailed Description

Deal with namelist files.

This module provides routines to open, close and position namelist files.

Authors

Matthias Cuntz

Date

Jan 2011

15.50.2 Function/Subroutine Documentation

15.50.2.1 `close_nml()`

```
subroutine, public mo_nml::close_nml (
    integer, intent(in), optional unit )
```

Close a namelist file.

Parameters

<code>in</code>	<code>integer, optional :: unit</code>	namelist unit
-----------------	--	---------------

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

Date

Dec 2011

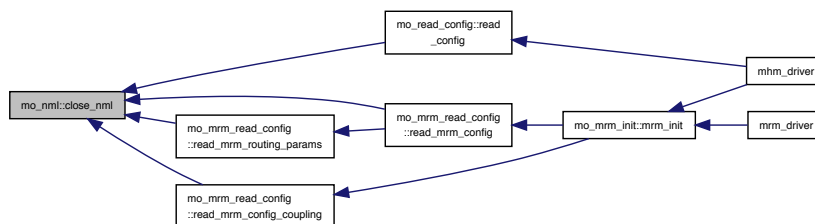
References `mo_finish::finish()`, and `nunitnml`.

Referenced by `mo_read_config::read_config()`, `mo_mrm_read_config::read_mrm_config()`, `mo_mrm_read_config::read_mrm_config_coupling()`, and `mo_mrm_read_config::read_mrm_routing_params()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.50.2.2 open_nml()

```

subroutine, public mo_nml::open_nml (
    character(len=*), intent(in) file,
    integer, intent(in) unit,
    logical, intent(in), optional quiet )
  
```

Open a namelist file.

Parameters

in	<i>character(len=*) :: file</i>	namelist filename
in	<i>integer :: unit</i>	namelist unit
in	<i>logical, optional :: logical :: quiet</i>	Be verbose or not (default: .true.) .true.: no messages .false.: write out messages

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

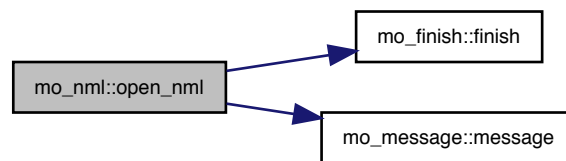
Date

Dec 2011

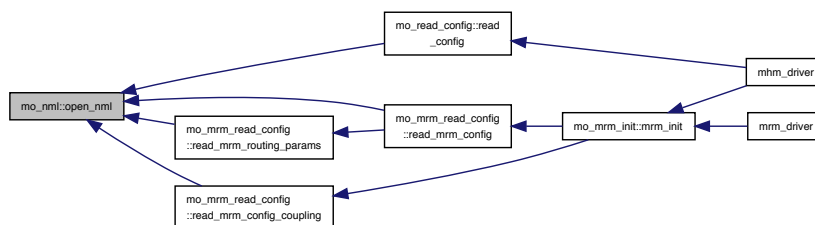
References mo_finish::finish(), mo_message::message(), mo_message::message_text, and nunitnml.

Referenced by mo_read_config::read_config(), mo_mrm_read_config::read_mrm_config(), mo_mrm_read_config::read_mrm_config_coupling(), and mo_mrm_read_config::read_mrm_routing_params().

Here is the call graph for this function:



Here is the caller graph for this function:

**15.50.2.3 position_nml()**

```

subroutine, public mo_nml::position_nml (
    character(len=*), intent(in) name,
    integer, intent(in), optional unit,
    integer(i4), intent(out), optional status,
    logical, intent(in), optional first )
  
```

Position a namelist file.

Position namelist file pointer for reading a new namelist next.

It positions the namelist file at the correct place for reading namelist /name/ (case independent).

Parameters

in	<i>character(len=*) :: name</i>	namelist name (case independent)
in	<i>integer, optional :: unit</i>	namelist unit (default: nunitnml)
in	<i>logical, optional :: first</i>	start search at beginning, i.e. rewind the namelist first (default: .true.) .true.: rewind .false.: continue search from current file pointer
out	<i>integer(i4), optional :: status</i>	Set on output to either of POSITIONED (0) - correct MISSING (1) - name not found LENGTH_ERROR (2) - namelist length longer then 256 characters READ_ERROR (3) - error while reading namelist file

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

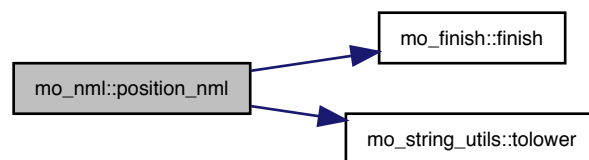
Date

Dec 2011

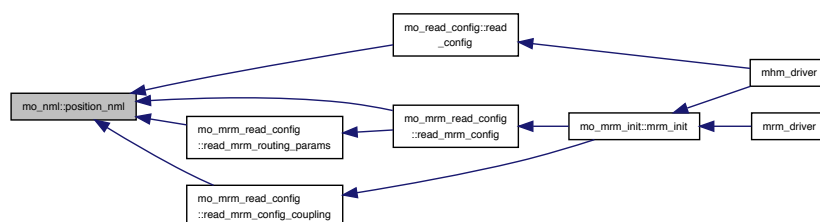
References `mo_finish::finish()`, `length_error`, `mo_message::message_text`, `missing`, `nunitnml`, `positioned`, `read_error`, and `mo_string_utils::tolower()`.

Referenced by `mo_read_config::read_config()`, `mo_mrm_read_config::read_mrm_config()`, `mo_mrm_read_config::read_mrm_config_coupling()`, and `mo_mrm_read_config::read_mrm_routing_params()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.50.3 Variable Documentation

15.50.3.1 length_error

```
integer(i4), parameter, public mo_nml::length_error = 2
```

Error: namelist group name too long.

Referenced by position_nml().

15.50.3.2 missing

```
integer(i4), parameter, public mo_nml::missing = 1
```

Error: namelist group is missing.

Referenced by position_nml().

15.50.3.3 nunitnml

```
integer, save, public mo_nml::nunitnml = -1
```

Referenced by close_nml(), open_nml(), and position_nml().

15.50.3.4 positioned

```
integer(i4), parameter, public mo_nml::positioned = 0
```

Information: file pointer set to namelist group.

Referenced by position_nml().

15.50.3.5 read_error

```
integer(i4), parameter, public mo_nml::read_error = 3
```

Error occurred during read of namelist file.

Referenced by position_nml().

15.51 mo_objective_function Module Reference

Objective Functions for Optimization of mHM.

Functions/Subroutines

- real(dp) function, public [objective](#) (parameterset)
Wrapper for objective functions.

- real(dp) function [objective_sm_kge_catchment_avg](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [objective_sm_corr](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [objective_sm_pd](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [objective_sm_sse_standard_score](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [objective_kge_q_rmse_tws](#) (parameterset)
Objective function of KGE for runoff and RMSE for basin_avg TWS (standarized scores)
- real(dp) function [objective_neutrons_kge_catchment_avg](#) (parameterset)
Objective function for neutrons.
- real(dp) function [objective_et_kge_catchment_avg](#) (parameterset)
Objective function for evapotranspiration (et).
- real(dp) function [objective_kge_q_sm_corr](#) (parameterset)
Objective function of KGE for runoff and correlation for SM.
- real(dp) function [objective_kge_q_et](#) (parameterset)
Objective function of KGE for runoff and KGE for ET.
- real(dp) function [objective_kge_q_rmse_et](#) (parameterset)
Objective function of KGE for runoff and RMSE for basin_avg ET (standarized scores)
- subroutine [extract_basin_avg_tws](#) (basinId, tws, tws_sim, tws_obs, tws_obs_mask)
extracts basin average tws data from global variables

15.51.1 Detailed Description

Objective Functions for Optimization of mHM.

This module provides a wrapper for several objective functions used to optimize mHM against various variables. If the objective is only regarding runoff move it to [mRM/mo_mrm_objective_function_runoff.f90](#). If it contains besides runoff another variable like TWS implement it here.

All the objective functions are supposed to be minimized!

(10) SO: SM: 1.0 - KGE of catchment average soilmoisture

(11) SO: SM: 1.0 - Pattern dissimilarity (PD) of spatially distributed soil moisture

(12) SO: SM: Sum of squared errors (SSE) of spatially distributed standard score (normalization) of soil moisture

(13) SO: SM: 1.0 - average temporal correlation of spatially distributed soil moisture

(15) SO: Q + TWS: $[1.0 - KGE(Q)] * RMSE(basin_avg_TWS)$ - objective function using Q and basin average (standard score) TWS

(17) SO: N: 1.0 - KGE of spatio-temporal neutron data, catchment-average

(27) SO: ET: 1.0 - KGE of catchment average evapotranspiration

Authors

Juliane Mai

Date

Dec 2012

15.51.2 Function/Subroutine Documentation

15.51.2.1 extract_basin_avg_tws()

```

subroutine mo_objective_function::extract_basin_avg_tws (
    integer(i4), intent(in) basinId,
    real(dp), dimension(:,:), intent(in) tws,
    real(dp), dimension(:), intent(out), allocatable tws_sim,
    real(dp), dimension(:), intent(out), allocatable tws_obs,
    logical, dimension(:), intent(out), allocatable tws_obs_mask )

```

extracts basin average tws data from global variables

extracts simulated and measured basin average tws from global variables, such that they overlay exactly. For measured tws, only the tws during the evaluation period are cut, not succeeding nodata values. For simulated tws, warming days as well as succeeding nodata values are neglected.

Parameters

in	<i>integer(i4) :: basinID</i>	- ID of the current basin to process
in	<i>real(dp) :: tws(:)</i>	- simulated basin average tws
out	<i>real(dp) :: tws_sim(:)</i>	- simulated basin average tws at this basin
out	<i>real(dp) :: tws_obs(:)</i>	- extracted observed basin average tws
out	<i>logical :: tws_obs_mask(:)</i>	- masking non-negative values in tws_obs

Author

O. Rakovec

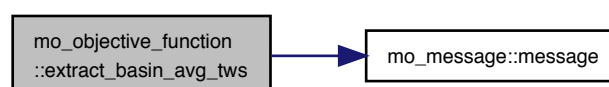
Date

Oct 2015

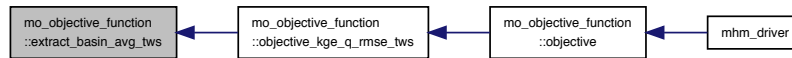
References mo_global_variables::basin_avg_tws_obs, mo_constants::eps_dp, mo_global_variables::evalper, mo_message::message(), mo_global_variables::nmeasperday_tws, mo_mrm_constants::nodata_dp, mo_global_variables::ntstepday, and mo_global_variables::warmingdays.

Referenced by objective_kge_q_rmse_tws().

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.2 objective()

```
real(dp) function, public mo_objective_function::objective (
    real(dp), dimension(:), intent(in) parameterset )
```

Wrapper for objective functions.

The functions selects the objective function case defined in a namelist, i.e. the global variable *opti_function*. It return the objective function value for a specific parameter set.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points. Numbering of objective functions has to be consistent with that of [mo_mrm_objective_function_runoff](#)

Author

Juliane Mai

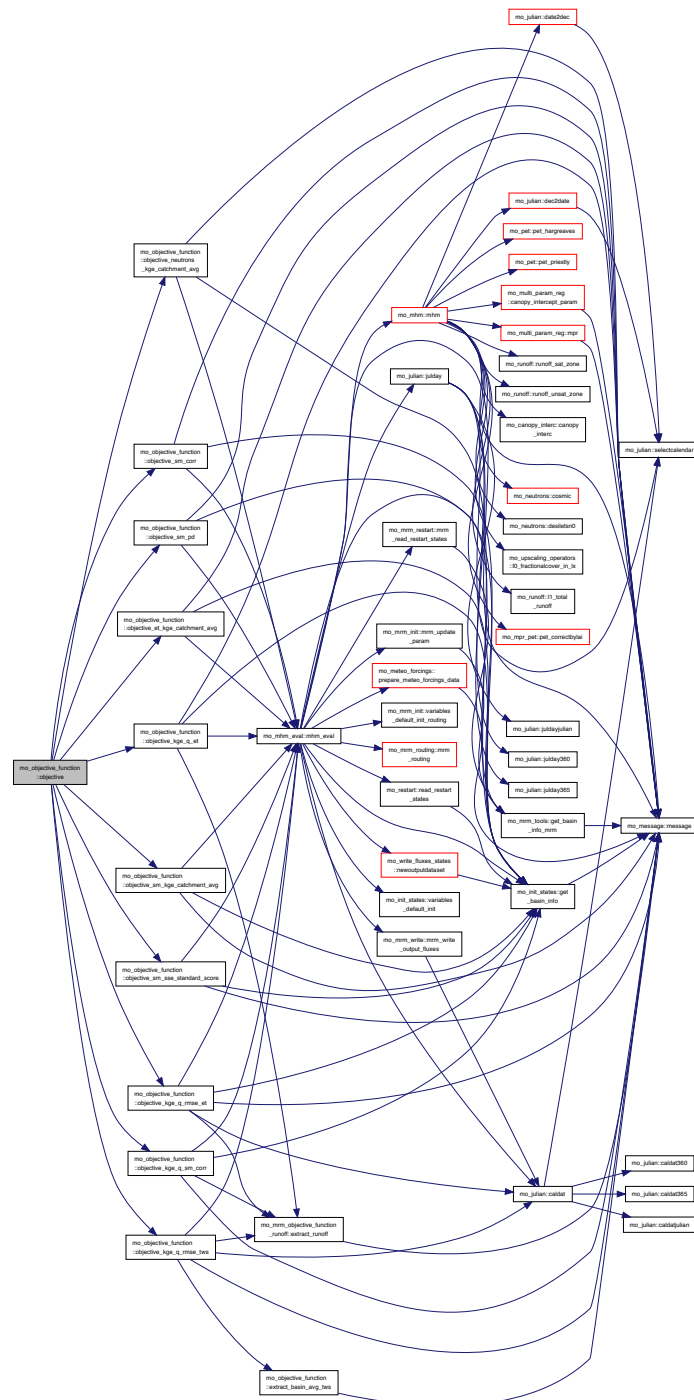
Date

Dec 2012

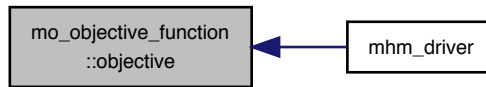
References [objective_et_kge_catchment_avg\(\)](#), [objective_kge_q_et\(\)](#), [objective_kge_q_rmse_et\(\)](#), [objective_kge_q_rmse_tws\(\)](#), [objective_kge_q_sm_corr\(\)](#), [objective_neutrons_kge_catchment_avg\(\)](#), [objective_sm_corr\(\)](#), [objective_sm_kge_catchment_avg\(\)](#), [objective_sm_pd\(\)](#), [objective_sm_sse_standard_score\(\)](#), and [mo_common_variables::opti_function](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.3 objective_et_kge_catchment_avg()

```
real(dp) function mo_objective_function::objective_et_kge_catchment_avg (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for evapotranspiration (et).

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore, the Kling-Gupta model efficiency KGE of the catchment average evapotranspiration (et) is calculated

$$KGE = 1.0 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment correlation coefficient

α = ratio of simulated mean to observed mean SM

β = ratio of simulated standard deviation to observed standard deviation

is calculated and the objective function for a given basin i is

$$\phi_i = 1.0 - KGE_i$$

ϕ_i is the objective since we always apply minimization methods. The minimal value of ϕ_i is 0 for the optimal KGE of 1.0.

Finally, the overall objective function value OF is estimated based on the power-6 norm to combine the ϕ_i from all basins N .

$$OF = \sqrt[6]{\sum ((1.0 - KGE_i)/N)^6}.$$

The observed data `L1_et`, `L1_et_mask` are global in this module.

Parameters

in	<code>real(dp) :: parameterset(:)</code>	1D-array with parameters the model is run with
----	--	--

Returns

`real(dp) :: objective_et_kge_catchment_avg` — objective function value (which will be e.g. minimized by an optimization routine)

Note

Input values must be floating points.

Author

Johannes Brenner

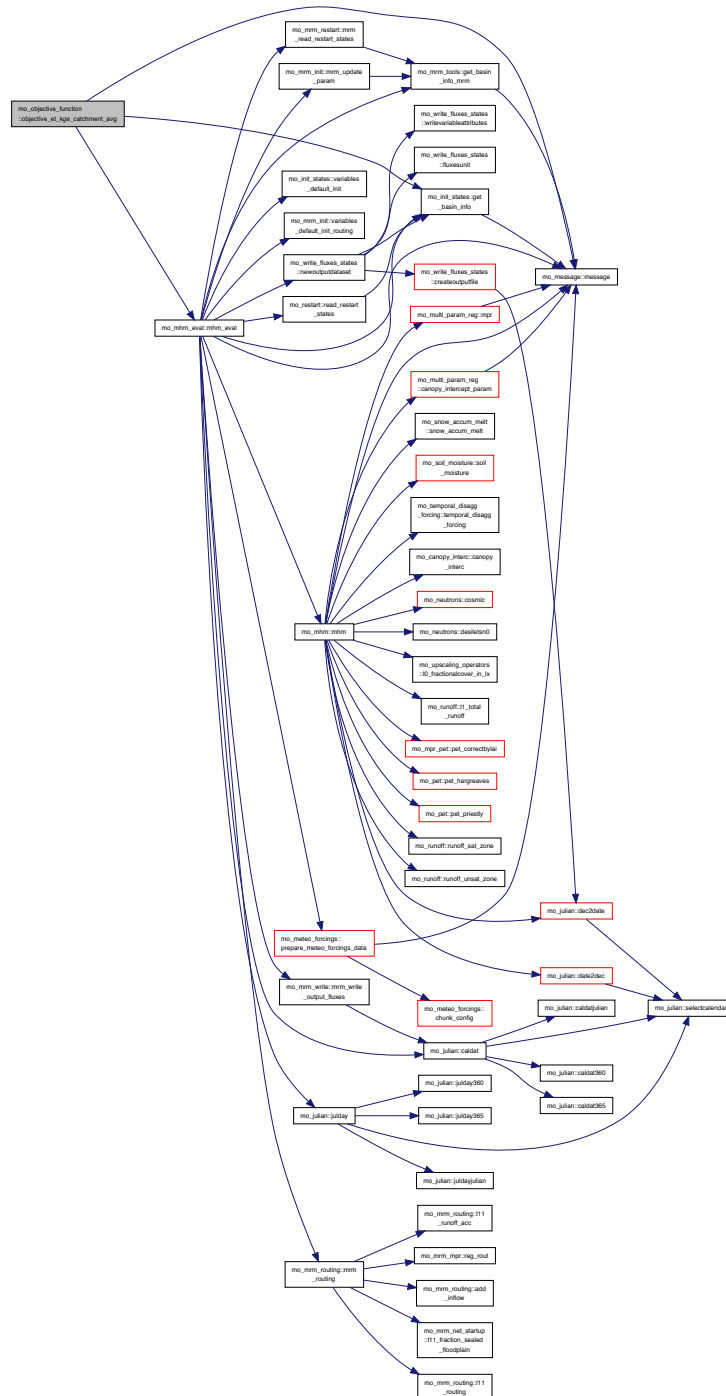
Date

Feb 2017

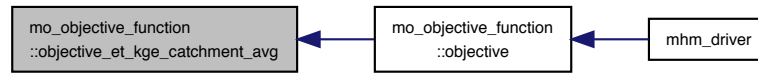
References mo_init_states::get_basin_info(), mo_message::message(), mo_mhm_eval::mhm_eval(), mo_global↵
_variables::nbasins, and mo_mhm_constants::nodata_dp.

Referenced by objective().

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.4 objective_kge_q_et()

```

real(dp) function mo_objective_function::objective_kge_q_et (
    real(dp), dimension(:), intent(in) parameterset )
  
```

Objective function of KGE for runoff and KGE for ET.

Objective function of KGE for runoff and KGE for ET. Further details can be found in the documentation of objective functions '14 - objective_multiple_gauges_kge_power6'.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_kge_q_et — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Johannes Brenner

Date

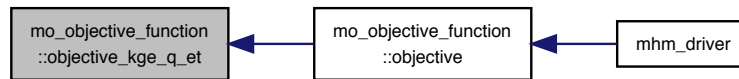
July 2017

References `mo_mrm_objective_function_runoff::extract_runoff()`, `mo_init_states::get_basin_info()`, `mo_global_variables::l1_et`, `mo_global_variables::l1_et_mask`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, and `mo_global_variables::nbasins`.

Referenced by `objective()`.



Here is the caller graph for this function:



15.51.2.5 objective_kge_q_rmse_et()

```

real(dp) function mo_objective_function::objective_kge_q_rmse_et (
    real(dp), dimension(:), intent(in) parameterset )
  
```

Objective function of KGE for runoff and RMSE for basin_avg ET (standardized scores)

Objective function of KGE for runoff and RMSE for basin_avg ET (standardized scores)

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_kge_q_rmse_et — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Actually, *KGE* will be returned such that it can be minimized.

For TWS required at least 5 years of data! Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." *Journal of Hydrology* 377.1 (2009): 80-91.

Author

Johannes Brenner

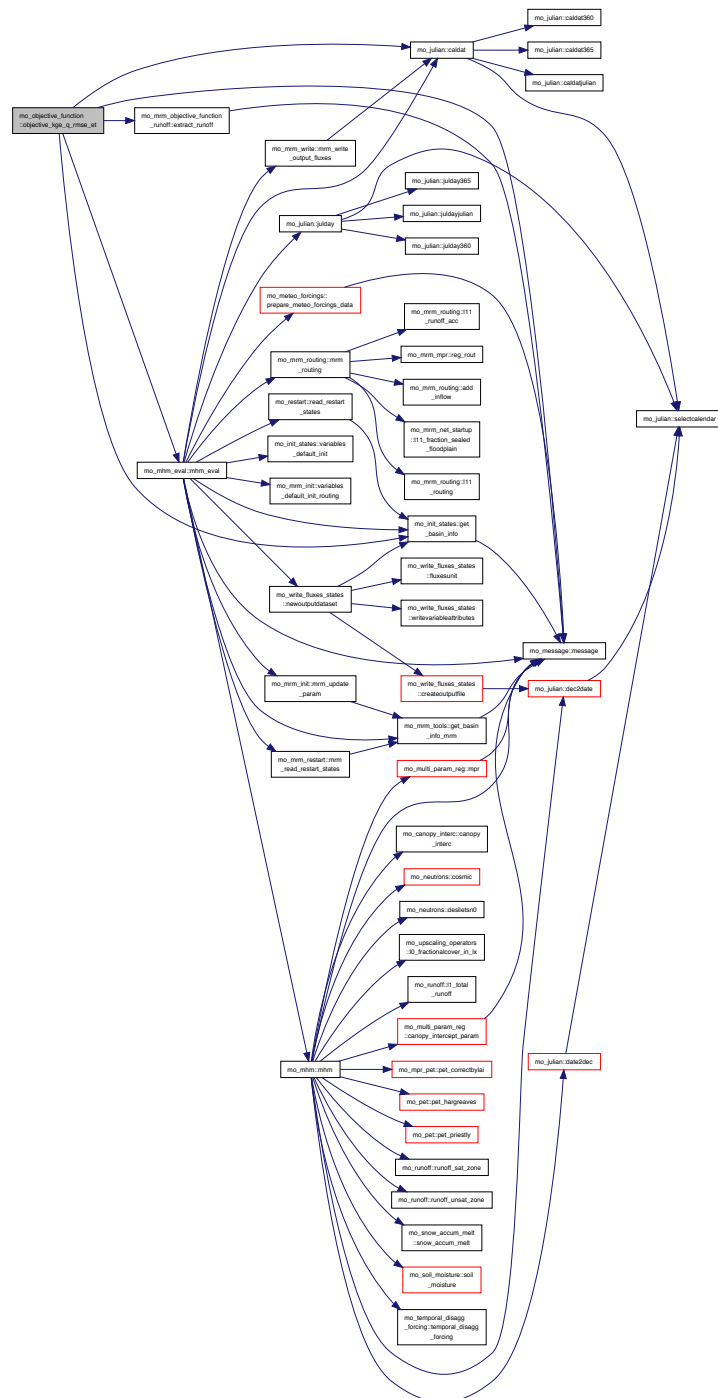
Date

July 2017

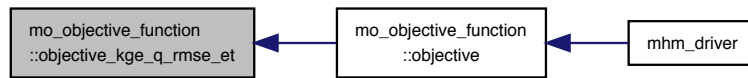
References `mo_julian::caldat()`, `mo_constants::eps_dp`, `mo_global_variables::evalper`, `mo_mrm_objective_function_runoff::extract_runoff()`, `mo_init_states::get_basin_info()`, `mo_global_variables::l1_et`, `mo_global_variables::l1_et_mask`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, `mo_global_variables::nbasins`, `mo_mrm_constants::nodata_dp`, and `mo_global_variables::timestep_et_input`.

Referenced by `objective()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.6 objective_kge_q_rmse_tws()

```
real(dp) function mo_objective_function::objective_kge_q_rmse_tws (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of KGE for runoff and RMSE for basin_avg TWS (standarized scores)

Objective function of KGE for runoff and RMSE for basin_avg TWS (standarized scores)

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_kge_q_rmse_tws — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Actually, *KGE* will be returned such that it can be minimized.

For TWS required at least 5 years of data! Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." *Journal of Hydrology* 377.1 (2009): 80-91.

Author

Oldrich Rakovec, Rohini Kumar

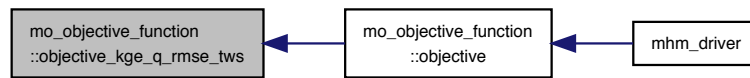
Date

Oct. 2015

References `mo_julian::caldat()`, `mo_constants::eps_dp`, `mo_global_variables::evalper`, `extract_basin_avg_tws()`, `mo_mrm_objective_function_runoff::extract_runoff()`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, `mo_global_variables::nbasins`, and `mo_mrm_constants::nodata_dp`.

Referenced by `objective()`.

Here is the caller graph for this function:



15.51.2.7 objective_kge_q_sm_corr()

```
real(dp) function mo_objective_function::objective_kge_q_sm_corr (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function of KGE for runoff and correlation for SM.

Objective function of KGE for runoff and SSE for soil moisture (standarized scores). Further details can be found in the documentation of objective functions '14 - objective_multiple_gauges_kge_power6' and '13 - objective_sm_corr'.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_kge_q_sse_sm — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Matthias Zink

Date

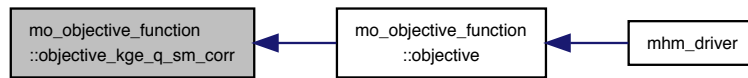
Mar. 2017

References `mo_mrm_objective_function_runoff::extract_runoff()`, `mo_init_states::get_basin_info()`, `mo_global_variables::l1_sm`, `mo_global_variables::l1_sm_mask`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, and `mo_global_variables::nbasins`.

Referenced by `objective()`.



Here is the caller graph for this function:



15.51.2.8 objective_neutrons_kge_catchment_avg()

```
real(dp) function mo_objective_function::objective_neutrons_kge_catchment_avg (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for neutrons.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore, the Kling-Gupta model efficiency KGE of the catchment average neutrons (N) is calculated

$$KGE = 1.0 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment CORRELATION coefficient

α = ratio of simulated mean to observed mean SM

β = ratio of simulated standard deviation to observed standard deviation

is calculated and the objective function for a given basin i is

$$\phi_i = 1.0 - KGE_i$$

ϕ_i is the objective since we always apply minimization methods. The minimal value of ϕ_i is 0 for the optimal KGE of 1.0.

Finally, the overall objective function value OF is estimated based on the power-6 norm to combine the ϕ_i from all basins N .

$$OF = \sqrt[6]{\sum ((1.0 - KGE_i)/N)^6}.$$

The observed data `L1_neutronsdata`, `L1_neutronsdata_mask` are global in this module.

Parameters

in	<code>real(dp) :: parameterset(:)</code>	1D-array with parameters the model is run with
----	--	--

Returns

`real(dp) :: objective_neutrons_kge_catchment_avg` — objective function value (which will be e.g. minimized by an optimization routine)

Note

Input values must be floating points.

Author

Martin Schroen

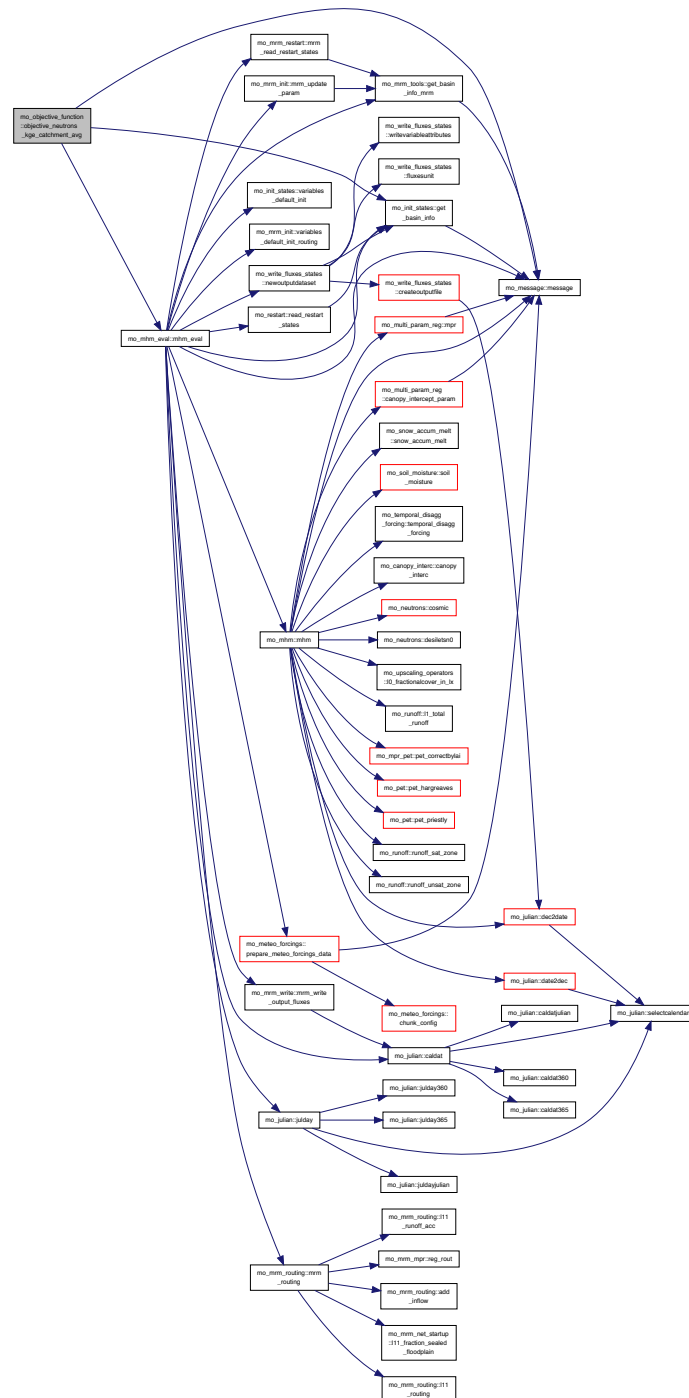
Date

Jun 2015

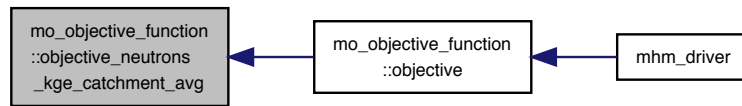
References `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, `mo_global←
_variables::nbasins`, and `mo_mhm_constants::nodata_dp`.

Referenced by `objective()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.9 objective_sm_corr()

```
real(dp) function mo_objective_function::objective_sm_corr (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for soil moisture.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore the Pearson correlation between observed and modeled soil moisture on each grid cell j is compared

$$r_j = r^2(SM_{obs}^j, SM_{sim}^j)$$

where

r^2 = Pearson correlation coefficient,

SM_{obs} = observed soil moisture,

SM_{sim} = simulated soil moisture.

The observed data SM_{obs} are global in this module.

The correlation is spatially averaged as

$$\phi_i = \frac{1}{K} \cdot \sum_{j=1}^K r_j$$

where K denotes the number of valid cells in the study domain.

Finally, the overall objective function value OF is estimated based on the power-6 norm to combine the ϕ_i from all basins N .

$$OF = \sqrt[6]{\sum ((1.0 - \phi_i)/N)^6}.$$

The observed data L1_sm, L1_sm_mask are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_sm_corr — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Matthias Zink

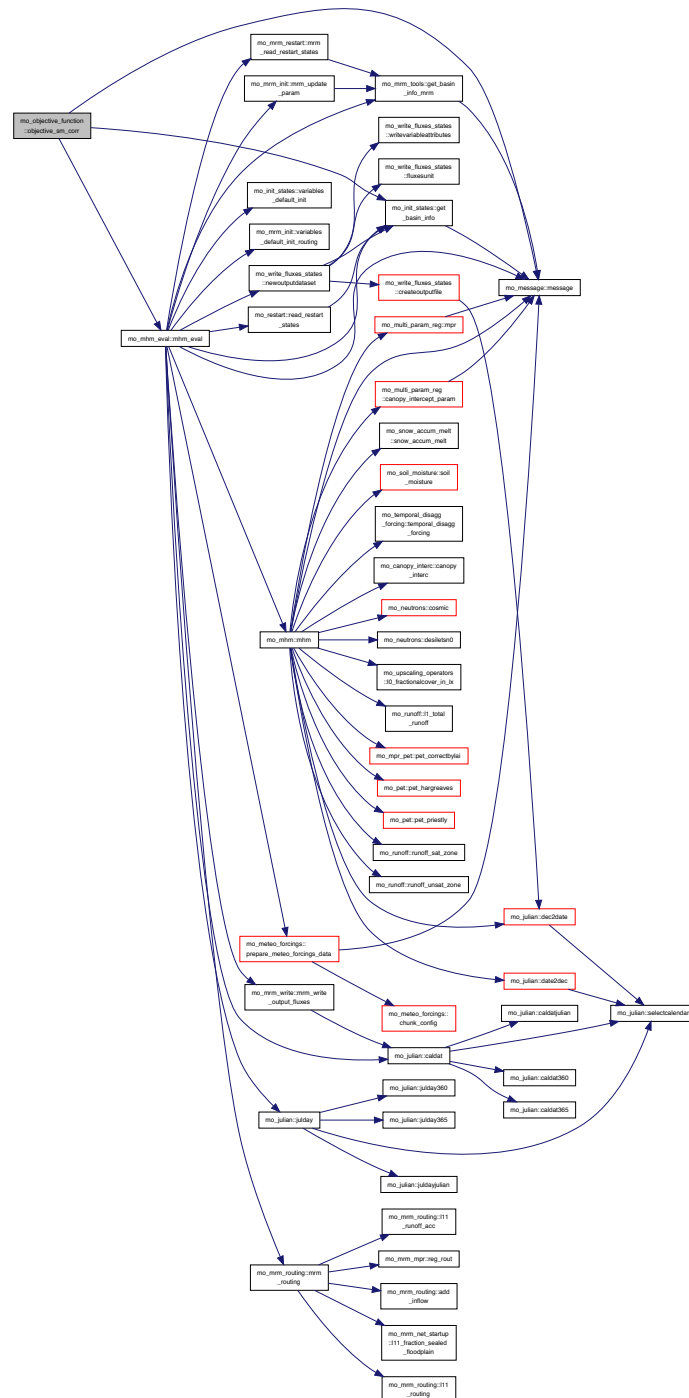
Date

March 2015

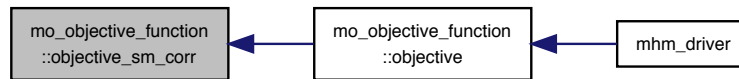
References mo_init_states::get_basin_info(), mo_message::message(), mo_mhm_eval::mhm_eval(), and mo_↵
global_variables::nbasins.

Referenced by objective().

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.10 objective_sm_kge_catchment_avg()

```
real(dp) function mo_objective_function::objective_sm_kge_catchment_avg (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for soil moisture.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore, the Kling-Gupta model efficiency KGE of the catchment average soil moisture (SM) is calculated

$$KGE = 1.0 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment correlation coefficient

α = ratio of simulated mean to observed mean SM

β = ratio of simulated standard deviation to observed standard deviation

is calculated and the objective function for a given basin i is

$$\phi_i = 1.0 - KGE_i$$

ϕ_i is the objective since we always apply minimization methods. The minimal value of ϕ_i is 0 for the optimal KGE of 1.0.

Finally, the overall objective function value OF is estimated based on the power-6 norm to combine the ϕ_i from all basins N .

$$OF = \sqrt[6]{\sum ((1.0 - KGE_i)/N)^6}.$$

The observed data `L1_sm`, `L1_sm_mask` are global in this module.

Parameters

in	<code>real(dp) :: parameterset(:)</code>	1D-array with parameters the model is run with
----	--	--

Returns

`real(dp) :: objective_sm_kge_catchment_avg` — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Matthias Zink

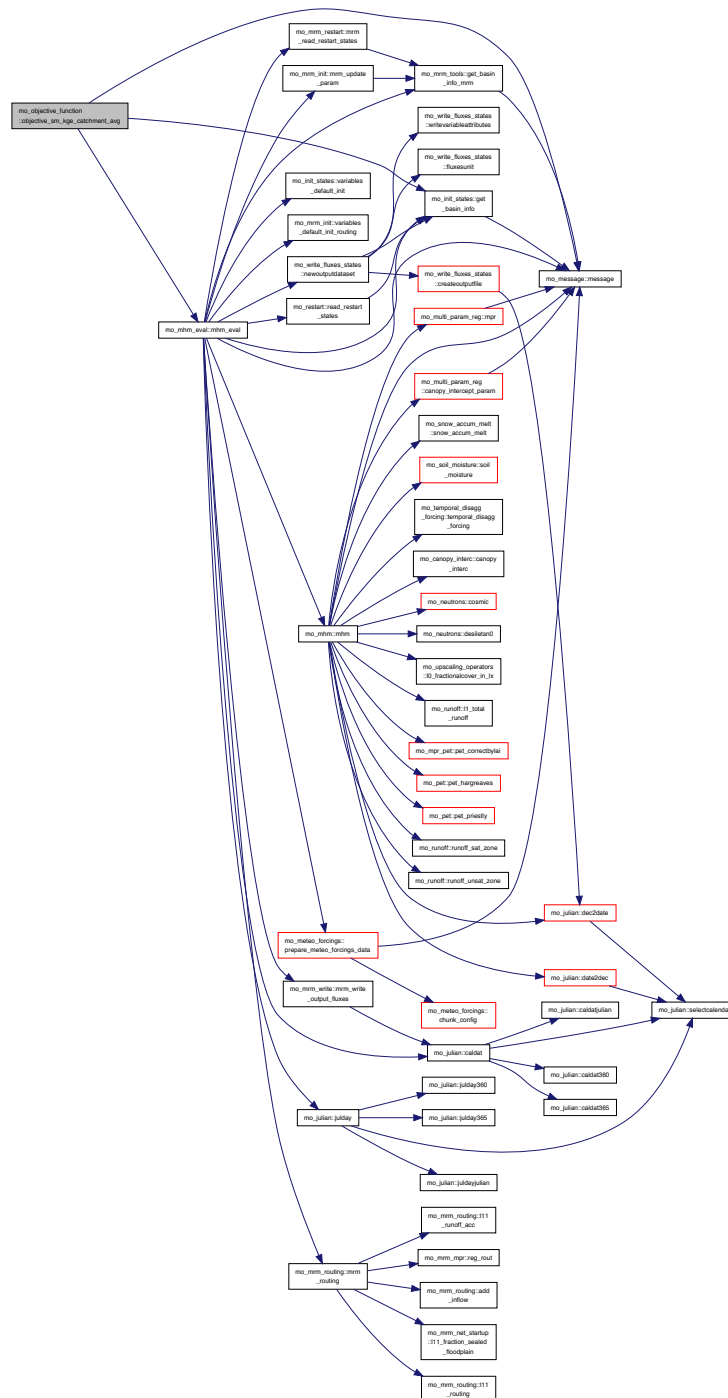
Date

May 2015

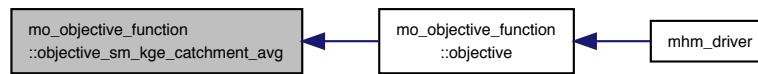
References `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, `mo_global←_variables::nbasins`, and `mo_mhm_constants::nodata_dp`.

Referenced by `objective()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.11 objective_sm_pd()

```
real(dp) function mo_objective_function::objective_sm_pd (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for soil moisture.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore the Pattern Dissimilarity (PD) of observed and modeled soil moisture fields is calculated - aim: matching spatial patterns

$$E(t) = PD(SM_{obs}(t), SM_{sim}(t))$$

where

PD = pattern dissimilarity function,

SM_{obs} = observed soil moisture,

SM_{sim} = simulated soil moisture.

$E(t)$ = pattern dissimilarity at timestep t .

The the pattern dissimilaity (E) is spatially averaged as

$$\phi_i = \frac{1}{T} \cdot \sum_{t=1}^T E_t$$

where T denotes the number of time steps.

Finally, the overall objective function value OF is estimated based on the power-6 norm to combine the ϕ_i from all basins N .

$$OF = \sqrt[6]{\sum ((1.0 - \phi_i)/N)^6}.$$

The observed data L1_sm, L1_sm_mask are global in this module. The observed data L1_sm, L1_sm_mask are global in this module.

Parameters

in	<i>real(dp) :: parameterset(:)</i>	1D-array with parameters the model is run with
----	------------------------------------	--

Returns

real(dp) :: objective_sm_pd — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Matthias Zink

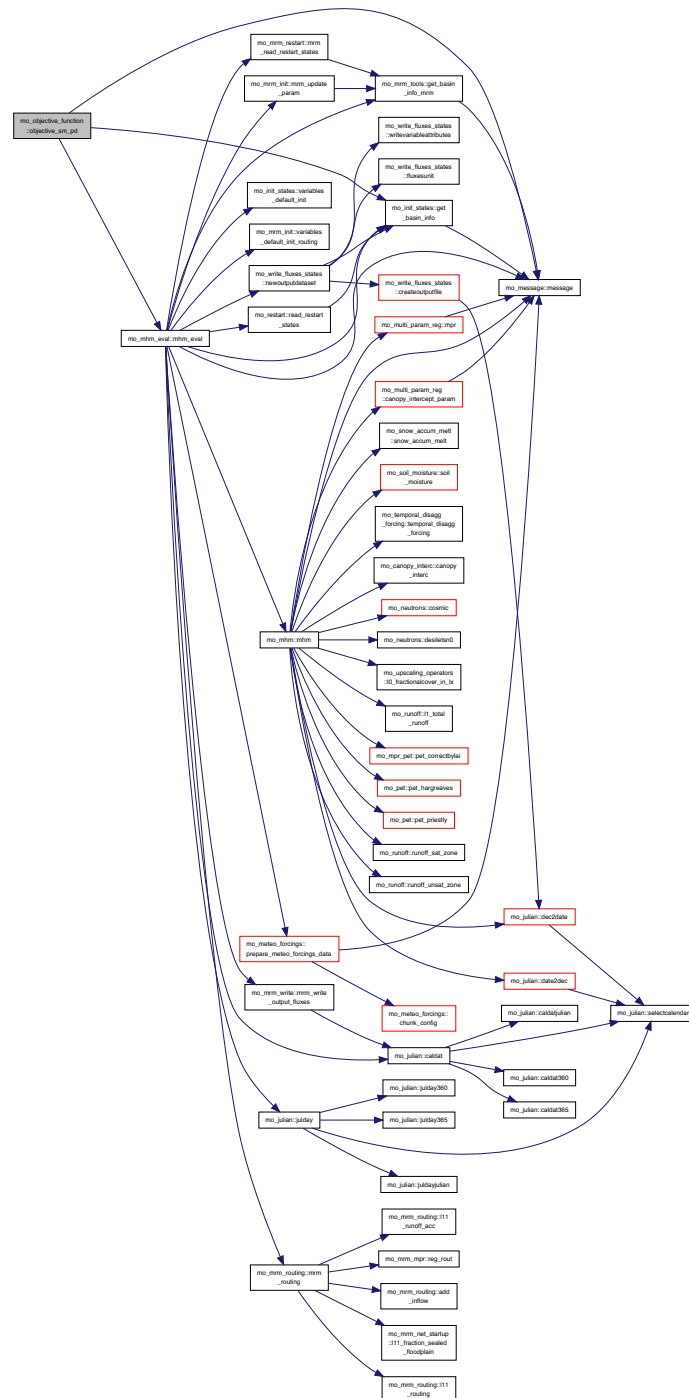
Date

May 2015

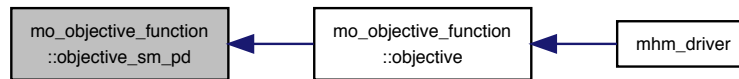
References mo_init_states::get_basin_info(), mo_message::message(), mo_mhm_eval::mhm_eval(), mo_global↵
_variables::nbasins, and mo_mhm_constants::nodata_dp.

Referenced by objective().

Here is the call graph for this function:



Here is the caller graph for this function:



15.51.2.12 objective_sm_sse_standard_score()

```
real(dp) function mo_objective_function::objective_sm_sse_standard_score (
    real(dp), dimension(:), intent(in) parameterset )
```

Objective function for soil moisture.

The objective function only depends on a parameter vector. The model will be called with that parameter vector and the model output is subsequently compared to observed data.

Therefore the sum of squared errors (SSE) of the standard score of observed and modeled soil moisture is calculated. The standard score or normalization (anomaly) make the objective function bias insensitive and basically the dynamics of the soil moisture is tried to capture by this objective function.

$$\phi_i = \sum_{j=1}^K \{ \text{standard_score}(SM_{obs}(j)) - \text{standard_score}(SM_{sim}(j)) \}^2$$

where

standard_score = standard score function,

SM_{obs} = observed soil moisture,

SM_{sim} = simulated soil moisture.

K = valid elements in study domain.

Finally, the overall objective function value *OF* is estimated based on the power-6 norm to combine the ϕ_i from all basins *N*.

$$OF = \sqrt[6]{\sum (\phi_i/N)^6}.$$

The observed data L1_sm, L1_sm_mask are global in this module.

Parameters

in	real(dp) :: parameterset(:)	1D-array with parameters the model is run with
----	--------------------------------	--

Returns

real(dp) :: objective_sm_sse_standard_score — objective function value (which will be e.g. minimized by an optimization routine like DDS)

Note

Input values must be floating points.

Author

Matthias Zink

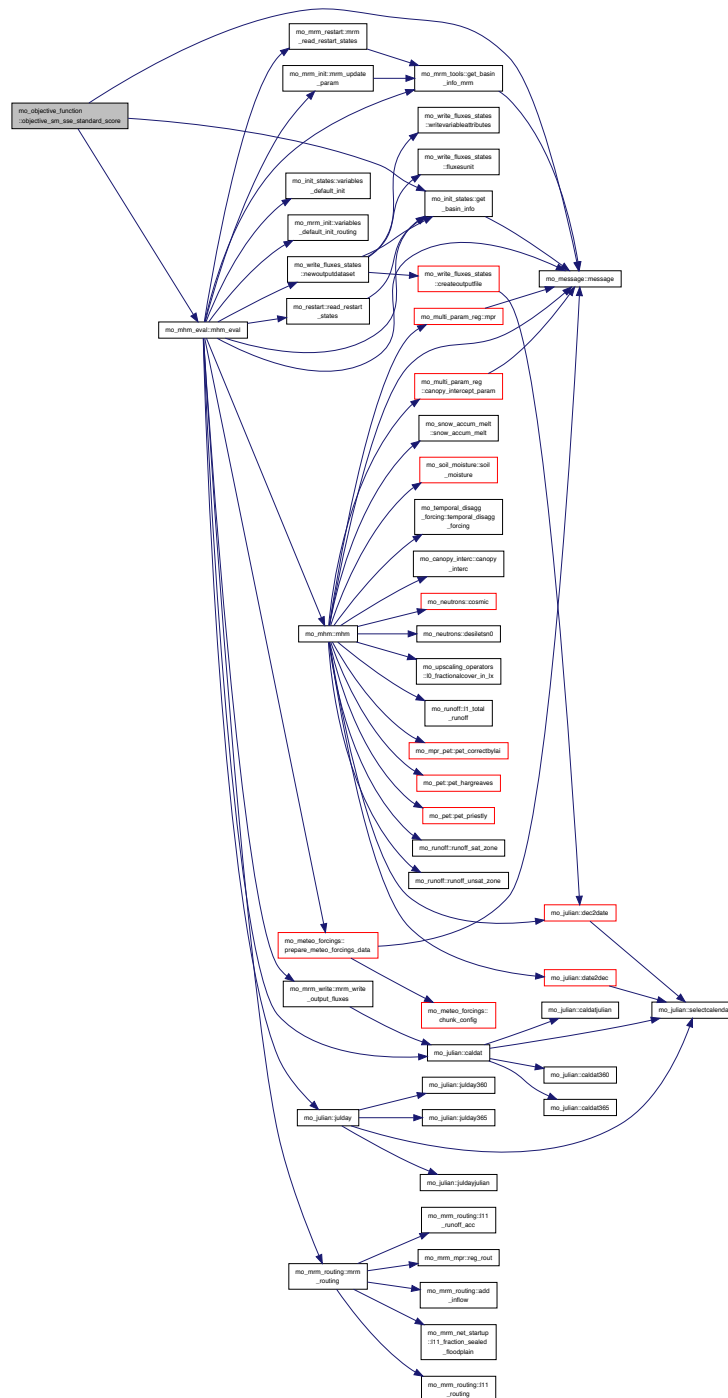
Date

March 2015

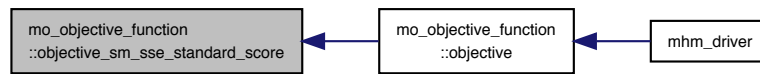
References `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_eval::mhm_eval()`, and `mo_↔
global_variables::nbasins`.

Referenced by `objective()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.52 mo_optimization Module Reference

Wrapper subroutine for optimization against runoff and sm.

Functions/Subroutines

- subroutine, public [optimization](#) (objective, dirConfigOut, funcBest, maskpara)

Wrapper for optimization.

15.52.1 Detailed Description

Wrapper subroutine for optimization against runoff and sm.

This module provides a wrapper subroutine for optimization of mRM/mHM against runoff or soil moisture.

Authors

Stephan Thober

Date

Oct 2015

15.52.2 Function/Subroutine Documentation

15.52.2.1 optimization()

```

subroutine, public mo_optimization::optimization (
    objective,
    character(len=*), intent(in) dirConfigOut,
    real(dp), intent(out) funcBest,
    logical, dimension(:), intent(out), allocatable maskpara )
  
```

Wrapper for optimization.

This subroutine selects the optimization defined in a namelist, i.e. the global variable *opti_method*. It return the objective function value for a specific parameter set.

Parameters

in	<i>real(dp) :: objective</i>	- objective function used in the optimization
----	------------------------------	---

Parameters

in	<i>character(len=*) :: dirConfigOut</i>	- directory where to write ascii output
out	<i>real(dp) :: funcBest</i>	- best objective function value obtained during optimization
out	<i>logical, allocatable :: maskpara(:)</i>	- mask of optimized parameters

Author

Matthias Cuntz, Luis Samaniego, Juliane Mai, Matthias Zink and Stephan Thober

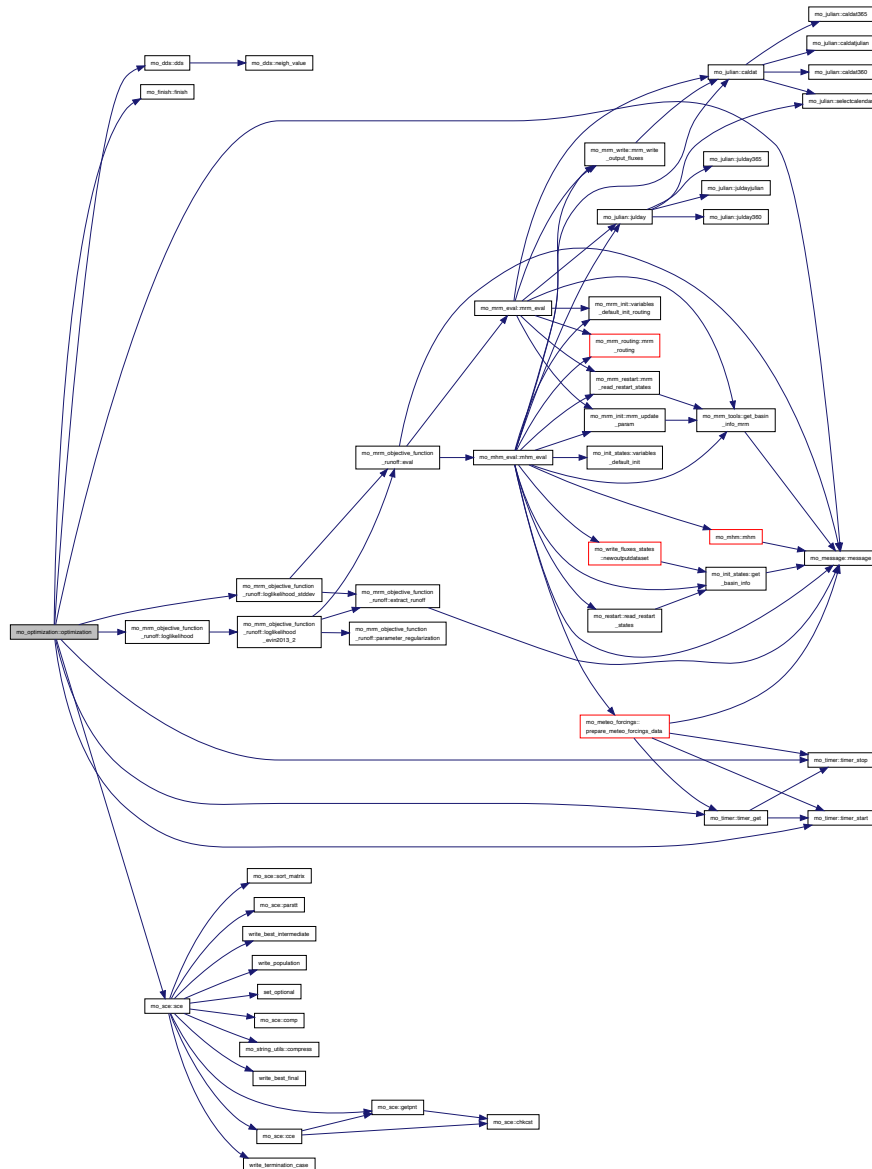
Date

Oct 2015

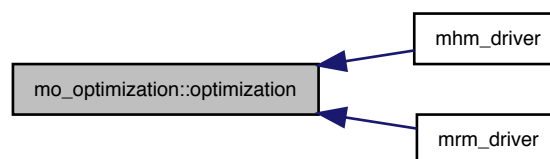
References `mo_dds::dds()`, `mo_kind::dp`, `mo_finish::finish()`, `mo_kind::i4`, `mo_kind::i8`, `mo_mrm_objective↵_function_runoff::loglikelihood()`, `mo_mrm_objective_function_runoff::loglikelihood_stddev()`, `mo_message↵::message()`, `mo_common_variables::opti_method`, `mo_sce::sce()`, `mo_timer::timer_get()`, `mo_timer::timer_start()`, and `mo_timer::timer_stop()`.

Referenced by `mhm_driver()`, and `mrm_driver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.53 mo_orderpack Module Reference

Sort and ranking routines.

Data Types

- interface [ctrper](#)
- interface [fndnth](#)
- interface [indmed](#)
- interface [indnth](#)
- interface [inspar](#)
- interface [inssor](#)
- interface [mrgref](#)
- interface [mrgrnk](#)
- interface [mulcnt](#)
- interface [nearless](#)
- interface [omedian](#)
- interface [rapknr](#)
- interface [refpar](#)
- interface [refsor](#)
- interface [rinpar](#)
- interface [rnkpar](#)
- interface [sort](#)

Unconditional ranking.

- interface [sort_index](#)
- interface [uniinv](#)
- interface [unipar](#)
- interface [unirnk](#)
- interface [unista](#)
- interface [valmed](#)
- interface [valnth](#)

Functions/Subroutines

- integer(i4) function, dimension(size(arr)) [sort_index_dp](#) (arr)
- integer(i4) function, dimension(size(arr)) [sort_index_sp](#) (arr)
- integer(i4) function, dimension(size(arr)) [sort_index_i4](#) (arr)
- subroutine, private [d_ctrper](#) (XDONT, PCLS)
- subroutine, private [r_ctrper](#) (XDONT, PCLS)
- subroutine, private [i_ctrper](#) (XDONT, PCLS)
- real(kind=dp) function, private [d_fndnth](#) (XDONT, NORD)
- real(kind=sp) function, private [r_fndnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [i_fndnth](#) (XDONT, NORD)
- subroutine, private [d_indmed](#) (XDONT, INDM)
- recursive subroutine, private [d_med](#) (XDATT, IDATT, ires_med)
- subroutine, private [r_indmed](#) (XDONT, INDM)
- recursive subroutine, private [r_med](#) (XDATT, IDATT, ires_med)
- subroutine, private [i_indmed](#) (XDONT, INDM)
- recursive subroutine, private [i_med](#) (XDATT, IDATT, ires_med)
- integer(kind=i4) function, private [d_indnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [r_indnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [i_indnth](#) (XDONT, NORD)
- subroutine, private [d_inspar](#) (XDONT, NORD)

- subroutine, private [r_inspar](#) (XDONT, NORD)
- subroutine, private [i_inspar](#) (XDONT, NORD)
- subroutine, private [d_inssor](#) (XDONT)
- subroutine, private [r_inssor](#) (XDONT)
- subroutine, private [i_inssor](#) (XDONT)
- real(kind=dp) function, private [d_median](#) (XDONT)
- real(kind=sp) function, private [r_median](#) (XDONT)
- integer(kind=i4) function, private [i_median](#) (XDONT)
- subroutine, private [d_mrgref](#) (XVALT, IRNGT)
- subroutine, private [r_mrgref](#) (XVALT, IRNGT)
- subroutine, private [i_mrgref](#) (XVALT, IRNGT)
- subroutine, private [d_mrgnrk](#) (XDONT, IRNGT)
- subroutine, private [r_mrgnrk](#) (XDONT, IRNGT)
- subroutine, private [i_mrgnrk](#) (XDONT, IRNGT)
- subroutine, private [d_mulcnt](#) (XDONT, IMULT)
- subroutine, private [r_mulcnt](#) (XDONT, IMULT)
- subroutine, private [i_mulcnt](#) (XDONT, IMULT)
- subroutine, private [d_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [r_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [i_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [d_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [r_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [i_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [d_refsor](#) (XDONT)
- recursive subroutine, private [d_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [r_refsor](#) (XDONT)
- recursive subroutine, private [r_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [i_refsor](#) (XDONT)
- recursive subroutine, private [i_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [d_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [r_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [i_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [d_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [r_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [i_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [d_uniinv](#) (XDONT, IGOEST)
- subroutine, private [r_uniinv](#) (XDONT, IGOEST)
- subroutine, private [i_uniinv](#) (XDONT, IGOEST)
- real(kind=dp) function, private [d_nearless](#) (XVAL)
- real(kind=sp) function, private [r_nearless](#) (XVAL)
- integer(kind=i4) function, private [i_nearless](#) (XVAL)
- subroutine, private [d_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [r_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [i_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [d_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [r_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [i_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [d_unista](#) (XDONT, NUNI)
- subroutine, private [r_unista](#) (XDONT, NUNI)
- subroutine, private [i_unista](#) (XDONT, NUNI)
- recursive real(kind=dp) function, private [d_valmed](#) (XDONT)
- recursive real(kind=sp) function, private [r_valmed](#) (XDONT)
- recursive integer(kind=i4) function, private [i_valmed](#) (XDONT)
- real(kind=dp) function, private [d_valnth](#) (XDONT, NORD)
- real(kind=sp) function, private [r_valnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [i_valnth](#) (XDONT, NORD)

Variables

- integer(kind=i4), dimension(:), allocatable, save [idont](#)

15.53.1 Detailed Description

Sort and ranking routines.

This module is the Orderpack 2.0 from Michel Olnon. It provides order and unconditional, unique, and partial ranking, sorting, and permutation.

Authors

Michel Olnon

Date

2000-2012

15.53.2 Function/Subroutine Documentation

15.53.2.1 d_ctrper()

```
subroutine, private mo_orderpack::d_ctrper (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    real(kind=dp), intent(in) PCLS ) [private]
```

15.53.2.2 d_fndnth()

```
real(kind=dp) function, private mo_orderpack::d_fndnth (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.3 d_indmed()

```
subroutine, private mo_orderpack::d_indmed (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM ) [private]
```

References [d_med\(\)](#), and [idont](#).

Here is the call graph for this function:



15.53.2.4 d_indnth()

```
integer(kind=i4) function, private mo_orderpack::d_indnth (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.5 d_inspar()

```
subroutine, private mo_orderpack::d_inspar (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.6 d_inssor()

```
subroutine, private mo_orderpack::d_inssor (
    real(kind=dp), dimension (:), intent(inout) XDONT ) [private]
```

Referenced by d_refsor().

Here is the caller graph for this function:



15.53.2.7 d_med()

```
recursive subroutine, private mo_orderpack::d_med (
    real(kind=dp), dimension (:), intent(in) XDATT,
    integer(kind=i4), dimension (:), intent(in) IDATT,
    integer(kind=i4), intent(out) ires_med ) [private]
```

Referenced by d_indmed().

Here is the caller graph for this function:



15.53.2.8 d_median()

```
real(kind=dp) function, private mo_orderpack::d_median (
    real(kind=dp), dimension (:), intent(in) XDONT ) [private]
```

15.53.2.9 d_mrgref()

```
subroutine, private mo_orderpack::d_mrgref (
    real(kind=dp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.10 d_mrgrnk()

```
subroutine, private mo_orderpack::d_mrgrnk (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.11 d_mulcnt()

```
subroutine, private mo_orderpack::d_mulcnt (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IMULT ) [private]
```

15.53.2.12 d_nearless()

```
real(kind=dp) function, private mo_orderpack::d_nearless (
    real(kind=dp), intent(in) XVAL ) [private]
```

15.53.2.13 d_rapknr()

```
subroutine, private mo_orderpack::d_rapknr (
    real(kind=dp), dimension (:), intent(in) XDONT,
```

```
integer(kind=i4), dimension (:), intent(out) IRNGT,
integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.14 d_refpar()

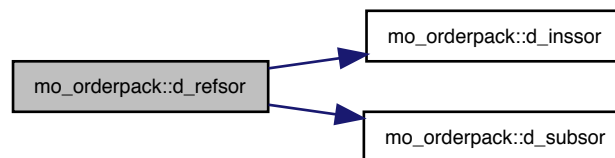
```
subroutine, private mo_orderpack::d_refpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.15 d_refsor()

```
subroutine, private mo_orderpack::d_refsor (
    real(kind=dp), dimension (:), intent(inout) XDONT ) [private]
```

References `d_inssor()`, and `d_subsor()`.

Here is the call graph for this function:



15.53.2.16 d_rinpar()

```
subroutine, private mo_orderpack::d_rinpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.17 d_rnkpar()

```
subroutine, private mo_orderpack::d_rnkpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.18 d_subsor()

```
recursive subroutine, private mo_orderpack::d_subsor (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) IDEB1,
    integer(kind=i4), intent(in) IFIN1 ) [private]
```

Referenced by d_refsr().

Here is the caller graph for this function:

**15.53.2.19 d_uniinv()**

```
subroutine, private mo_orderpack::d_uniinv (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST ) [private]
```

15.53.2.20 d_unipar()

```
subroutine, private mo_orderpack::d_unipar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD ) [private]
```

15.53.2.21 d_unirnk()

```
subroutine, private mo_orderpack::d_unirnk (
    real(kind=dp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(out) NUNI ) [private]
```

15.53.2.22 d_unista()

```
subroutine, private mo_orderpack::d_unista (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI ) [private]
```

15.53.2.23 d_valmed()

```
recursive real(kind=dp) function, private mo_orderpack::d_valmed (
    real(kind=dp), dimension (:), intent(in) XDONT ) [private]
```

15.53.2.24 d_valnth()

```
real(kind=dp) function, private mo_orderpack::d_valnth (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.25 i_ctrper()

```
subroutine, private mo_orderpack::i_ctrper (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    real(kind=sp), intent(in) PCLS ) [private]
```

15.53.2.26 i_fndnth()

```
integer(kind=i4) function, private mo_orderpack::i_fndnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.27 i_indmed()

```
subroutine, private mo_orderpack::i_indmed (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM ) [private]
```

References `i_med()`, and `idont`.

Here is the call graph for this function:

**15.53.2.28 i_indnth()**

```
integer(kind=i4) function, private mo_orderpack::i_indnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
```

```
integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.29 i_inspar()

```
subroutine, private mo_orderpack::i_inspar (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.30 i_inssor()

```
subroutine, private mo_orderpack::i_inssor (
    integer(kind=i4), dimension (:), intent(inout) XDONT ) [private]
```

Referenced by i_refsor().

Here is the caller graph for this function:



15.53.2.31 i_med()

```
recursive subroutine, private mo_orderpack::i_med (
    integer(kind=i4), dimension (:), intent(in) XDATT,
    integer(kind=i4), dimension (:), intent(in) IDATT,
    integer(kind=i4), intent(out) ires_med ) [private]
```

Referenced by i_indmed().

Here is the caller graph for this function:



15.53.2.32 i_median()

```
integer(kind=i4) function, private mo_orderpack::i_median (  
    integer(kind=i4), dimension (:), intent(in) XDONT ) [private]
```

15.53.2.33 i_mrgref()

```
subroutine, private mo_orderpack::i_mrgref (  
    integer(kind=i4), dimension (:), intent(in) XVALT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.34 i_mrgrnk()

```
subroutine, private mo_orderpack::i_mrgrnk (  
    integer(kind=i4), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.35 i_mulcnt()

```
subroutine, private mo_orderpack::i_mulcnt (  
    integer(kind=i4), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IMULT ) [private]
```

15.53.2.36 i_nearless()

```
integer(kind=i4) function, private mo_orderpack::i_nearless (  
    integer(kind=i4), intent(in) XVAL ) [private]
```

15.53.2.37 i_rapknr()

```
subroutine, private mo_orderpack::i_rapknr (  
    integer(kind=i4), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT,  
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.38 i_refpar()

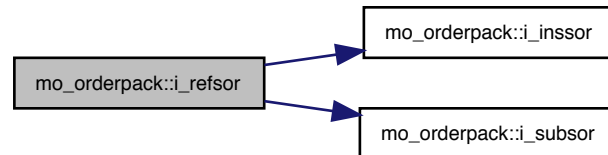
```
subroutine, private mo_orderpack::i_refpar (  
    integer(kind=i4), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT,  
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.39 i_refsor()

```
subroutine, private mo_orderpack::i_refsor (
    integer(kind=i4), dimension (:), intent(inout) XDONT ) [private]
```

References i_inssor(), and i_subsor().

Here is the call graph for this function:

**15.53.2.40 i_rinpar()**

```
subroutine, private mo_orderpack::i_rinpar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.41 i_rnkpar()

```
subroutine, private mo_orderpack::i_rnkpar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.42 i_subsor()

```
recursive subroutine, private mo_orderpack::i_subsor (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) IDEB1,
    integer(kind=i4), intent(in) IFIN1 ) [private]
```

Referenced by i_refsor().

Here is the caller graph for this function:



15.53.2.43 i_uniinv()

```

subroutine, private mo_orderpack::i_uniinv (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST ) [private]
  
```

15.53.2.44 i_unipar()

```

subroutine, private mo_orderpack::i_unipar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD ) [private]
  
```

15.53.2.45 i_unirnk()

```

subroutine, private mo_orderpack::i_unirnk (
    integer(kind=i4), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(out) NUNI ) [private]
  
```

15.53.2.46 i_unista()

```

subroutine, private mo_orderpack::i_unista (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI ) [private]
  
```

15.53.2.47 i_valmed()

```

recursive integer(kind=i4) function, private mo_orderpack::i_valmed (
    integer(kind=i4), dimension (:), intent(in) XDONT ) [private]
  
```

15.53.2.48 i_valnth()

```
integer(kind=i4) function, private mo_orderpack::i_valnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.49 r_ctrper()

```
subroutine, private mo_orderpack::r_ctrper (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    real(kind=sp), intent(in) PCLS ) [private]
```

15.53.2.50 r_fndnth()

```
real(kind=sp) function, private mo_orderpack::r_fndnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.51 r_indmed()

```
subroutine, private mo_orderpack::r_indmed (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM ) [private]
```

References `ident`, and `r_med()`.

Here is the call graph for this function:

**15.53.2.52 r_indnth()**

```
integer(kind=i4) function, private mo_orderpack::r_indnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.53 r_inspar()

```
subroutine, private mo_orderpack::r_inspar (
```

```

real(kind=sp), dimension (:), intent(inout) XDONT,
integer(kind=i4), intent(in) NORD ) [private]

```

15.53.2.54 r_inssor()

```

subroutine, private mo_orderpack::r_inssor (
    real(kind=sp), dimension (:), intent(inout) XDONT ) [private]

```

Referenced by r_refsor().

Here is the caller graph for this function:



15.53.2.55 r_med()

```

recursive subroutine, private mo_orderpack::r_med (
    real(kind=sp), dimension (:), intent(in) XDATT,
    integer(kind=i4), dimension (:), intent(in) IDATT,
    integer(kind=i4), intent(out) ires_med ) [private]

```

Referenced by r_indmed().

Here is the caller graph for this function:



15.53.2.56 r_median()

```

real(kind=sp) function, private mo_orderpack::r_median (
    real(kind=sp), dimension (:), intent(in) XDONT ) [private]

```

15.53.2.57 r_mrgref()

```
subroutine, private mo_orderpack::r_mrgref (  
    real(kind=sp), dimension (:), intent(in) XVALT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.58 r_mrgrnk()

```
subroutine, private mo_orderpack::r_mrgrnk (  
    real(kind=sp), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT ) [private]
```

15.53.2.59 r_mulcnt()

```
subroutine, private mo_orderpack::r_mulcnt (  
    real(kind=sp), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IMULT ) [private]
```

15.53.2.60 r_nearless()

```
real(kind=sp) function, private mo_orderpack::r_nearless (  
    real(kind=sp), intent(in) XVAL ) [private]
```

15.53.2.61 r_rapknr()

```
subroutine, private mo_orderpack::r_rapknr (  
    real(kind=sp), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT,  
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.62 r_refpar()

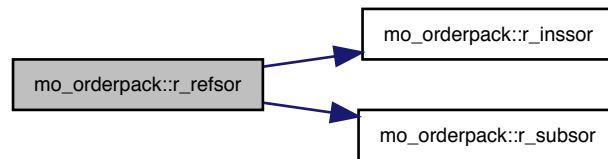
```
subroutine, private mo_orderpack::r_refpar (  
    real(kind=sp), dimension (:), intent(in) XDONT,  
    integer(kind=i4), dimension (:), intent(out) IRNGT,  
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.63 r_refsor()

```
subroutine, private mo_orderpack::r_refsor (  
    real(kind=sp), dimension (:), intent(inout) XDONT ) [private]
```

References `r_inssor()`, and `r_subsor()`.

Here is the call graph for this function:



15.53.2.64 r_rinpar()

```

subroutine, private mo_orderpack::r_rinpar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
  
```

15.53.2.65 r_rnkpar()

```

subroutine, private mo_orderpack::r_rnkpar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD ) [private]
  
```

15.53.2.66 r_subsor()

```

recursive subroutine, private mo_orderpack::r_subsor (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) IDEB1,
    integer(kind=i4), intent(in) IFIN1 ) [private]
  
```

Referenced by r_refsor().

Here is the caller graph for this function:



15.53.2.67 r_uniinv()

```
subroutine, private mo_orderpack::r_uniinv (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST ) [private]
```

15.53.2.68 r_unipar()

```
subroutine, private mo_orderpack::r_unipar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD ) [private]
```

15.53.2.69 r_unirnk()

```
subroutine, private mo_orderpack::r_unirnk (
    real(kind=sp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(out) NUNI ) [private]
```

15.53.2.70 r_unista()

```
subroutine, private mo_orderpack::r_unista (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI ) [private]
```

15.53.2.71 r_valmed()

```
recursive real(kind=sp) function, private mo_orderpack::r_valmed (
    real(kind=sp), dimension (:), intent(in) XDONT ) [private]
```

15.53.2.72 r_valnth()

```
real(kind=sp) function, private mo_orderpack::r_valnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD ) [private]
```

15.53.2.73 sort_index_dp()

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index_dp (
    real(dp), dimension(:), intent(in) arr ) [private]
```

15.53.2.74 sort_index_i4()

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index_i4 (
    integer(i4), dimension(:), intent(in) arr ) [private]
```

15.53.2.75 sort_index_sp()

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index_sp (
    real(sp), dimension(:), intent(in) arr ) [private]
```

15.53.3 Variable Documentation**15.53.3.1 idont**

```
integer(kind=i4), dimension(:), allocatable, save mo_orderpack::idont [private]
```

Referenced by `d_indmed()`, `i_indmed()`, and `r_indmed()`.

15.54 mo_percentile Module Reference**Data Types**

- interface [median](#)
- interface [n_element](#)
- interface [percentile](#)
- interface [qmedian](#)

Functions/Subroutines

- real(dp) function [median_dp](#) (arrin, mask)
- real(sp) function [median_sp](#) (arrin, mask)
- real(dp) function [n_element_dp](#) (idat, n, mask, before, after, previous, next)
- real(sp) function [n_element_sp](#) (idat, n, mask, before, after, previous, next)
- real(dp) function [percentile_0d_dp](#) (arrin, k, mask, mode_in)
- real(sp) function [percentile_0d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function, dimension(size(k)) [percentile_1d_dp](#) (arrin, k, mask, mode_in)
- real(sp) function, dimension(size(k)) [percentile_1d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function [qmedian_dp](#) (dat)
- real(sp) function [qmedian_sp](#) (dat)

15.54.1 Function/Subroutine Documentation

15.54.1.1 median_dp()

```
real(dp) function mo_percentile::median_dp (
    real(dp), dimension(:), intent(in) arrin,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.54.1.2 median_sp()

```
real(sp) function mo_percentile::median_sp (
    real(sp), dimension(:), intent(in) arrin,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.54.1.3 n_element_dp()

```
real(dp) function mo_percentile::n_element_dp (
    real(dp), dimension(:), intent(in) idat,
    integer(i4), intent(in) n,
    logical, dimension(:), intent(in), optional mask,
    real(dp), intent(out), optional before,
    real(dp), intent(out), optional after,
    real(dp), intent(out), optional previous,
    real(dp), intent(out), optional next ) [private]
```

15.54.1.4 n_element_sp()

```
real(sp) function mo_percentile::n_element_sp (
    real(sp), dimension(:), intent(in) idat,
    integer(i4), intent(in) n,
    logical, dimension(:), intent(in), optional mask,
    real(sp), intent(out), optional before,
    real(sp), intent(out), optional after,
    real(sp), intent(out), optional previous,
    real(sp), intent(out), optional next ) [private]
```

15.54.1.5 percentile_0d_dp()

```
real(dp) function mo_percentile::percentile_0d_dp (
    real(dp), dimension(:), intent(in) arrin,
    real(dp), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in ) [private]
```

References mo_kind::i4.

15.54.1.6 percentile_0d_sp()

```
real(sp) function mo_percentile::percentile_0d_sp (
    real(sp), dimension(:), intent(in) arrin,
```

```

    real(sp), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in ) [private]

```

References `mo_kind::i4`.

15.54.1.7 percentile_1d_dp()

```

real(dp) function, dimension(size(k)) mo_percentile::percentile_1d_dp (
    real(dp), dimension(:), intent(in) arrin,
    real(dp), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in ) [private]

```

References `mo_kind::i4`.

15.54.1.8 percentile_1d_sp()

```

real(sp) function, dimension(size(k)) mo_percentile::percentile_1d_sp (
    real(sp), dimension(:), intent(in) arrin,
    real(sp), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in ) [private]

```

References `mo_kind::i4`.

15.54.1.9 qmedian_dp()

```

real(dp) function mo_percentile::qmedian_dp (
    real(dp), dimension(:), intent(inout) dat ) [private]

```

15.54.1.10 qmedian_sp()

```

real(sp) function mo_percentile::qmedian_sp (
    real(sp), dimension(:), intent(inout) dat ) [private]

```

15.55 mo_pet Module Reference

Module for calculating reference/potential evapotranspiration [mm s⁻¹].

Functions/Subroutines

- elemental pure real(dp) function, public [pet_hargreaves](#) (HarSamCoeff, HarSamConst, tavg, tmax, tmin, latitude, doy)
Reference Evapotranspiration after Hargreaves.
- elemental pure real(dp) function, public [pet_priestly](#) (PriTayParam, Rn, tavg)
Reference Evapotranspiration after Priestly-Taylor.

- elemental pure real(dp) function, public [pet_penman](#) (net_rad, tavg, act_vap_pressure, aerodyn_resistance, bulksurface_resistance, a_s, a_sh)

Reference Evapotranspiration after Penman-Monteith.

- elemental pure real(dp) function, private [extraterr_rad_approx](#) (doy, latitude)

Approximation of extraterrestrial radiation.

- elemental pure real(dp) function, private [slope_satpressure](#) (tavg)

slope of saturation vapour pressure curve

- elemental pure real(dp) function, private [sat_vap_pressure](#) (tavg)

calculation of the saturation vapour pressure

15.55.1 Detailed Description

Module for calculating reference/potential evapotranspiration [mm s⁻¹].

This module calculates PET [mm/s] based on one of the methods

- Hargreaves-Samani (1982)
- Priestly-Taylor (1972)
- Penman-Monteith FAO (1998)

Author

Matthias Zink, Christoph Schneider, Matthias Cuntz

Date

Apr 2014

15.55.2 Function/Subroutine Documentation

15.55.2.1 extraterr_rad_approx()

```
elemental pure real(dp) function, private mo_pet::extraterr_rad_approx (
    integer(i4), intent(in) doyear,
    real(dp), intent(in) latitude ) [private]
```

Approximation of extraterrestrial radiation.

Approximation of extraterrestrial radiation at the top of the atmosphere R_a after Duffie and Beckman (1980). R_a is converted from [$J m^{-2} d^{-1}$] in [$mm d^{-1}$].

$$R_a = \frac{86400}{\pi \cdot \lambda} \cdot E_0 \cdot d_r \cdot (\omega \cdot \sin(latitude) \cdot \sin(\delta) + \cos(latitude) \cdot \cos(\delta) \cdot \sin(\omega))$$

where $E_0 = 1367 J m^{-2} s^{-1}$ is the solar constant and it is dependent on the following sub equations:

The relative distance Earth-Sun:

$$d_r = 1 + 0.033 \cdot \cos\left(\frac{2 \cdot \pi \cdot doyear}{365}\right)$$

in which *doy* is the day of the year.

The solar declination [radians] defined by

$$\delta = 0.4093 \cdot \sin\left(\frac{2 \cdot \pi \cdot \text{doy}}{365} - 1.405\right)$$

The sunset hour angle [radians]:

$$\omega = \arccos(-\tan(\text{latitude}) * \tan(\delta))$$

Parameters

in	<i>integer(i4), intent(in) :: doy</i>	day of year [-]
in	<i>real(dp), intent(in) :: latitude</i>	latitude [rad]

Returns

real(dp) :: extraterr_rad_approx — extraterrestrial radiation approximation [W m^{-2}]

Note

Duffie, J.A. and W.A. Beckman. 1980. Solar engineering of thermal processes. John Wiley and Sons, New York. pp. 1-109.

Author

Matthias Zink

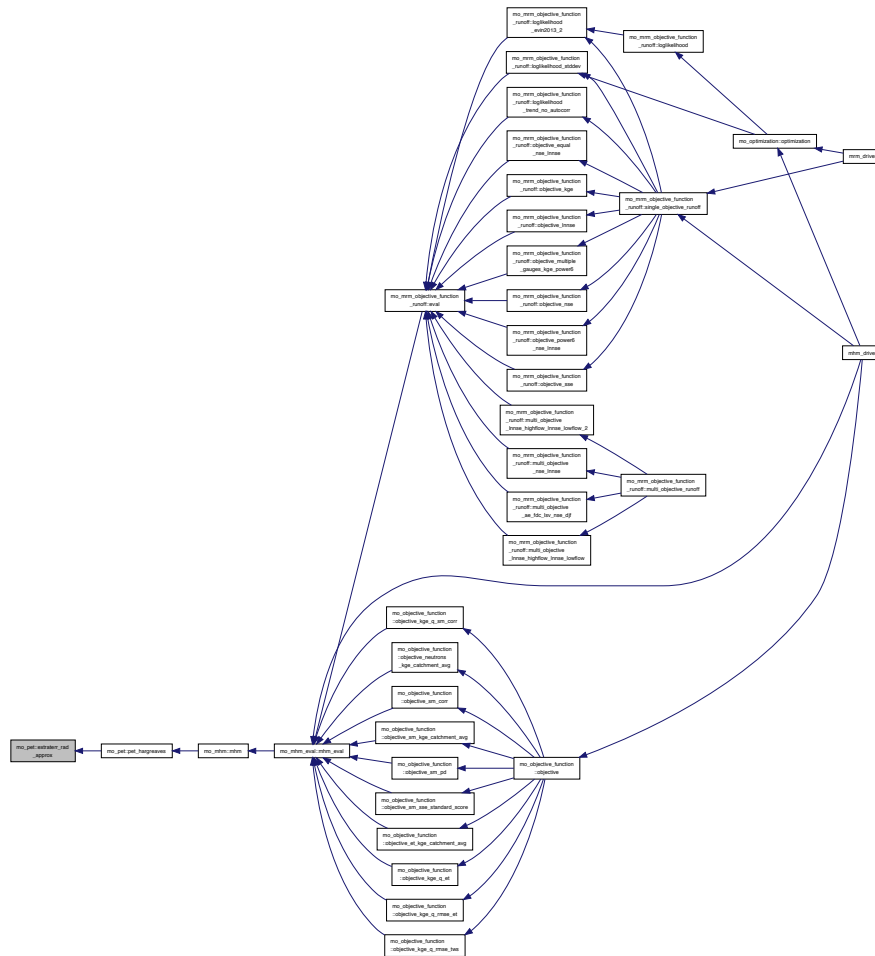
Date

Apr 2014

References *mo_mhm_constants::daysecs*, *mo_mhm_constants::duffiedelta1*, *mo_mhm_constants::duffiedelta2*, *mo_mhm_constants::duffiedr*, *mo_constants::pi_d*, *mo_constants::solarconst_dp*, *mo_constants::specheatet_dp*, *mo_constants::twopi_d*, and *mo_mhm_constants::yeardays*.

Referenced by *pet_hargreaves()*.

Here is the caller graph for this function:



15.55.2.2 pet_hargreaves()

```

elemental pure real(dp) function, public mo_pet::pet_hargreaves (
    real(dp), intent(in) HarSamCoeff,
    real(dp), intent(in) HarSamConst,
    real(dp), intent(in) tavg,
    real(dp), intent(in) tmax,
    real(dp), intent(in) tmin,
    real(dp), intent(in) latitude,
    integer(i4), intent(in) doy )

```

Reference Evapotranspiration after Hargreaves.

Calculates the Reference Evapotranspiration [$mm\ d^{-1}$] based on the Hargreaves-Samani (1982) model for a given cell by applying the equation

$$PET = HarSamCoeff * R_a * (T_{avg} + HarSamConst) * \sqrt{T_{max} - T_{min}}$$

where R_a [$W\ m^{-2}$] is the incoming solar radiation and T_{avg} , T_{max} and T_{min} [$^{\circ}C$] are the mean, maximum, and minimum daily temperatures at the given day, respectively.

Parameters

in	<i>real(dp), intent(in) :: HarSamCoeff</i>	coefficient of Hargreaves-Samani equation [-]
in	<i>real(dp), intent(in) :: HarSamConst</i>	constant of Hargreaves-Samani equation [-]
in	<i>real(dp), intent(in) :: tavg</i>	daily mean temperature [$^{\circ}\text{C}$]
in	<i>real(dp), intent(in) :: tmax</i>	daily maximum of temp [$^{\circ}\text{C}$]
in	<i>real(dp), intent(in) :: tmin</i>	daily minimum of temp [$^{\circ}\text{C}$]
in	<i>real(dp), intent(in) :: latitude</i>	latitude of the cell for Ra estimation [<i>radians</i>]

Returns

real(dp) :: pet_hargreaves — Hargreaves-Samani pot. evapotranspiration [mm s-1]

Author

Matthias Zink

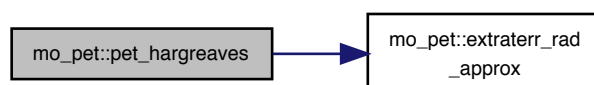
Date

Dec 2012

References `mo_constants::deg2rad_dp`, and `extraterr_rad_approx()`.

Referenced by `mo_mhm::mhm()`.

Here is the call graph for this function:



The diagram is a directed graph with nodes representing different types of objectives and functions. The nodes are organized into several clusters. At the top, there are nodes like 'ms_mem_objective_function' and 'ms_mem_objective_function_runoff'. In the middle, there are nodes like 'ms_mem_objective_function_runoff' and 'ms_mem_objective_function_runoff'. At the bottom, there are nodes like 'ms_objective_function' and 'ms_objective_function'. The graph shows a dense network of directed edges connecting these nodes, indicating a complex dependency or relationship structure.

```

elemental pure real(dp) function, public mo_pet::pet_penman (
    real(dp), intent(in) net_rad,
    real(dp), intent(in) tavg,
    real(dp), intent(in) act_vap_pressure,
    real(dp), intent(in) aerodyn_resistance,
    real(dp), intent(in) bulksurface_resistance,
    real(dp), intent(in) a_s,
    real(dp), intent(in) a_sh )

```

Calculates the reference evapotranspiration [$mm\ d^{-1}$] based on the Penman-Monteith model for every given cell by applying the equation

$$PET = \frac{1}{\lambda} \cdot \frac{\Delta \cdot R_n + \rho \cdot c_p \cdot (e_s - e) \cdot \frac{a_s h}{r_a}}{\Delta + \gamma \cdot \frac{a_s h}{r_a} \cdot \left(1 + \frac{r_s}{r_a}\right)}$$

where R_n [$W m^{-2}$] is the net solar radiation, Δ [$kPa K^{-1}$] is the slope of the saturation-vapour pressure curve, λ [$MJ kg^{-1}$] is the latent heat of vaporization, $(e_s - e)$ [kPa] is the vapour pressure deficit of the air, ρ [$kg m^{-3}$] is the mean atmospheric density, $c_p = 1005.0 J kg^{-1} K^{-1}$ is the specific heat of the air, γ [$kPa K^{-1}$] is the psychrometric constant, r_s [sm^{-1}] is the bulk canopy resistance, r_a [sm^{-1}] is the aerodynamic resistance, a_s [1] is the fraction of one-sided leaf area covered by stomata (1 if stomata are on one side only, 2 if they are on both sides) and a_{sh} [–] is the fraction of projected area exchanging sensible heat with the air (2) Implementation refers to the so-called Penman-Monteith equation for transpiration. Adjusting the arguments a_{sh} and a_s we obtain the corrected MU equation (for details see Schymanski and Or, 2017). If $a_{sh} = 1 = a_s$ Penman-Monteith equation for transpiration is preserved. For reproducing characteristics of symmetrical amphistomatous leaves use $a_{sh} = 2 = a_s$, in which case the classic PM equation is only missing a factor of 2 in the nominator, as pointed out by Jarvis and McNaughton (1986, Eq. A9). These analytical solutions eliminated the non-linearity problem of the saturation vapour pressure curve, but they do not consider the dependency of the long-wave component of the soil surface or leaf energy balance (R_l) on soil or leaf temperature (T_l). We assume that net radiation equals the absorbed short-wave radiation, i.e. $R_N = R_s$ (p.79 in Monteith and Unsworth, 2013).

Parameters

in	<i>real(dp), intent(in) :: net_rad</i>	net radiation [$W m^{-2}$]
in	<i>real(dp), intent(in) :: tavg</i>	average daily temperature [$^{\circ}C$]
in	<i>real(dp), intent(in) :: act_vap_pressure</i>	actual vapour pressure [kPa]
in	<i>real(dp), intent(in) :: aerodyn_resistance</i>	aerodynamical resistance $s m^{-1}$
in	<i>real(dp), intent(in) :: bulksurface_resistance</i>	bulk surface resistance $s m^{-1}$
in	<i>real(dp), intent(in) :: a_s</i>	fraction of one-sided leaf area covered by stomata 1
in	<i>real(dp), intent(in) :: a_sh</i>	fraction of projected area exchanging sensible heat with the air 1
in	<i>real(dp) :: pet_penman</i>	reference evapotranspiration [$mm s^{-1}$]

Returns

real(dp) :: pet_penman — Reference Evapotranspiration [$mm s^{-1}$]

Note

Allen, R. G. R., Pereira, L., Raes, D., & Smith, M. (1998). Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56. Rome.
 Schymanski, S. J., & Or, D. (2017). Leaf-scale experiments reveal an important omission in the Penman-Monteith equation. *HESS*, 21(2), 685-706.
 Monteith, J. L. and Unsworth, M. H. (2013) Principles of environmental physics: plants, animals, and the atmosphere, 4th Edn., Elsevier/Academic Press, Amsterdam, Boston.

Author

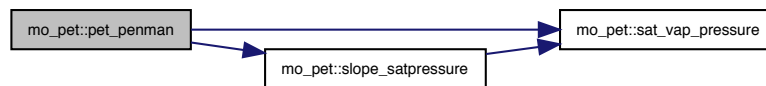
Matthias Zink

Date

Apr 2014

References mo_constants::cp0_dp, mo_mhm_constants::daysecs, mo_constants::psychro_dp, mo_constants::rho0_dp, sat_vap_pressure(), slope_satpressure(), and mo_constants::specheatet_dp.

Here is the call graph for this function:



15.55.2.4 pet_priestly()

```

elemental pure real(dp) function, public mo_pet::pet_priestly (
    real(dp), intent(in) PrieTayParam,
    real(dp), intent(in) Rn,
    real(dp), intent(in) tavg )
  
```

Reference Evapotranspiration after Priestly-Taylor.

Calculates the Reference Evapotranspiration [$mm\ d^{-1}$] based on the Priestly-Taylor (1972) model for every given cell by applying the equation

$$PET = \alpha * \frac{\Delta}{(\gamma + \Delta)} * R_n$$

where R_n [$W\ m^{-2}$] is the net solar radiation $\Delta = f(T_{avg})$ is the slope of the saturation-vapour pressure curve and α is a empirical coefficient.

Parameters

in	<i>real(dp) :: PrieTayParam</i>	Priestley-Taylor coefficient $\alpha[-]$
in	<i>real(dp) :: Rn</i>	net solar radiation [$W\ m^{-2}$]
in	<i>real(dp) :: Tavg</i>	daily mean air temperature [$^{\circ}C$]

Returns

real(dp) :: pet_priestly — Priestley-Taylor pot. evapotranspiration [$mm\ s^{-1}$]

Note

Priestley, C.H.B., and R.J. Taylor. 1972. On the assessment of surface heat flux and evaporation using large-scale parameters. Mon. Weather Rev., 100:81-82.
 ASAE Standards. 1998. EP406.2: heating, cooling, and ventilating greenhouses. St. Joseph, MI, USA.

Author

Matthias Zink

Apr 2014

Referenced by `mo_mhm::mhm()`.

```
graph LR; mo_pet::pet_priestly --> mo_pet::slope_satpressure; mo_pet::slope_satpressure --> mo_pet::sat_vap_pressure;
```

[illegible]

15.55.2.5 sat_vap_pressure()

```
elemental pure real(dp) function, private mo_pet::sat_vap_pressure (
    real(dp), intent(in) tavg ) [private]
```

calculation of the saturation vapour pressure

Calculation of the saturation vapour pressure

$$e_s(T_a) = 0.6108 \cdot \exp\left(\frac{17.27 \cdot T_a}{T_a + 237.3}\right)$$

Parameters

in	real(dp), intent(in) :: tavg	temperature [degC]
----	------------------------------	--------------------

Returns

real(dp) :: sat_vap_pressure — saturation vapour pressure [kPa]

Note

Tetens, O., 1930. Ueber einige meteorologische Begriffe. z. Geophys. 6:297-309.
 Allen, R. G. R., Pereira, L., Raes, D., & Smith, M. (1998). Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56. Rome.

Author

Matthias Zink

Date

Apr 2014

References mo_mhm_constants::tetens_c1, mo_mhm_constants::tetens_c2, and mo_mhm_constants::tetens_c3.
 Referenced by pet_penman(), and slope_satpressure().

Note

Tetens, O., 1930. Ueber einige meteorologische Begriffe. z. Geophys. 6:297-309.
Murray, F.W. 1967. On the computation of saturation vapor pressure. J. Appl. Meteor. 6: 203-204.
Allen, R. G. R., Pereira, L., Raes, D., & Smith, M. (1998). Crop evapotranspiration - Guidelines for computing crop water requirements - FAO Irrigation and drainage paper 56. Rome.

Author

Matthias Zink

Date

Apr 2014

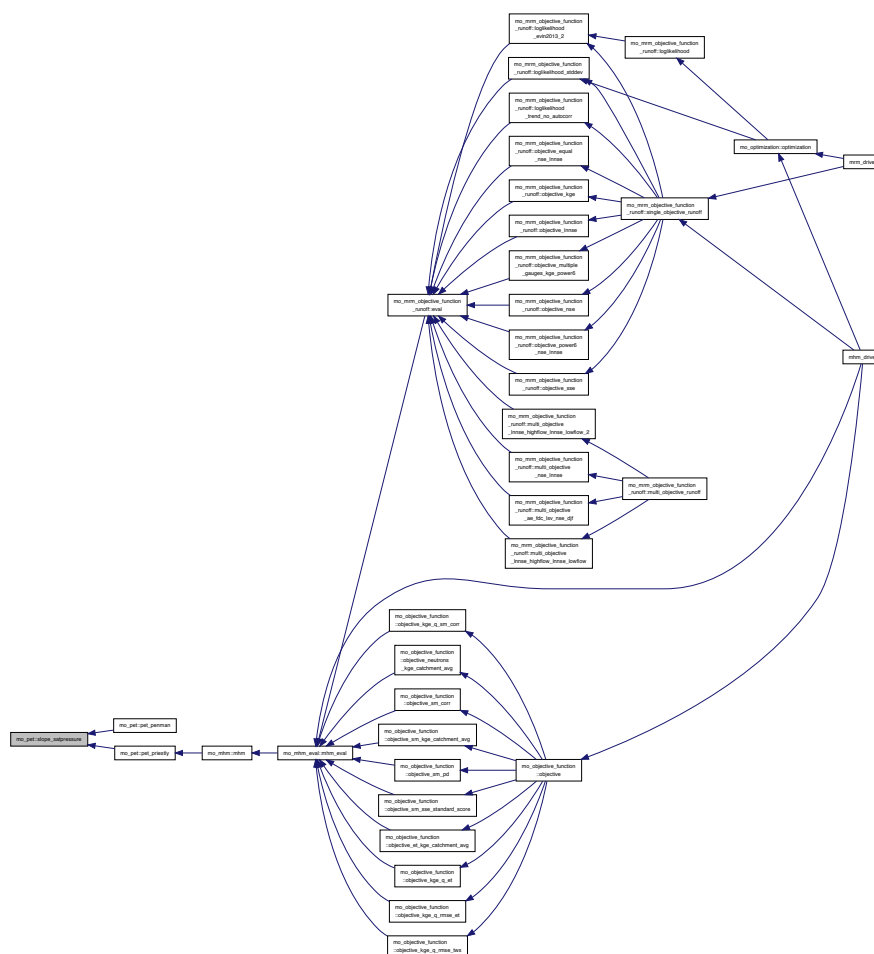
References `sat_vap_pressure()`, `mo_mhm_constants::satpressureslope1`, and `mo_mhm_constants::tetens_c3`.

Referenced by `pet_penman()`, and `pet_priestly()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.56 mo_prepare_gridded_lai Module Reference

Prepare daily LAI fields (e.g., MODIS data) for mHM.

Functions/Subroutines

- subroutine, public [prepare_gridded_daily_lai_data](#) (iBasin)
Prepare gridded daily LAI data.
- subroutine, public [prepare_gridded_mean_monthly_lai_data](#) (iBasin)
prepare_gridded_mean_monthly_LAI_data

15.56.1 Detailed Description

Prepare daily LAI fields (e.g., MODIS data) for mHM.

Prepare daily LAI fields(e.g., MODIS data) for mHM

Authors

John Craven & Rohini Kumar

Date

Aug 2013

15.56.2 Function/Subroutine Documentation**15.56.2.1 prepare_gridded_daily_lai_data()**

```
subroutine, public mo_prepare_gridded_lai::prepare_gridded_daily_lai_data (
    integer(i4), intent(in) iBasin )
```

Prepare gridded daily LAI data.

Prepare gridded daily LAI data at Level-0 (e.g., using MODIS datasets)

Parameters

in	integer(i4) :: iBasin	Basin Id
----	-----------------------	----------

Author

John Craven & Rohini Kumar

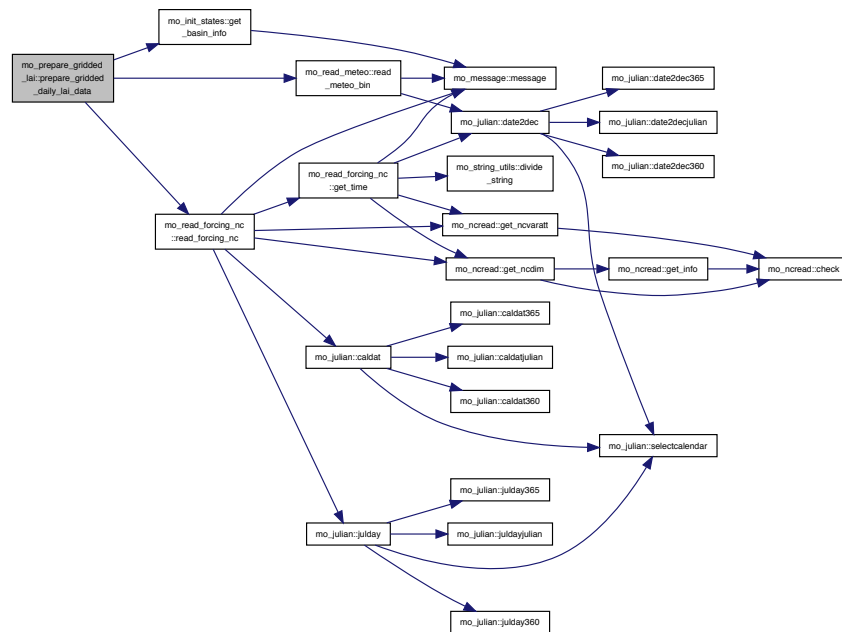
Date

Aug 2013

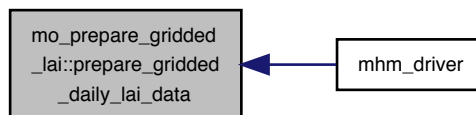
References mo_global_variables::dirgridded_lai, mo_init_states::get_basin_info(), mo_global_variables::inputformat_gridded_lai, mo_global_variables::l0_gridded_lai, mo_read_forcing_nc::read_forcing_nc(), mo_read_meteo::read_meteo_bin(), mo_global_variables::simper, and mo_global_variables::timestep_lai_input.

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.56.2.2 prepare_gridded_mean_monthly_lai_data()

```
subroutine, public mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data (
    integer(i4), intent(in) iBasin )
```

prepare_gridded_mean_monthly_LAI_data

Long term mean monthly gridded LAI data at Level-0 (e.g., using MODIS datasets)
The netcdf file should contain 12 (calender months) gridded fields of climatological LAI data at the input L0 data resolution.

Parameters

in	integer(i4) :: iBasin	Basin Id
----	-----------------------	----------

Author

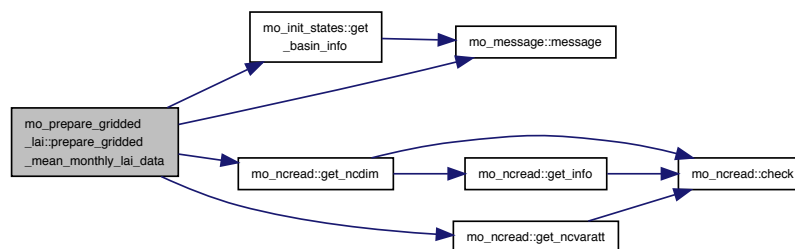
Rohini Kumar

Date

Dec 2016

References mo_global_variables::dirgridded_lai, mo_init_states::get_basin_info(), mo_ncread::get_ncdim(), mo_ncread::get_ncvaratt(), mo_global_variables::l0_gridded_lai, and mo_message::message().

Here is the call graph for this function:



15.57 mo_read_config Module Reference

Reading of main model configurations.

Functions/Subroutines

- subroutine, public `read_config` ()
Read main configurations for mHM.
- logical function `in_bound` (params)

15.57.1 Detailed Description

Reading of main model configurations.

This routine reads the configurations of mHM including, input and output directories, module usage specification, simulation time periods, global parameters, ...

Authors

Matthias Zink

Date

Dec 2012

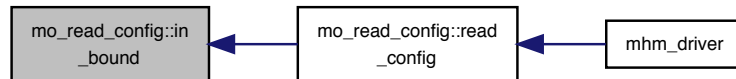
15.57.2 Function/Subroutine Documentation

15.57.2.1 in_bound()

```
logical function mo_read_config::in_bound (
    real(dp), dimension(:,:), intent(in) params )
```

Referenced by read_config().

Here is the caller graph for this function:



15.57.2.2 read_config()

```
subroutine, public mo_read_config::read_config ( )
```

Read main configurations for mHM.

The main configurations in mHM are read from three files:

1. mhm.nml
2. mhm_parameters.nml
3. mhm_outputs.nml

For details please refer to the above mentioned namelist files.

Author

Matthias Zink

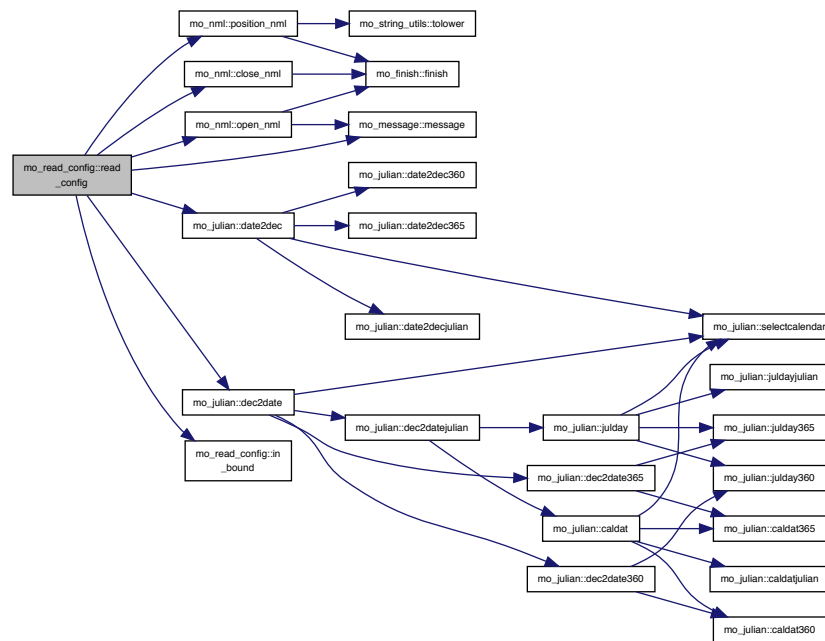
Date

Dec 2012

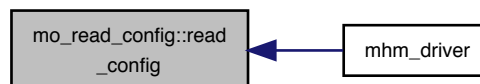
References mo_nml::close_nml(), mo_julian::date2dec(), mo_julian::dec2date(), mo_constants::eps_dp, mo_file::file_namelist, mo_kind::i4, in_bound(), mo_message::message(), mo_mhm_constants::nodata_dp, mo_mhm_constants::nodata_i4, mo_common_variables::nprocesses, mo_nml::open_nml(), mo_nml::position_nml(), mo_common_variables::processmatrix, mo_global_variables::timestep, and mo_file::unamelist.

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.58 mo_read_forcing_nc Module Reference

Reads forcing input data.

Functions/Subroutines

- subroutine, public [read_forcing_nc](#) (folder, nRows, nCols, periode, varName, data, mask, lower, upper, nctimestep, nocheck, maskout)

Reads forcing input in NetCDF file format.

- subroutine, public [read_weights_nc](#) (folder, nRows, nCols, varName, data, mask, lower, upper, nocheck, maskout)

Reads weights for meteo forcings input in NetCDF file format.

- subroutine [get_time](#) (fName, vName, julStart, julEnd, nctimestep)

15.58.1 Detailed Description

Reads forcing input data.

This module is to read forcing input data contained in netcdf files, e.g. temperature, precipitation, total_runoff, lai. Timesteps can be hourly, daily, monthly, and annual. The module provides a subroutine for NetCDF files only. First, the dimensions given are cross-checked with header.txt information. Second, the data of the specified period are read from the specified directory. The names of files in this directory have to be always "<var_name.nc".

If the optional lower and/or upper bound for the data values is given, the read data are checked for validity. The program is stopped if any value lies out of range.

Authors

Juliane Mai

Date

Dec 2012

15.58.2 Function/Subroutine Documentation

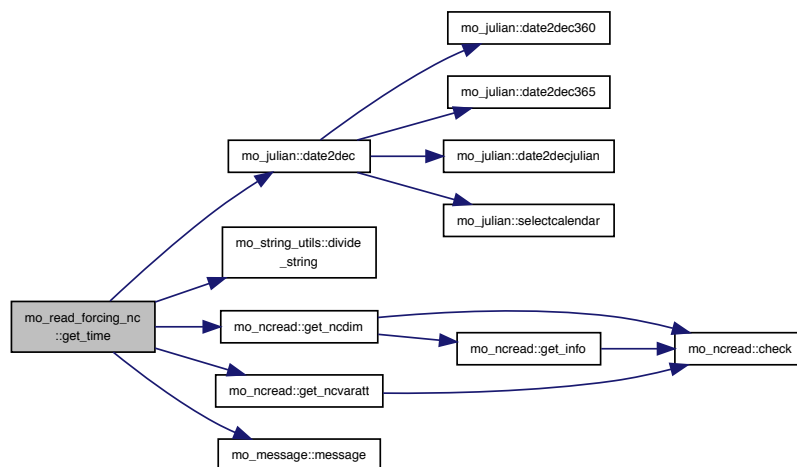
15.58.2.1 [get_time\(\)](#)

```
subroutine mo_read_forcing_nc::get_time (
    character(len=*), intent(in) fName,
    character(len=*), intent(in) vName,
    integer(i4), intent(out) julStart,
    integer(i4), intent(out) julEnd,
    integer(i4), intent(in), optional nctimestep )
```

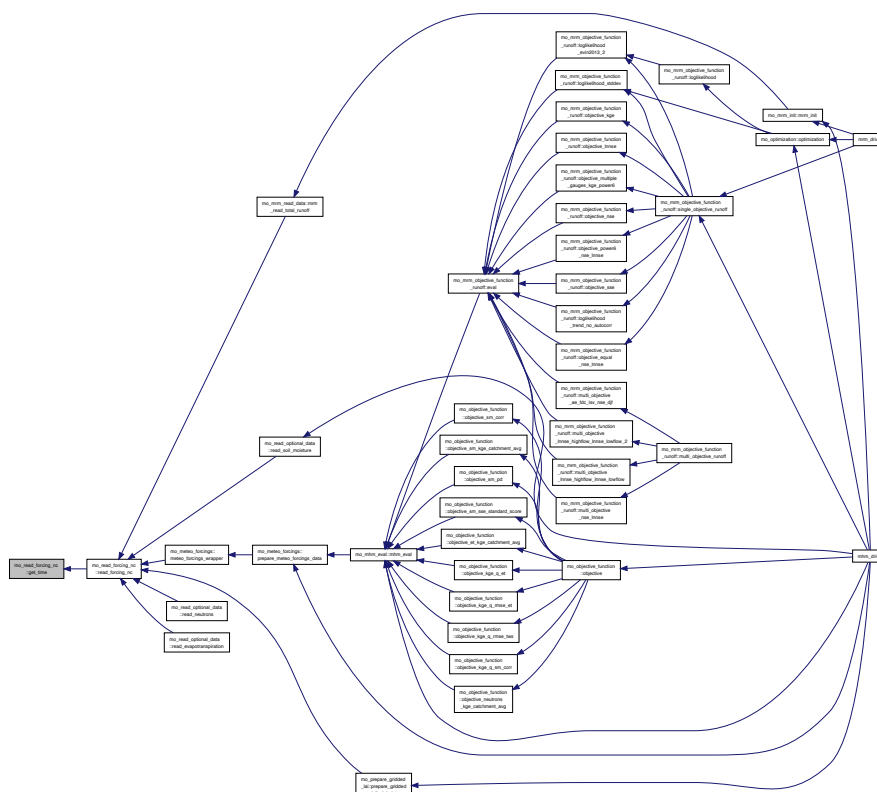
References [mo_julian::date2dec\(\)](#), [mo_string_utils::divide_string\(\)](#), [mo_kind::dp](#), [mo_ncread::get_ncdim\(\)](#), [mo_ncread::get_ncvaratt\(\)](#), [mo_kind::i4](#), and [mo_message::message\(\)](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.58.2.2 read_forcing_nc()

```
subroutine, public mo_read_forcing_nc::read_forcing_nc (
    character(len=*), intent(in) folder,
    integer(i4), intent(in) nRows,
    integer(i4), intent(in) nCols,
    type(period), intent(in) periode,
    character(len=*), intent(in) varName,
    real(dp), dimension(:,:,:), intent(out), allocatable data,
    logical, dimension(:,:), intent(in) mask,
    real(dp), intent(in), optional lower,
    real(dp), intent(in), optional upper,
    integer(i4), intent(in), optional nctimestep,
    logical, intent(in), optional nocheck,
    logical, dimension(:,:,:), intent(out), optional, allocatable maskout )
```

Reads forcing input in NetCDF file format.

Reads netCDF forcing files.

First, the dimensions given are cross-checked with header.txt information. Second, the data of the specified period are read from the specified directory. If the optional lower and/or upper bound for the data values is given, the read data are checked for validity. The program is stopped if any value lies out of range.

If the optional argument nocheck is true, the data are not checked for coverage with the input mask. Additionally in this case an mask of vild data points can be received from the routine in maskout.

Parameters

in	<i>character(len=*) :: folder</i>	Name of the folder where data are stored
in	<i>integer(i4) :: nRows</i>	Number of datapoints in longitudinal direction
in	<i>integer(i4) :: nCols</i>	Number of datapoints in latitudinal direction
in	<i>type(period) :: periode</i>	Period the data are needed for
in	<i>character(len=*) :: varName</i>	Name of variable name to read
in	<i>logical, dimension(:,:) :: mask</i>	mask of valid data fields
out	<i>real(dp), dimension(:,:,:) :: data</i>	Data matrix dim_1 = longitude, dim_2 = latitude, dim_3 = time
in	<i>real(dp), optional, intent(in) :: lower</i>	Lower bound for check of validity of data values
in	<i>real(dp), optional, intent(in) :: upper</i>	Upper bound for check of validity of data values
in	<i>logical, optional, intent(in) :: nocheck</i>	.TRUE. if check for nodata values deactivated default = .FALSE. - check is done
in	<i>integer(i4) optional, intent(in) :: nctimestep</i>	timestep in netcdf file
in	<i>logical, dimension(:,:,:), allocatable, optional, intent(out) :: maskout</i>	! mask of valid data points

Note

Files have to be called like defined in mo_files. Furthermore the variable names have to be called like they are defined in the declaration of this subroutine. The NetCDF file has to have 3 dimensions:

1. x, 2. y, 3. t. It is expected that the variables (especially) within the NetCDF files contain an unit attribute. The timestep has to be equidistant.

Author

Matthias Zink

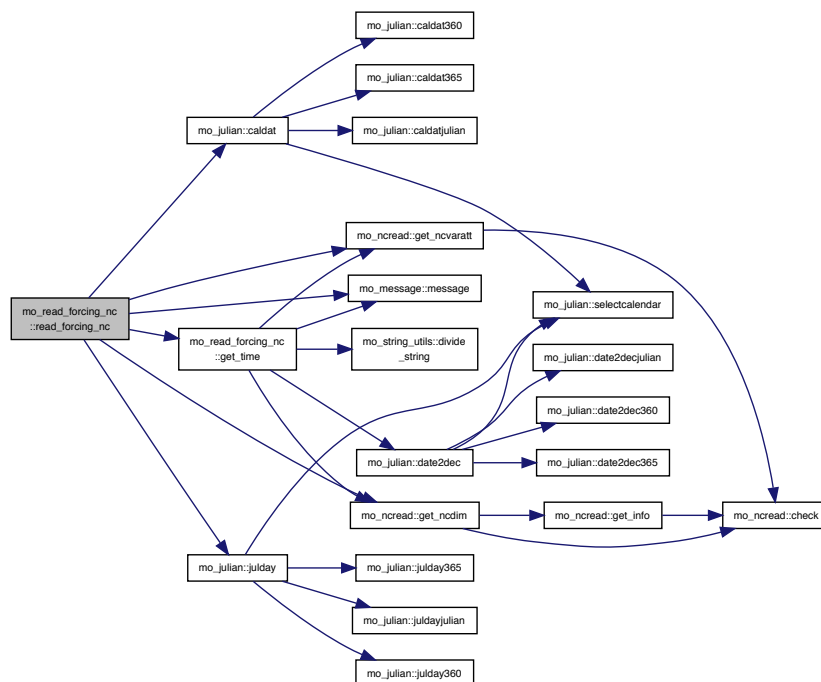
Date

May 2013

References `mo_julian::caldat()`, `mo_kind::dp`, `mo_ncread::get_ncdim()`, `mo_ncread::get_ncvaratt()`, `get_time()`, `mo_kind::i4`, `mo_julian::julday()`, and `mo_message::message()`.

Referenced by `mo_meteo_forcings::meteo_forcings_wrapper()`, `mo_mrm_read_data::mrm_read_total_runoff()`, `mo_prepare_gridded_lai::prepare_gridded_daily_lai_data()`, `mo_read_optional_data::read_evapotranspiration()`, `mo_read_optional_data::read_neutrons()`, and `mo_read_optional_data::read_soil_moisture()`.

Here is the call graph for this function:



Parameters

in	<i>character(len=*) :: folder</i>	Name of the folder where data are stored
in	<i>integer(i4) :: nRows</i>	Number of datapoints in longitudinal direction
in	<i>integer(i4) :: nCols</i>	Number of datapoints in latitudinal direction
in	<i>character(len=*) :: varName</i>	Name of variable name to read
in	<i>logical, dimension(:,) :: mask</i>	mask of valid data fields
out	<i>real(dp), dimension(:, :, :,) :: data</i>	Data matrix dim_1 = longitude, dim_2 = latitude, dim_3 = months, dim_4 = hours
in	<i>real(dp), optional, intent(in) :: lower</i>	Lower bound for check of validity of data values
in	<i>real(dp), optional, intent(in) :: upper</i>	Upper bound for check of validity of data values
in	<i>logical, optional, intent(in) :: nocheck</i>	.TRUE. if check for nodata values deactivated default = .FALSE. - check is done
in	<i>logical, dimension(:, :, :,), allocatable, optional, intent(out) :: maskout</i>	! mask of valid data points

Author

Stephan Thober & Matthias Zink

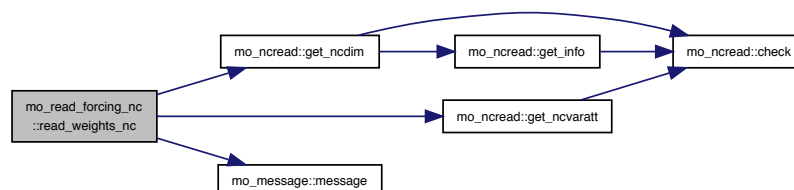
Date

Jan 2017

References mo_kind::dp, mo_ncread::get_ncdim(), mo_ncread::get_ncvaratt(), mo_kind::i4, and mo_message::message().

Referenced by mo_meteo_forcings::meteo_weights_wrapper().

Here is the call graph for this function:



15.59.2.1 read_latlon()

```
subroutine, public mo_read_latlon::read_latlon (
    integer(i4), intent(in) ii )
```

reads latitude and longitude coordinates

reads latitude and longitude coordinates from netcdf file for each basin and appends it to the global variables latitude and longitude.

Parameters

in	<i>integer(i4) :: ii</i>	basin index File name of the basins must be xxx_latlon.nc, where xxx is the basin id. Variable names in the netcdf file have to be 'lat' for latitude and 'lon' for longitude.
----	--------------------------	--

Author

Stephan Thober

Date

Nov 2013

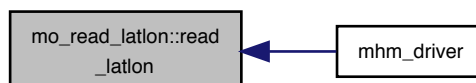
References mo_global_variables::basin, mo_global_variables::filelatlon, mo_global_variables::l0_latitude, mo_global_variables::l0_longitude, mo_global_variables::l1_latitude, mo_global_variables::l1_longitude, mo_global_variables::l1_rect_latitude, mo_global_variables::l1_rect_longitude, mo_global_variables::level0, mo_global_variables::level1, and mo_message::message().

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.60 mo_read_lut Module Reference

Routines reading lookup tables (lut).

Functions/Subroutines

- subroutine, public [read_geoformation_lut](#) (filename, fileunit, nGeo, geo_unit, geo_karstic)
Reads LUT containing geological formation information.
- subroutine, public [read_lai_lut](#) (filename, fileunit, nLAI, LAIDlist, LAI)
Reads LUT containing LAI information.

15.60.1 Detailed Description

Routines reading lookup tables (lut).

This module contains routines reading various lookup tables (lut).

- (1) LUT containing gauge information.
- (2) LUT containing geological formation information.
- (3) LUT containing LAI class information.

Authors

Juliane Mai, Matthias Zink

Date

Jan 2013

15.60.2 Function/Subroutine Documentation

15.60.2.1 read_geoformation_lut()

```
subroutine, public mo_read_lut::read_geoformation_lut (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(out) nGeo,
    integer(i4), dimension(:), intent(out), allocatable geo_unit,
    integer(i4), dimension(:), intent(out), allocatable geo_karstic )
```

Reads LUT containing geological formation information.

The LUT needs to have the following header:

```
nGeo_Formations < Number of lines containing data >
GeoParam(i)    ClassUnit    Karstic    Description
```

The subsequent lines contains the geological formation information:

```
<GeoParam(i)> <ClassUnit_i4> <Karstic_i4> <Description_char>
```

All following lines will be discarded while reading.

GeoParam is a running index while ClassUnit is the unit of the map containing the geological formations such that it does not necessarily contains subsequent numbers. The parametrization of this unit is part of the namelist mhm_parameter.nml under <geoparameter>.

Parameters

in	<i>character(len=*) :: filename</i>	File name of LUT
in	<i>integer(i4) :: fileunit</i>	Unit to open file
out	<i>integer(i4) :: nGeo</i>	Number of geological formations
out	<i>integer(i4), dimension(:), allocatable :: geo_unit</i>	List of id numbers of each geological formations
out	<i>integer(i4), dimension(:), allocatable :: geo_karstic</i>	ID of the Karstic formation (0 == does not exist)

Author

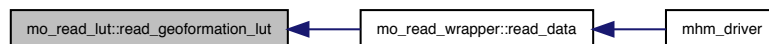
Juliane Mai

Date

Jan 2013

Referenced by mo_read_wrapper::read_data().

Here is the caller graph for this function:



15.60.2.2 read_lai_lut()

```

subroutine, public mo_read_lut::read_lai_lut (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(out) nLAI,
    integer(i4), dimension(:), intent(out), allocatable LAIIDlist,
    real(dp), dimension(:, :), intent(out), allocatable LAI )

```

Reads LUT containing LAI information.

The LUT needs to have the following header:

```

NoLAIclasses <Number of lines containing data>
Id land-use Jan. Feb. Mar. Apr. May Jun. Jul. Aug. Sep. Oct. Nov. De

```

The subsequent lines contains the lai class information:

```

<ID_i4> <landuse_char> <val_1_dp> <val_2_dp> <val_3_dp> <val_4_dp> ... <val_12_dp>

```

All following lines will be discarded while reading.

Parameters

in	<i>character(len=*) :: filename</i>	File name of LUT
----	-------------------------------------	------------------

Parameters

in	<i>integer(i4) :: fileunit</i>	Unit to open file
out	<i>integer(i4) :: nLAI</i>	Number of LAI classes
out	<i>integer(i4), dimension(:), allocatable :: LAIIDlist</i>	List of ids of LAI classes
out	<i>real(dp), dimension(:, :), allocatable :: LAI</i>	LAI per class (row) and month (col)

Author

Juliane Mai

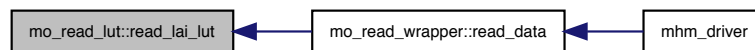
Date

Jan 2013

References `mo_kind::i4`, and `mo_mhm_constants::yearmonths`.

Referenced by `mo_read_wrapper::read_data()`.

Here is the caller graph for this function:



15.61 mo_read_meteo Module Reference

Reads meteorological input data.

Functions/Subroutines

- subroutine, public [read_meteo_bin](#) (folder, nRows, nCols, periode, data, mask, lower, upper)
Reads binary meteorological files.

15.61.1 Detailed Description

Reads meteorological input data.

This module is to read meteorological input data, e.g. temperature, precipitation.

The module provides a subroutine for binary files.

First, the dimensions given are cross-checked with header.txt information. Second, the data of the specified period are read from the specified directory. The names of files in this directory have to be always "YYYY.bin".

If the optional lower and/or upper bound for the data values is given, the read data are checked for validity. The program is stopped if any value lies out of range.

Authors

Juliane Mai

Date

Dec 2012

15.61.2 Function/Subroutine Documentation

15.61.2.1 read_meteo_bin()

```
subroutine, public mo_read_meteo::read_meteo_bin (
    character(len=*), intent(in) folder,
    integer(i4), intent(in) nRows,
    integer(i4), intent(in) nCols,
    type(period), intent(in) periode,
    real(dp), dimension(:, :, :), intent(out), allocatable data,
    logical, dimension(:, :), intent(in) mask,
    real(dp), intent(in), optional lower,
    real(dp), intent(in), optional upper )
```

Reads binary meteorological files.

Reads binary meteorological files.

First, the dimensions given are cross-checked with header.txt information. Second, the data of the specified period are read from the specified directory. The names of files in this directory have to be always "YYYY.bin" (file ending .bin can be set in [mo_file](#)).

If the optional lower and/or upper bound for the data values is given, the read data are checked for validity. The program is stopped if any value lies out of range.

Parameters

in	<i>character(len=*) :: folder</i>	Name of the folder where data are stored
in	<i>integer(i4) :: nRows</i>	Number of datapoints in longitudinal direction
in	<i>integer(i4) :: nCols</i>	Number of datapoints in latitudinal direction
in	<i>type(period) :: periode</i>	Period the data are needed for
in	<i>logical, dimension(:, :) :: mask</i>	Mask of valid field for checking lower and upper bounds
out	<i>real(dp), dimension(:, :, :) :: data</i>	Data matrix dim_1 = longitude, dim_2 = latitude, dim_3 = time
in	<i>real(dp), optional) :: lower</i>	Lower bound for check of validity of data values
in	<i>real(dp), optional) :: upper</i>	Upper bound for check of validity of data values

Note

The values stored in YYYY.bin are of single precision, i.e. 4 bytes per value.

Author

Juliane Mai

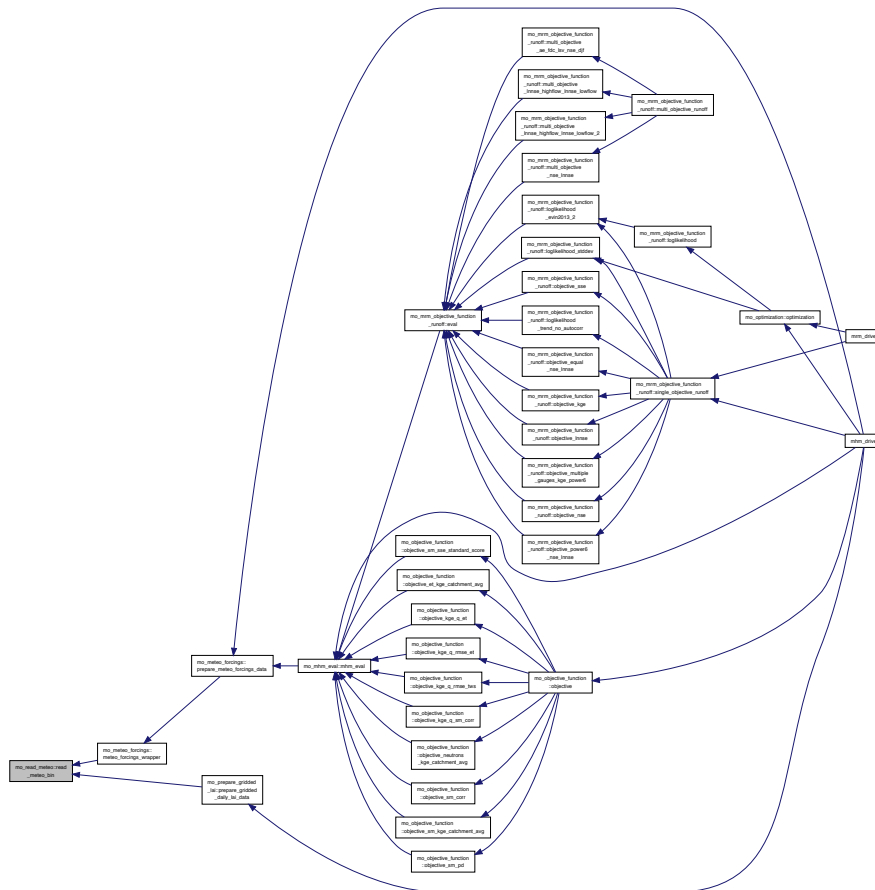
Date

Dec 2012

References `mo_julian::date2dec()`, `mo_kind::dp`, `mo_file::file_meteo_binary_end`, `mo_file::file_meteo_header`, `mo_↵_kind::i4`, `mo_message::message()`, `mo_kind::sp`, `mo_file::umeteo`, and `mo_file::umeteo_header`.

Referenced by `mo_meteo_forcings::meteo_forcings_wrapper()`, and `mo_prepare_gridded_lai::prepare_gridded_↵daily_lai_data()`.

Here is the caller graph for this function:



Read optional data for mHM calibration.

Functions/Subroutines

- subroutine, public [read_soil_moisture](#) (iBasin)
Read soil moisture data from NetCDF file for calibration.
- subroutine, public [read_basin_avg_tws](#) ()
Read basin average TWS timeseries from file, the same way runoff is read.
- subroutine, public [read_neutrons](#) (iBasin)
Read neutrons data from NetCDF file for calibration.
- subroutine, public [read_evapotranspiration](#) (iBasin)
Read evapotranspiration data from NetCDF file for calibration.

15.62.1 Detailed Description

Read optional data for mHM calibration.

Data have to be provided in resolution of the hydrology.

Authors

Matthias Zink

Date

Mar 2015

15.62.2 Function/Subroutine Documentation

15.62.2.1 read_basin_avg_tws()

```
subroutine, public mo_read_optional_data::read_basin_avg_tws ( )
```

Read basin average TWS timeseries from file, the same way runoff is read.

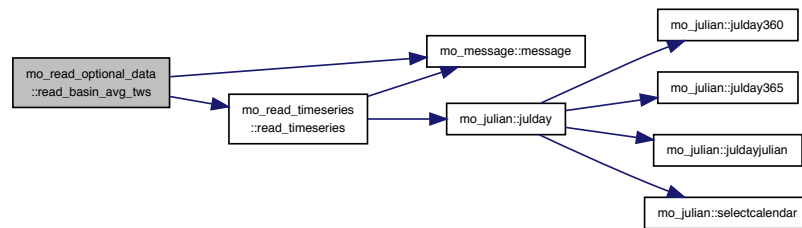
Read basin average TWS timeseries Allocate global basin_avg_TWS variable that contains the simulated values after the simulation.

Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

References `mo_global_variables::basin_avg_tws_obs`, `mo_global_variables::basin_avg_tws_sim`, `mo_message`↔`::message()`, `mo_global_variables::nbasins`, `mo_mhm_constants::nodata_dp`, `mo_common_variables::optimize`, `mo_read_timeseries::read_timeseries()`, and `mo_file::utws`.

Here is the call graph for this function:



15.62.2.2 read_evapotranspiration()

```
subroutine, public mo_read_optional_data::read_evapotranspiration (
    integer(i4), intent(in) iBasin )
```

Read evapotranspiration data from NetCDF file for calibration.

This routine reads observed evapotranspiration fields which are used for model calibration. The evapotranspiration file is expected to be called "et.nc" with a variable "et" inside. The data are read only for the evaluation period they are intended to be used for calibration. Evapotranspiration data are only read if one of the corresponding objective functions is chosen.

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

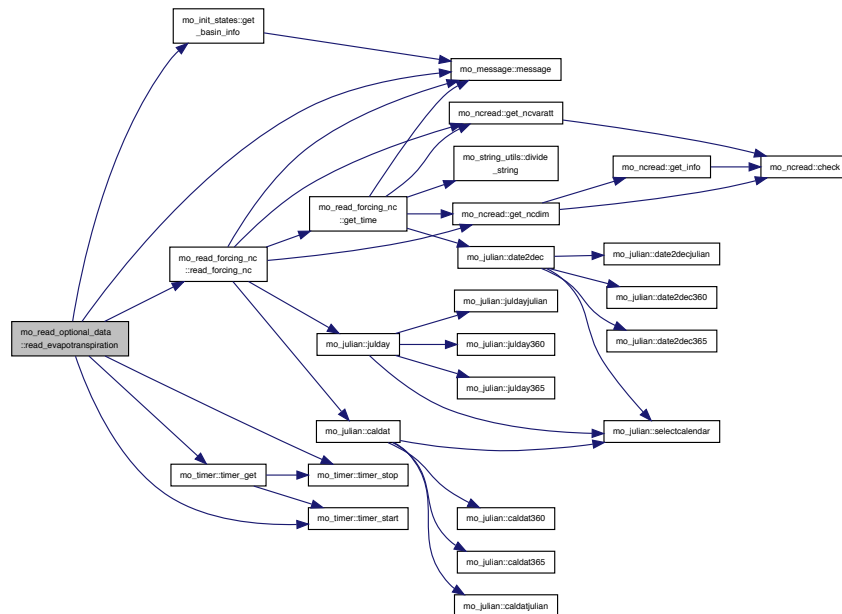
Johannes Brenner

Date

Feb 2017

References `mo_global_variables::direvapotranspiration`, `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_constants::nodata_dp`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_timer::timer_get()`, `mo_timer::timer_start()`, and `mo_timer::timer_stop()`.

Here is the call graph for this function:



15.62.2.3 read_neutrons()

```

subroutine, public mo_read_optional_data::read_neutrons (
    integer(i4), intent(in) iBasin )

```

Read neutrons data from NetCDF file for calibration.

This routine reads observed neutron fields which are used for model calibration. The neutrons file is expected to be called "neutrons.nc" with a variable "neutrons" inside. The data are read only for the evaluation period they are intended to be used for calibration. Neutrons data are only read if one of the corresponding objective functions is chosen.

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

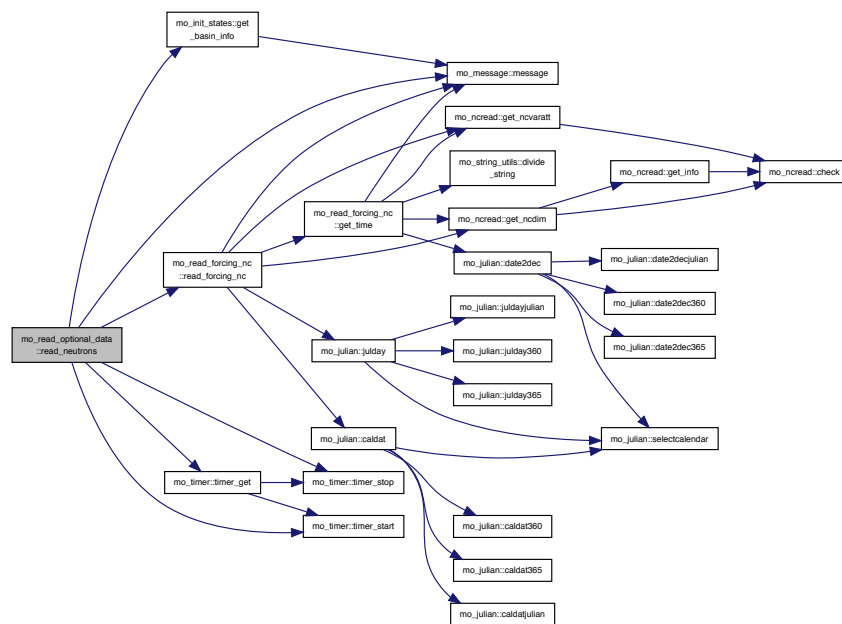
Martin Schroen

Date

Jul 2015

References `mo_global_variables::dirneutrons`, `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_constants::nodata_dp`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_timer::timer_get()`, `mo_timer::timer_start()`, and `mo_timer::timer_stop()`.

Here is the call graph for this function:



15.62.2.4 read_soil_moisture()

```
subroutine, public mo_read_optional_data::read_soil_moisture (
    integer(i4), intent(in) iBasin )
```

Read soil moisture data from NetCDF file for calibration.

This routine reads observed soil moisture fields which are used for model calibration. The soil moisture file is expected to be called "sm.nc" with a variable "sm" inside. The data are read only for the evaluation period they are intended to be used for calibration. Soil moisture data are only read if one of the corresponding objective functions is chosen.

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Matthias Zink

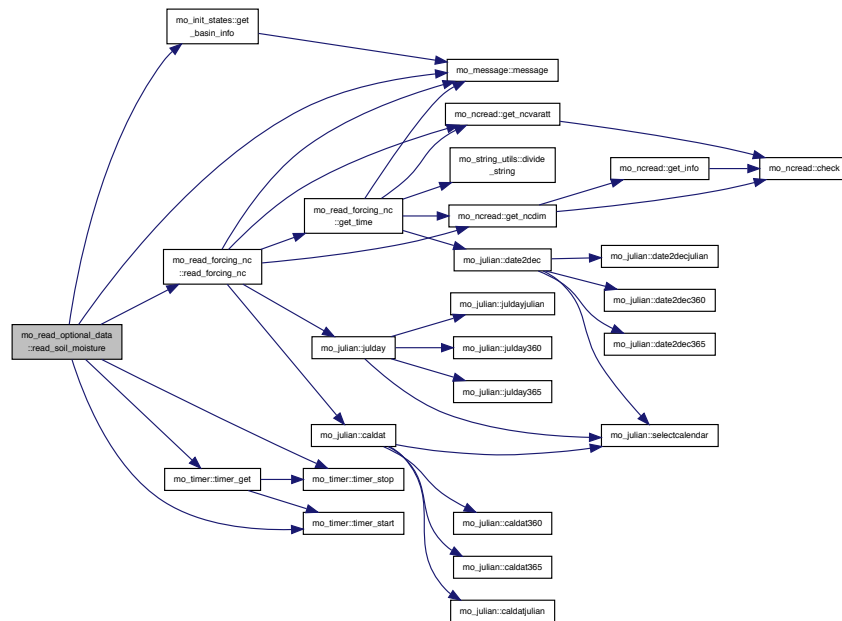
Date

Mar 2015

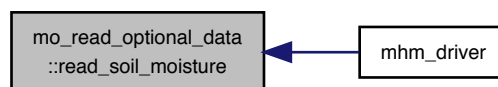
References `mo_global_variables::dirsoil_moisture`, `mo_init_states::get_basin_info()`, `mo_message::message()`, `mo_mhm_constants::nodata_dp`, `mo_read_forcing_nc::read_forcing_nc()`, `mo_timer::timer_get()`, `mo_timer::timer_start()`, and `mo_timer::timer_stop()`.

Referenced by `mhm_driver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.63 mo_read_spatial_data Module Reference

Reads spatial input data.

Data Types

- interface [read_spatial_data_ascii](#)
Reads spatial data files of ASCII format.

Functions/Subroutines

- subroutine [read_spatial_data_ascii_dp](#) (filename, fileunit, header_ncols, header_nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)
- subroutine [read_spatial_data_ascii_i4](#) (filename, fileunit, header_ncols, header_nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)
- subroutine, public [read_header_ascii](#) (filename, fileunit, header_ncols, header_nrows, header_xllcorner, header_yllcorner, header_cellsize, header_nodata)

Reads header lines of ASCII files.

15.63.1 Detailed Description

Reads spatial input data.

This module is to read spatial input data, e.g. dem, aspect, flow direction.

The module provides a subroutine for ASCII files.

(Subroutine for NetCDF files will come with release 5.1).

The data are read from the specified directory.

Authors

Juliane Mai

Date

Dec 2012

15.63.2 Function/Subroutine Documentation

15.63.2.1 read_header_ascii()

```
subroutine, public mo_read_spatial_data::read_header_ascii (
    character(len=*) , intent(in) filename,
    integer(i4) , intent(in) fileunit,
    integer(i4) , intent(out) header_ncols,
    integer(i4) , intent(out) header_nrows,
    real(dp) , intent(out) header_xllcorner,
    real(dp) , intent(out) header_yllcorner,
    real(dp) , intent(out) header_cellsize,
    real(dp) , intent(out) header_nodata )
```

Reads header lines of ASCII files.

Reads header lines of ASCII files, e.g. dem, aspect, flow direction.

Parameters

in	<i>character(len=*) :: filename</i>	Name of file and its location
in	<i>integer(i4) :: fileunit</i>	File unit for open file
out	<i>integer(i4) :: header_ncols</i>	Reference number of columns
out	<i>integer(i4) :: header_nrows</i>	Reference number of rows
out	<i>real(dp) :: header_xllcorner</i>	Reference lower left corner (x)
out	<i>real(dp) :: header_yllcorner</i>	Reference lower left corner (y)
out	<i>integer(i4) :: header_cellsize</i>	Reference cell size [m]
out	<i>real(dp) :: header_nodata</i>	Reference nodata value

Author

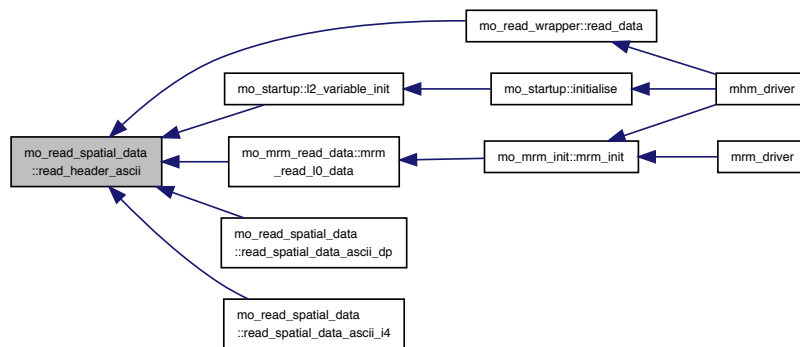
Juliane Mai

Date

Jan 2013

Referenced by `mo_startup::i2_variable_init()`, `mo_mrm_read_data::mrm_read_i0_data()`, `mo_read_wrapper::read_data()`, `read_spatial_data_ascii_dp()`, and `read_spatial_data_ascii_i4()`.

Here is the caller graph for this function:

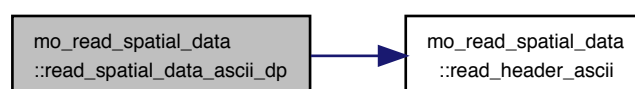
**15.63.2.2 read_spatial_data_ascii_dp()**

```

subroutine mo_read_spatial_data::read_spatial_data_ascii_dp (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(in) header_ncols,
    integer(i4), intent(in) header_nrows,
    real(dp), intent(in) header_xllcorner,
    real(dp), intent(in) header_yllcorner,
    real(dp), intent(in) header_cellsize,
    real(dp), dimension(:, :), intent(out), allocatable data,
    logical, dimension(:, :), intent(out), allocatable mask ) [private]
  
```

References `read_header_ascii()`.

Here is the call graph for this function:

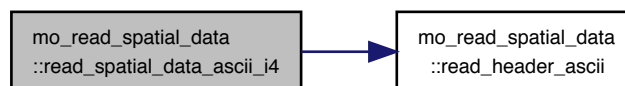


15.63.2.3 read_spatial_data_ascii_i4()

```
subroutine mo_read_spatial_data::read_spatial_data_ascii_i4 (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(in) header_ncols,
    integer(i4), intent(in) header_nrows,
    real(dp), intent(in) header_xllcorner,
    real(dp), intent(in) header_yllcorner,
    real(dp), intent(in) header_cellsize,
    integer(i4), dimension(:, :), intent(out), allocatable data,
    logical, dimension(:, :), intent(out), allocatable mask ) [private]
```

References mo_kind::i4, and read_header_ascii().

Here is the call graph for this function:



15.64 mo_read_timeseries Module Reference

Routines to read files containing timeseries data.

Functions/Subroutines

- subroutine, public [read_timeseries](#) (filename, fileunit, periodStart, periodEnd, optimize, opti_function, data, mask, nMeasPerDay)

Reads time series in ASCII format.

15.64.1 Detailed Description

Routines to read files containing timeseries data.

This routine is reading time series input data for a particular time period. The files need to have a specific header specified in the different routines.

Authors

Matthias Zink, Juliane Mai

Date

Jan 2013

15.64.2 Function/Subroutine Documentation

15.64.2.1 read_timeseries()

```
subroutine, public mo_read_timeseries::read_timeseries (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), dimension(3), intent(in) periodStart,
    integer(i4), dimension(3), intent(in) periodEnd,
    logical, intent(in) optimize,
    integer(i4), intent(in) opti_function,
    real(dp), dimension(:), intent(out), allocatable data,
    logical, dimension(:), intent(out), optional, allocatable mask,
    integer(i4), intent(out), optional nMeasPerDay )
```

Reads time series in ASCII format.

Reads time series in ASCII format. Needs specific header lines:

```
<description>
nodata <nodata value>
n <number of measurements per day> measurements per day [1, 1440]
start <YYYY_i4> <MM_i4> <DD_i4> <HH_i4> <MM_i4> (YYYY MM DD HH MM)
end <YYYY_i4> <MM_i4> <DD_i4> <HH_i4> <MM_i4> (YYYY MM DD HH MM)
```

Line 6 is the first line with data in the following format:

```
<YYYY_i4> <MM_i4> <DD_i4> <HH_i4> <MM_i4> <data_dp>
```

The routine checks for missing data points and if data points are equal distanced. The first data point at each day has to be at HH:MM = 00:00.

Parameters

in	<i>character(len=*) :: filename</i>	File name
in	<i>integer(i4) :: fileunit</i>	Unit to open file
in	<i>integer(i4), dimension(3) :: periodStart</i>	Start day of reading (YYYY,MM,DD)
in	<i>integer(i4), dimension(3) :: periodEnd</i>	End day of reading (YYYY,MM,DD)
in	<i>logical :: optimize</i>	optimization flag
out	<i>real(dp), dimension(:), allocatable :: data</i>	Data vector
out	<i>logical, dimension(:), allocatable, optional :: mask</i>	Mask for nodata values in data
out	<i>integer(i4), optional :: nMeasPerDay</i>	Number of data points per day

Note

Routine reads in only whole days with equidistant time steps between data points.
The first data point has to be at HH:MM = 00:00.

Authors

Matthias Zink, Juliane Mai

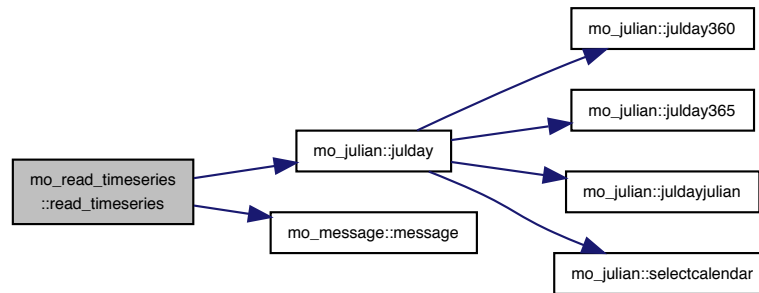
Date

Jan 2013

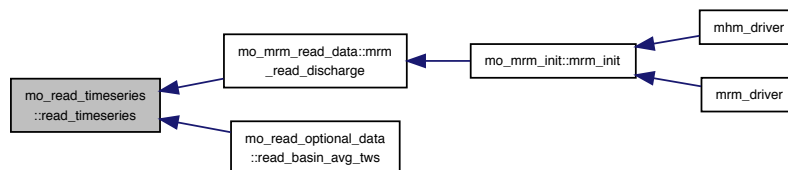
References `mo_julian::julday()`, and `mo_message::message()`.

Referenced by `mo_mrm_read_data::mrm_read_discharge()`, and `mo_read_optional_data::read_basin_avg_tws()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.65 mo_read_wrapper Module Reference

Wrapper for all reading routines.

Functions/Subroutines

- subroutine, public [read_data](#)
Reads data.
- subroutine [check_consistency_lut_map](#) (data, lookuptable, filename, unique_values)
Checks if classes in input maps appear in look up tables.

15.65.1 Detailed Description

Wrapper for all reading routines.

This module is to wrap up all reading routines.

The general written reading routines are used to store now the read data into global variables.

Authors

Juliane Mai, Matthias Zink

Date

Jan 2013

15.65.2 Function/Subroutine Documentation**15.65.2.1 check_consistency_lut_map()**

```
subroutine mo_read_wrapper::check_consistency_lut_map (
    integer(i4), dimension(:), intent(in) data,
    integer(i4), dimension(:), intent(in) lookuptable,
    character(*), intent(in) filename,
    integer(i4), dimension(:), intent(out), optional, allocatable unique_values )
```

Checks if classes in input maps appear in look up tables.

Determines whether a class appearing in the morphological input is occurring in the respective look up table. mHM breaks if inconsistencies are discovered.

Parameters

in	<i>integer(i4), dimension(:) :: data</i>	map of study domain
in	<i>integer(i4), dimension(:) :: lookuptable</i>	look up table corresponding to map
in	<i>character(*) :: filename</i>	name of the lut file - ERRORR warn
out	<i>integer(i4), dimension(:), allocatable :: unique_values</i>	array of unique values in data

Author

Matthias Zink

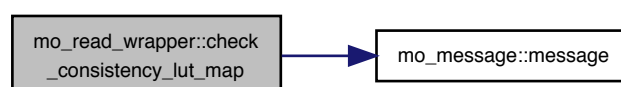
Date

Nov 2016

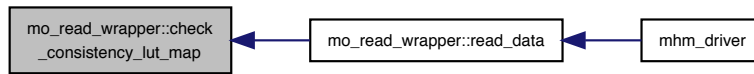
References mo_message::message().

Referenced by read_data().

Here is the call graph for this function:



Here is the caller graph for this function:



15.65.2.2 read_data()

```
subroutine, public mo_read_wrapper::read_data ( )
```

Reads data.

The namelists are already read by read_config call./n All LUTs are read from their respective directory and information within those files are shared across all basins to be modeled.

Note

read_config has to be called before

Author

Juliane Mai & Matthias Zink

Date

Feb 2013

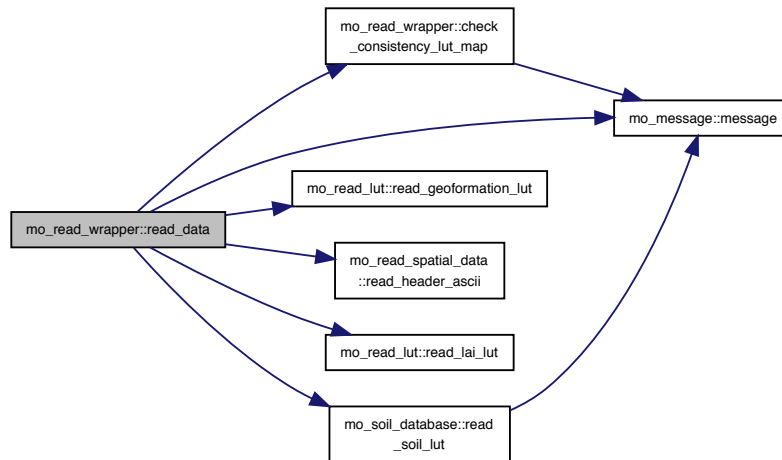
by default; when iFlag_soilDB = 0

by default; when iFlag_soilDB = 0

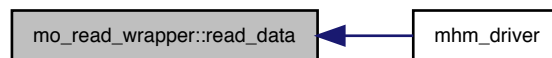
References check_consistency_lut_map(), mo_file::file_geolut, mo_global_variables::geounitkar, mo_global_variables::geounitlist, mo_message::message(), mo_global_variables::ngeounits, mo_mhm_constants::nodata_dp, mo_mhm_constants::nodata_i4, mo_common_variables::processmatrix, mo_read_lut::read_geofomation_lut(), mo_read_spatial_data::read_header_ascii(), mo_read_lut::read_lai_lut(), mo_soil_database::read_soil_lut(), and mo_file::ugeolut.

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.66 mo_restart Module Reference

reading and writing states, fluxes and configuration for restart of mHM.

Functions/Subroutines

- subroutine, public [write_restart_files](#) (OutPath)
write restart files for each basin
- subroutine, public [read_restart_config](#) (iBasin, soilId_isPresent, InPath)
reads configuration apart from Level 11 configuration from a restart directory
- subroutine, public [read_restart_states](#) (iBasin, InPath)
reads fluxes and state variables from file

15.66.1 Detailed Description

reading and writing states, fluxes and configuration for restart of mHM.

routines are seperated for reading and writing variables for:

- states and fluxes, and
- configuration.
Reading of L11 configuration is also separated from the rest, since it is only required when routing is activated.

Authors

Stephan Thober

Date

Jul 2013

15.66.2 Function/Subroutine Documentation**15.66.2.1 read_restart_config()**

```
subroutine, public mo_restart::read_restart_config (
    integer(i4), intent(in) iBasin,
    integer(i4), dimension(:), intent(inout), allocatable soilId_isPresent,
    character(256), intent(in) InPath )
```

reads configuration apart from Level 11 configuration from a restart directory

read configuration variables from a given restart directory and initializes all configuration variables, that are initialized in the subroutine initialise, contained in module [mo_startup](#).

Parameters

in	<i>integer(i4) :: iBasin</i>	number of basin
in	<i>character(256) :: InPath</i>	Input Path including trailing slash

Note

Restart Files must have the format, as if it would have been written by subroutine `write_restart_files`

Author

Stephan Thober

Date

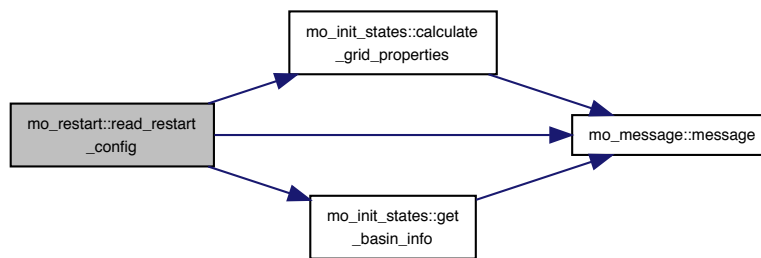
Apr 2013

by default; when iFlag_soilDB = 0

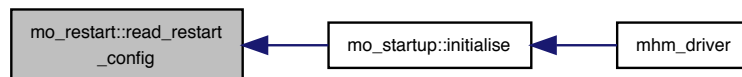
References mo_init_states::calculate_grid_properties(), mo_kind::dp, mo_init_states::get_basin_info(), mo_kind::i4, mo_global_variables::!0_basin, mo_message::message(), and mo_mhm_constants::nodata_dp.

Referenced by mo_startup::initialise().

Here is the call graph for this function:



Here is the caller graph for this function:



15.66.2.2 read_restart_states()

```

subroutine, public mo_restart::read_restart_states (
    integer(i4), intent(in) iBasin,
    character(256), intent(in) InPath )
  
```

reads fluxes and state variables from file

read fluxes and state variables from given restart directory and initialises all state variables that are initialized in the subroutine initialise, contained in module [mo_startup](#).

Parameters

in	<i>integer(i4) :: iBasin</i>	number of basin
in	<i>character(256) :: InPath</i>	Input Path including trailing slash

Note

Restart Files must have the format, as if it would have been written by subroutine `write_restart_files`

Author

Stephan Thober

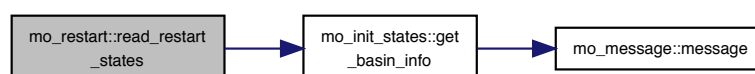
Date

Apr 2013

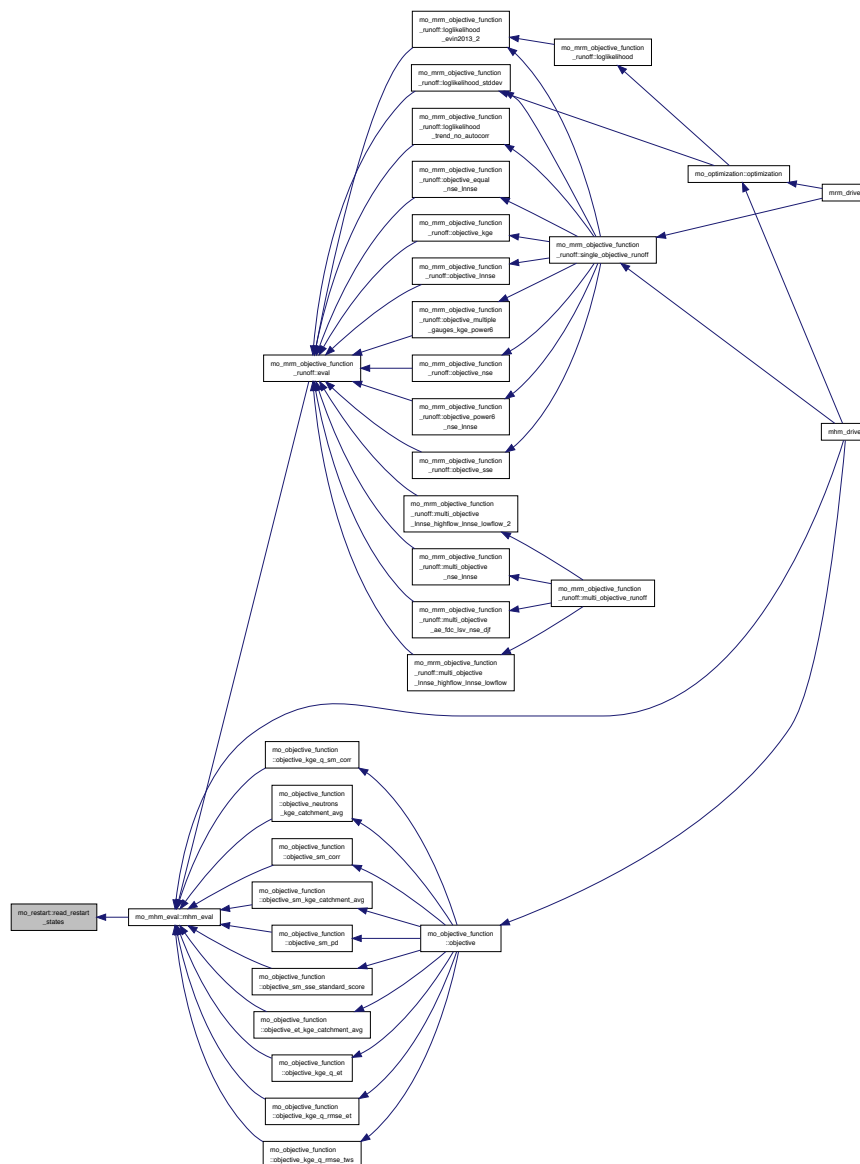
References `mo_kind::dp`, `mo_init_states::get_basin_info()`, `mo_kind::i4`, `mo_global_variables::l1_aeroresist`, `mo_global_variables::l1_aetcanopy`, `mo_global_variables::l1_aetsealed`, `mo_global_variables::l1_aetsoil`, `mo_global_variables::l1_alpha`, `mo_global_variables::l1_baseflow`, `mo_global_variables::l1_degday`, `mo_global_variables::l1_degdayinc`, `mo_global_variables::l1_degdaymax`, `mo_global_variables::l1_degdaynopre`, `mo_global_variables::l1_fasp`, `mo_global_variables::l1_fastrunoff`, `mo_global_variables::l1_fforest`, `mo_global_variables::l1_fperm`, `mo_global_variables::l1_froots`, `mo_global_variables::l1_fsealed`, `mo_global_variables::l1_harsamcoeff`, `mo_global_variables::l1_infilsoil`, `mo_global_variables::l1_inter`, `mo_global_variables::l1_jarvis_thresh_c1`, `mo_global_variables::l1_karstloss`, `mo_global_variables::l1_kbaseflow`, `mo_global_variables::l1_kfastflow`, `mo_global_variables::l1_kperco`, `mo_global_variables::l1_kslowflow`, `mo_global_variables::l1_maxinter`, `mo_global_variables::l1_melt`, `mo_global_variables::l1_percol`, `mo_global_variables::l1_petlaicorfactor`, `mo_global_variables::l1_preeffect`, `mo_global_variables::l1_prietayalpha`, `mo_global_variables::l1_rain`, `mo_global_variables::l1_runoffseal`, `mo_global_variables::l1_satstw`, `mo_global_variables::l1_sealedthresh`, `mo_global_variables::l1_sealstw`, `mo_global_variables::l1_slowrunoff`, `mo_global_variables::l1_snow`, `mo_global_variables::l1_snowpack`, `mo_global_variables::l1_soilmoist`, `mo_global_variables::l1_soilmoistexp`, `mo_global_variables::l1_soilmoistfc`, `mo_global_variables::l1_soilmoistsat`, `mo_global_variables::l1_surfresist`, `mo_global_variables::l1_tempthresh`, `mo_global_variables::l1_throughfall`, `mo_global_variables::l1_total_runoff`, `mo_global_variables::l1_unsatstw`, `mo_global_variables::l1_unsatthresh`, `mo_global_variables::l1_wiltingpoint`, `mo_global_variables::nsoilhorizons_mhm`, `mo_common_variables::processmatrix`, and `mo_mhm_constants::yearmonths_i4`.

Referenced by `mo_mhm_eval::mhm_eval()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.66.2.3 write_restart_files()

```
subroutine, public mo_restart::write_restart_files (
    character(256), dimension(:), intent(in) OutPath )
```

write restart files for each basin

write restart files for each basin. For each basin three restart files are written. These are xxx_states.nc, xxx_L11↔_config.nc, and xxx_config.nc (xxx being the three digit basin index). If a variable is added here, it should also be added in the read restart routines below.

Parameters

in	character(256), dimension(:) :: OutPath	Output Path for each basin
----	---	----------------------------

Author

Stephan Thober

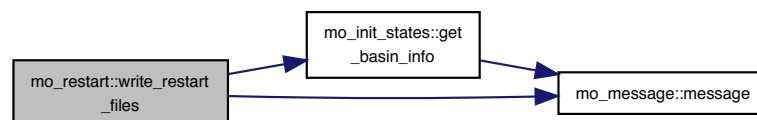
Date

Jun 2014

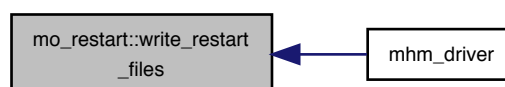
References `mo_global_variables::basin`, `mo_kind::dp`, `mo_init_states::get_basin_info()`, `mo_kind::i4`, `mo_global_variables::l0_cellcoor`, `mo_global_variables::l0_id`, `mo_global_variables::l1_aeroresist`, `mo_global_variables::l1_aetcanopy`, `mo_global_variables::l1_aetsealed`, `mo_global_variables::l1_aetsoil`, `mo_global_variables::l1_alpha`, `mo_global_variables::l1_baseflow`, `mo_global_variables::l1_degday`, `mo_global_variables::l1_degdayinc`, `mo_global_variables::l1_degdaymax`, `mo_global_variables::l1_degdaynopre`, `mo_global_variables::l1_fasp`, `mo_global_variables::l1_fastrunoff`, `mo_global_variables::l1_fforest`, `mo_global_variables::l1_fperm`, `mo_global_variables::l1_froots`, `mo_global_variables::l1_fsealed`, `mo_global_variables::l1_harsamcoeff`, `mo_global_variables::l1_infilsoil`, `mo_global_variables::l1_inter`, `mo_global_variables::l1_jarvis_thresh_c1`, `mo_global_variables::l1_karstloss`, `mo_global_variables::l1_kbaseflow`, `mo_global_variables::l1_kfastflow`, `mo_global_variables::l1_kperco`, `mo_global_variables::l1_kslowflow`, `mo_global_variables::l1_maxinter`, `mo_global_variables::l1_melt`, `mo_global_variables::l1_percol`, `mo_global_variables::l1_petlaicorfactor`, `mo_global_variables::l1_preeffect`, `mo_global_variables::l1_prietayalpha`, `mo_global_variables::l1_rain`, `mo_global_variables::l1_runoffseal`, `mo_global_variables::l1_satstw`, `mo_global_variables::l1_sealedthresh`, `mo_global_variables::l1_sealstw`, `mo_global_variables::l1_slowrunoff`, `mo_global_variables::l1_snow`, `mo_global_variables::l1_snowpack`, `mo_global_variables::l1_soilmoist`, `mo_global_variables::l1_soilmoistexp`, `mo_global_variables::l1_soilmoistfc`, `mo_global_variables::l1_soilmoistsat`, `mo_global_variables::l1_surfresist`, `mo_global_variables::l1_tempthresh`, `mo_global_variables::l1_throughfall`, `mo_global_variables::l1_total_runoff`, `mo_global_variables::l1_unsatstw`, `mo_global_variables::l1_unsatthresh`, `mo_global_variables::l1_wiltingpoint`, `mo_message::message()`, `mo_mhm_constants::nodata_dp`, `mo_mhm_constants::nodata_i4`, and `mo_common_variables::processmatrix`.

Referenced by `mhm_driver()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.67 mo_runoff Module Reference

Runoff generation for the unsaturated zone, saturated zone (or groundwater zone), and runoff accumulation.

Functions/Subroutines

- subroutine, public [runoff_unsat_zone](#) (k1, kp, k0, alpha, karst_loss, pefec_soil, unsat_thresh, sat_storage, unsat_storage, slow_interflow, fast_interflow, perc)
Runoff generation for the saturated zone.
- subroutine, public [runoff_sat_zone](#) (k2, sat_storage, baseflow)
Runoff generation for the saturated zone.
- subroutine, public [l1_total_runoff](#) (fSealed_area_fraction, fast_interflow, slow_interflow, baseflow, direct_runoff, total_runoff)
total runoff accumulation at level 1

15.67.1 Detailed Description

Runoff generation for the unsaturated zone, saturated zone (or groundwater zone), and runoff accumulation.

This module generates the runoff for the unsaturated and saturated zones and provides runoff accumulation.

Authors

Vladyslav Prykhodko

Date

Dec 2012

15.67.2 Function/Subroutine Documentation

15.67.2.1 l1_total_runoff()

```
subroutine, public mo_runoff::l1_total_runoff (
    real(dp), intent(in) fSealed_area_fraction,
    real(dp), intent(in) fast_interflow,
    real(dp), intent(in) slow_interflow,
    real(dp), intent(in) baseflow,
    real(dp), intent(in) direct_runoff,
    real(dp), intent(out) total_runoff )
```

total runoff accumulation at level 1

Accumulates runoff.

$$q_T = (q_0 + q_1 + q_2) * (1 - fSealed) + q_D * fSealed$$

, where fSealed is the fraction of sealed area.

Parameters

in	"real(dp)	:: fSealed_area_fraction sealed area fraction [1]
in	"real(dp)	:: fast_interflow q_0 Fast runoff component [mm tst-1]
in	"real(dp)	:: slow_interflow q_1 Slow runoff component [mm tst-1]

15.67.2.2 runoff_sat_zone()

```
subroutine, public mo_runoff::runoff_sat_zone (  
    real(dp), intent(in) k2,  
    real(dp), intent(inout) sat_storage,  
    real(dp), intent(out) baseflow )
```

Runoff generation for the saturated zone.

Calculates the runoff generation for the saturated zone. If the level of the ground water reservoir is zero, then the baseflow is also zero. If the level of the ground water reservoir is greater than zero, then the baseflow is equal to baseflow recession coefficient times the level of the ground water reservoir, which will be then reduced by the value of baseflow.

Parameters

in	<i>real(dp) :: k2</i>	Baseflow recession coefficient [d-1]
in, out	<i>real(dp) :: sat_storage</i>	Groundwater storage [mm]
out	<i>real(dp) :: baseflow</i>	Baseflow [mm d-1]

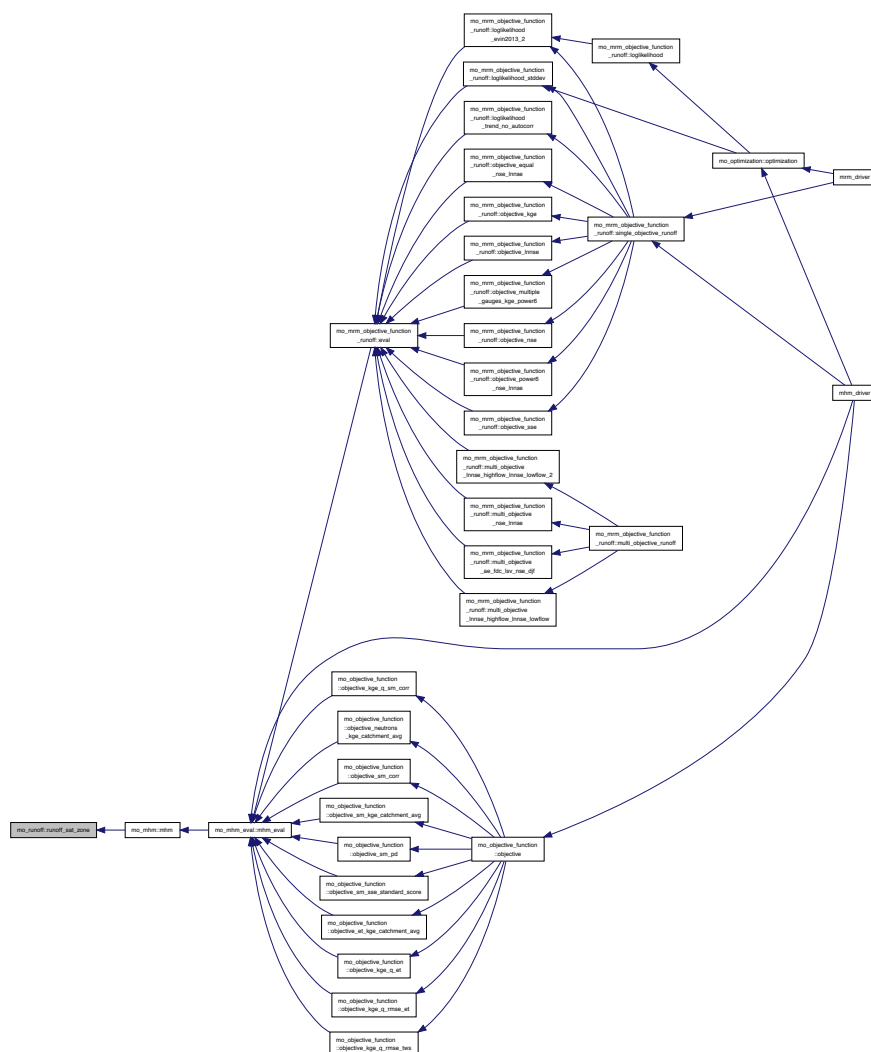
Author

Vladyslav Prykhodko

Date

Dec 2012

Referenced by mo_mhm::mhm().



Runoff generation for the saturated zone.

Calculates the runoff generation for the unsaturated zone.
 Calculates percolation, interflow and baseflow.
 Updates upper soil and groundwater storages.

Parameters

in	<i>real(dp) :: pefec_soil</i>	Input to the soil layer [mm]
in	<i>real(dp) :: k0</i>	Recession coefficient of the upper reservoir, upper outlet [d-1]
in	<i>real(dp) :: k1</i>	Recession coefficient of the upper reservoir, lower outlet [d-1]
in	<i>real(dp) :: kp</i>	Percolation coefficient [d-1]
in	<i>real(dp) :: alpha</i>	Exponent for the upper reservoir [-]
in	<i>real(dp) :: unsat_thresh</i>	Threshold water depth in upper reservoir (for Runoff contribution) [mm]
in	<i>real(dp) :: karst_loss</i>	Karstic percolation loss [-]
in, out	<i>"real(dp)"</i>	:: unsat_storage Upper soil storage [mm]
in, out	<i>real(dp) :: sat_storage</i> Groundwater storage [mm] \param[out], <i>real(dp)</i>	:: perc Percolation [mm d-1]
out	<i>"real(dp)"</i>	:: fast_interflow Fast runoff component [mm d-1]
out	<i>"real(dp)"</i>	:: slow_interflow Slow runoff component [mm d-1]

Author

Vladyslav Prykhodko

Date

Dec 2012

References mo_constants::eps_dp.

Referenced by mo_mhm::mhm().

[illegible]

Shuffled Complex Evolution optimization algorithm.

- `real(dp)` function, `dimension(size(pini, 1))`, public [sce](#) (`functn`, `pini`, `prange`, `mymaxn`, `mymaxit`, `mykstop`, `mypcento`, `mypeps`, `myseed`, `myngs`, `mynpg`, `mynps`, `mynspl`, `mymings`, `myiniflg`, `myprint`, `mymask`, `myalpha`, `mybeta`, `tmp_file`, `popul_file`, `popul_file_append`, `parallel`, `restart`, `restart_file`, `bestf`, `neval`, `history`)
 - *Shuffled Complex Evolution (SCE) algorithm for global optimization.*
- subroutine [parstt](#) (`x`, `bound`, `peps`, `mask`, `xnstd`, `gnrng`, `ipcnvg`)
- subroutine [comp](#) (`ngs2`, `npg`, `a`, `af`, `b`, `bf`)
- subroutine [sort_matrix](#) (`rb`, `ra`)
- subroutine [chkcst](#) (`x`, `bl`, `bu`, `mask`, `ibound`)
- subroutine [getpnt](#) (`idist`, `bl`, `bu`, `std`, `xi`, `mask`, `save_state`, `x`)
- subroutine [cce](#) (`s`, `sf`, `bl`, `bu`, `maskpara`, `xnstd`, `icall`, `maxn`, `maxit`, `save_state_gauss`, `functn`, `alpha`, `beta`, `history`, `idot`)

15.68.1 Detailed Description

Shuffled Complex Evolution optimization algorithm.

Optimization algorithm using Shuffled Complex Evolution strategy. Original version 2.1 of Qingyun Duan (1992) rewritten in Fortran 90.

Authors

Juliane Mai

Date

Feb 2013

15.68.2 Function/Subroutine Documentation

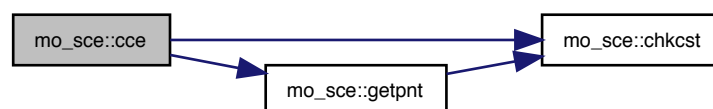
15.68.2.1 cce()

```
subroutine mo_sce::cce (
    real(dp), dimension(:, :), intent(inout) s,
    real(dp), dimension(:), intent(inout) sf,
    real(dp), dimension(:), intent(in) bl,
    real(dp), dimension(:), intent(in) bu,
    logical, dimension(:), intent(in) maskpara,
    real(dp), dimension(:), intent(in) xnstd,
    integer(i8), intent(inout) icall,
    integer(i8), intent(in) maxn,
    logical, intent(in) maxit,
    integer(i8), dimension(n_save_state), intent(inout) save_state_gauss,
    functn,
    real(dp), intent(in) alpha,
    real(dp), intent(in) beta,
    real(dp), dimension(maxn), intent(inout) history,
    logical, intent(in) idot )
```

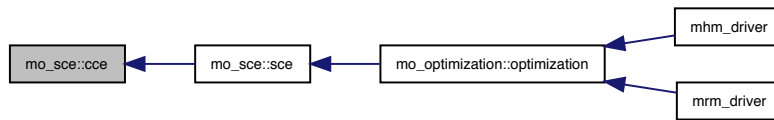
References chkfst(), mo_kind::dp, getpnt(), mo_kind::i4, mo_kind::i8, and mo_xor4096::n_save_state.

Referenced by sce().

Here is the call graph for this function:



Here is the caller graph for this function:



15.68.2.2 chkfst()

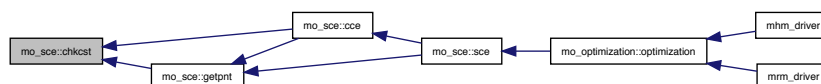
```

subroutine mo_sce::chkfst (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) bl,
    real(dp), dimension(:), intent(in) bu,
    logical, dimension(:), intent(in) mask,
    integer(i4), intent(out) ibound )
  
```

References `mo_kind::dp`, and `mo_kind::i4`.

Referenced by `cce()`, and `getpnt()`.

Here is the caller graph for this function:



15.68.2.3 comp()

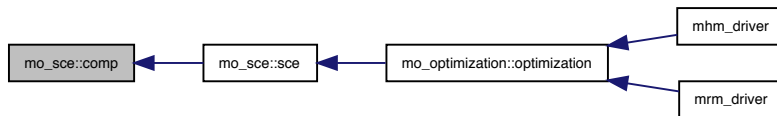
```

subroutine mo_sce::comp (
    integer(i4), intent(in) ngs2,
    integer(i4), intent(in) npg,
    real(dp), dimension(:, :), intent(inout) a,
    real(dp), dimension(:), intent(inout) af,
    real(dp), dimension(size(a,1), size(a,2)), intent(out) b,
    real(dp), dimension(size(af)), intent(out) bf )
  
```

References `mo_kind::dp`, and `mo_kind::i4`.

Referenced by `sce()`.

Here is the caller graph for this function:



15.68.2.4 getpnt()

```

subroutine mo_sce::getpnt (
    integer(i4), intent(in) idist,
    real(dp), dimension(:), intent(in) bl,
    real(dp), dimension(:), intent(in) bu,
    real(dp), dimension(:), intent(in) std,
    real(dp), dimension(:), intent(in) xi,
    logical, dimension(:), intent(in) mask,
    integer(i8), dimension(n_save_state), intent(inout) save_state,
    real(dp), dimension(size(xi,1)), intent(out) x )
  
```

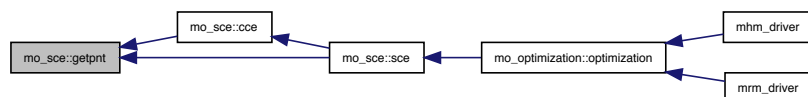
References `chkcst()`, `mo_kind::dp`, `mo_kind::i4`, `mo_kind::i8`, and `mo_xor4096::n_save_state`.

Referenced by `cce()`, and `sce()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.68.2.5 parstt()

```

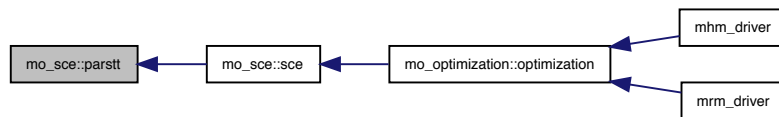
subroutine mo_sce::parstt (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:), intent(in) bound,
    real(dp), intent(in) peps,
    logical, dimension(:), intent(in) mask,
    real(dp), dimension(size(bound,1)), intent(out) xnstd,
    real(dp), intent(out) gnrng,
    integer(i4), intent(out) ipcngv ) [private]

```

References mo_kind::dp, and mo_kind::i4.

Referenced by sce().

Here is the caller graph for this function:



15.68.2.6 sce()

```

real(dp) function, dimension(size(pini,1)), public mo_sce::sce (
    functn,
    real(dp), dimension(:), intent(in) pini,
    real(dp), dimension(:,:), intent(in) prange,
    integer(i8), intent(in), optional mymaxn,
    logical, intent(in), optional mymaxit,
    integer(i4), intent(in), optional mykstop,
    real(dp), intent(in), optional mypcento,
    real(dp), intent(in), optional mypeps,
    integer(i8), intent(in), optional myseed,
    integer(i4), intent(in), optional myngs,
    integer(i4), intent(in), optional mynpg,
    integer(i4), intent(in), optional mynps,
    integer(i4), intent(in), optional mynspl,
    integer(i4), intent(in), optional mymings,
    integer(i4), intent(in), optional myiniflg,
    integer(i4), intent(in), optional myprint,
    logical, dimension(size(pini,1)), intent(in), optional mymask,
    real(dp), intent(in), optional myalpha,
    real(dp), intent(in), optional mybeta,
    character(len=*), intent(in), optional tmp_file,
    character(len=*), intent(in), optional popul_file,
    logical, intent(in), optional popul_file_append,
    logical, intent(in), optional parallel,
    logical, intent(in), optional restart,
    character(len=*), intent(in), optional restart_file,
    real(dp), intent(out), optional bestf,

```

```
integer(i8), intent(out), optional neval,
real(dp), dimension(:), intent(out), optional, allocatable history )
```

Shuffled Complex Evolution (SCE) algorithm for global optimization.

Shuffled Complex Evolution method for global optimization
– version 2.1

by Qingyun Duan
Department of Hydrology & Water Resources
University of Arizona, Tucson, AZ 85721
(602) 621-9360, email: duan@hwr.arizona.edu

Written by Qingyun Duan, Oct 1990.
Revised by Qingyun Duan, Aug 1991.
Revised by Qingyun Duan, Apr 1992.

Re-written by Juliane Mai, Feb 2013.

Statement by Qingyun Duan:

This general purpose global optimization program is developed at the Department of Hydrology & Water Resources of the University of Arizona. Further information regarding the SCE-UA method can be obtained from Dr. Q. Duan, Dr. S. Sorooshian or Dr. V.K. Gupta at the address and phone number listed above. We request all users of this program make proper reference to the paper entitled 'Effective and Efficient Global Optimization for Conceptual Rainfall-runoff Models' by Duan, Q., S. Sorooshian, and V.K. Gupta, Water Resources Research, Vol 28(4), pp.1015-1031, 1992.

The function to be minimized is the first argument of DDS and must be defined as

```
function functn(p)
  use mo_kind, only: dp
  implicit none
  real(dp), dimension(:), intent(in) :: p
  real(dp) :: functn
end function functn
```

Parameters

in	<i>real(dp) :: functn(p)</i>	Function on which to search the optimum
in	<i>real(dp) :: pini(:)</i>	initial value of decision variables
in	<i>real(dp) :: prange(size(pini),2)</i>	Min/max range of decision variables
in	<i>integer(i8), optional :: mymaxn</i>	max no. of trials allowed before optimization is terminated DEFAULT: 1000_i8
in	<i>logical, optional :: mymaxit</i>	maximization (.true.) or minimization (.false.) of function DEFAULT: false
in	<i>integer(i4), optional :: mykstop</i>	number of shuffling loops in which the criterion value must change by given percentage before optimiz. is terminated DEFAULT: 10_i4
in	<i>real(dp), optional :: mypcento</i>	percentage by which the criterion value must change in given number of shuffling loops DEFAULT: 0.0001_dp
in	<i>real(dp), optional :: mypeps</i>	optimization is terminated if volume of complex has converged to given percentage of feasible space DEFAULT: 0.001_dp
in	<i>integer(i8), optional :: myseed</i>	initial random seed DEFAULT: get_timeseed

Parameters

in	<i>integer(i4), optional :: myngs</i>	number of complexes in the initial population DEFAULT: 2_i4
in	<i>integer(i4), optional :: mynpg</i>	number of points in each complex DEFAULT: 2*n+1
in	<i>integer(i4), optional :: mynps</i>	number of points in a sub-complex DEFAULT: n+1
in	<i>integer(i4), optional :: mynspl</i>	number of evolution steps allowed for each complex before complex shuffling DEFAULT: 2*n+1
in	<i>integer(i4), optional :: mymings</i>	minimum number of complexes required, if the number of complexes is allowed to reduce as the optimization proceeds DEFAULT: ngs = number of complexes in initial population
in	<i>integer(i4), optional :: myiniflg</i>	flag on whether to include the initial point in population 0, not included 1, included (DEFAULT)
in	<i>integer(i4), optional :: myprint</i>	flag for controlling print-out after each shuffling loop 0, print information on the best point of the population 1, print information on every point of the population 2, no printing (DEFAULT) 3, same as 0 but print progress '.' on every function call 4, same as 1 but print progress '.' on every function call
in	<i>logical, optional :: mymask(size(pini))</i>	parameter included in optimization (true) or discarded (false) DEFAULT: .true.
in	<i>real(dp), optional :: myalpha</i>	parameter for reflection of points in complex DEFAULT: 0.8_dp
in	<i>real(dp), optional :: mybeta</i>	parameter for contraction of points in complex DEFAULT: 0.45_dp
in	<i>character(len=*), optional :: tmp_file</i>	if given: write results after each evolution loop to temporal output file of that name of headlines: 7

format: '# nloop icall ngs1 bestf worstf ...
... gnrng (bestx(j),j=1,nn)'

Parameters

in	<i>character(len=*), optional :: popul_file</i>	if given: write whole population to file of that name of headlines: 1
----	---	---

format: #_evolution_loop, xf(i), (x(i,j),j=1,nn)
total number of lines written <= neval <= mymaxn

Parameters

in	<i>logical, optional :: popul_file_append</i>	if true, append to existing population file (default: false)
----	---	--

Parameters

in	<i>logical, optional :: parallel</i>	sce runs in parallel (true) or not (false) parallel sce should only be used if model/ objective is not parallel DEFAULT: .false.
in	<i>logical, optional :: restart</i>	if .true.: restart former sce run from restart_file
in	<i>character(len=*), optional :: restart_file</i>	file name for read/write of restart file (default: mo_sce.restart)
out	<i>real(dp), optional :: bestf</i>	the best value of the function.
out	<i>integer(i8), optional :: neval</i>	number of function evaluations needed.
out	<i>real(dp), optional, allocatable :: history(:)</i>	the history of best function values, history(neval)=bestf

Returns

`real(dp) :: bestx(size(pini))` — The parameters of the point which is estimated to minimize/maximize the function.

Note

Maximal number of parameters is 1000.

SCE is OpenMP enabled on the loop over the complexes.

OMP_NUM_THREADS > 1 does not give reproducible results even when seeded!

Author

Juliane Mai

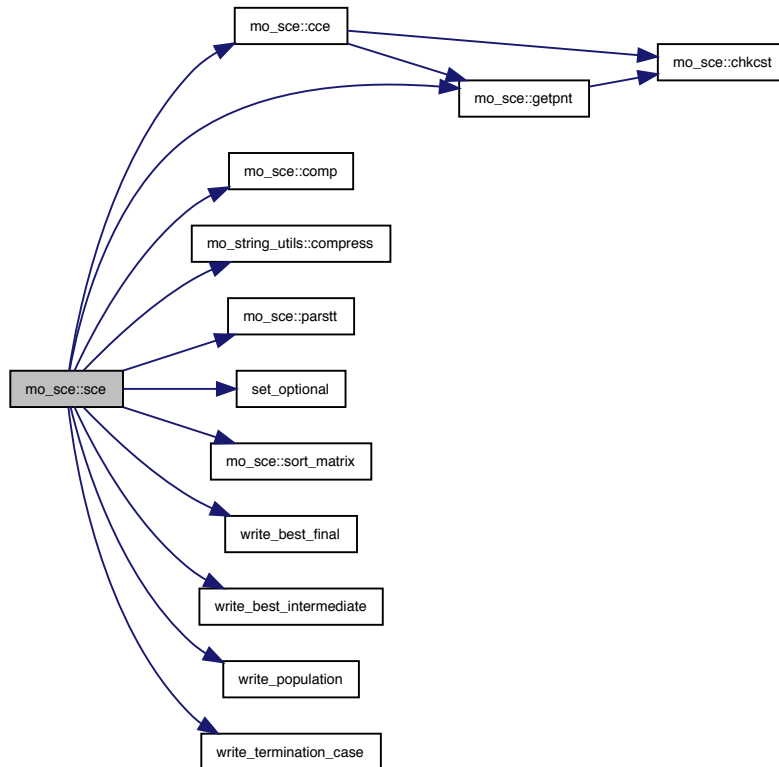
Date

Feb 2013

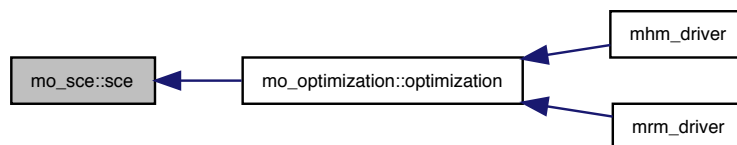
References `cce()`, `comp()`, `mo_string_utils::compress()`, `mo_kind::dp`, `getpnt()`, `mo_kind::i4`, `mo_kind::i8`, `mo_xor4096::n_save_state`, `parstt()`, `set_optional()`, `sort_matrix()`, `write_best_final()`, `write_best_intermediate()`, `write_population()`, and `write_termination_case()`.

Referenced by `mo_optimization::optimization()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.68.2.7 sort_matrix()

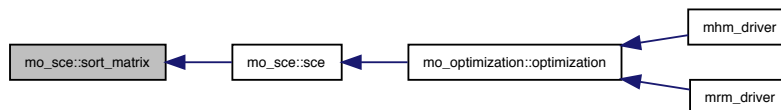
```

subroutine mo_sce::sort_matrix (
    real(dp), dimension(:, :), intent(inout) rb,
    real(dp), dimension(:), intent(inout) ra )
  
```

References `mo_kind::dp`, and `mo_kind::i4`.

Referenced by `sce()`.

Here is the caller graph for this function:



15.69 mo_set_netcdf_outputs Module Reference

Defines the structure of the netCDF to write the output in.

Functions/Subroutines

- subroutine, public [set_netcdf](#) (NoNetcdfVars, nrows, ncols)
Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

15.69.1 Detailed Description

Defines the structure of the netCDF to write the output in.

All output variables are initialized for the NetCDF.

Authors

Matthias Zink

Date

Apr 2013

15.69.2 Function/Subroutine Documentation

15.69.2.1 set_netcdf()

```

subroutine, public mo_set_netcdf_outputs::set_netcdf (
    integer(i4), intent(in) NoNetcdfVars,
    integer(i4), intent(in) nrows,
    integer(i4), intent(in) ncols )

```

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Author

Matthias Zink

Date

Apr 2013

References `mo_ncwrite::dnc`, `mo_ncwrite::ndims`, `mo_ncwrite::nvars`, and `mo_ncwrite::v`.

15.70 mo_snow_accum_melt Module Reference

Snow melting and accumulation.

Functions/Subroutines

- subroutine, public `snow_accum_melt` (`deg_day_incr`, `deg_day_max`, `deg_day_noprec`, `prec`, `temperature`, `temperature_thresh`, `thrfall`, `snow_pack`, `deg_day`, `melt`, `prec_effect`, `rain`, `snow`)

Snow melting and accumulation.

15.70.1 Detailed Description

Snow melting and accumulation.

This module calculates snow melting and accumulation.

Authors

Vladyslav Prykhodko

Date

Dec 2012

15.70.2 Function/Subroutine Documentation**15.70.2.1 snow_accum_melt()**

```
subroutine, public mo_snow_accum_melt::snow_accum_melt (
    real(dp), intent(in) deg_day_incr,
    real(dp), intent(in) deg_day_max,
    real(dp), intent(in) deg_day_noprec,
    real(dp), intent(in) prec,
    real(dp), intent(in) temperature,
    real(dp), intent(in) temperature_thresh,
    real(dp), intent(in) thrfall,
    real(dp), intent(inout) snow_pack,
    real(dp), intent(out) deg_day,
    real(dp), intent(out) melt,
    real(dp), intent(out) prec_effect,
    real(dp), intent(out) rain,
    real(dp), intent(out) snow )
```

Snow melting and accumulation.

Separates throughfall into rain and snow by comparing the temperature with the treshhold. by comparing the temperature with the treshhold. Calculates degree daily factor. Calculates snow melting rates. Calculates snow, rain and effective precipitation depth and snow pack.

Parameters

in	<i>real(dp) :: deg_day_incr</i>	Increase of the Degree-day factor per mm of increase in precipitation [s-1 degreeC-1]
in	<i>real(dp) :: deg_day_max</i>	Maximum Degree-day factor [m-1 degreeC-1]
in	<i>real(dp) :: deg_day_noprec</i>	Degree-day factor with no precipitation [m-1 degreeC-1]
in	<i>real(dp) :: prec</i>	Daily mean precipitation [m]
in	<i>real(dp) :: temperature</i>	Daily mean temperature [degreeC]
in	<i>real(dp) :: temperature_thresh</i>	Threshold temperature for snow/rain [degreeC]
in	<i>real(dp) :: thrfall</i>	Throughfall [m s-1]
in, out	<i>real(dp) :: snow_pack</i>	Snow pack [m]
out	<i>real(dp) :: deg_day</i>	Degree-day factor [m s-1 degreeC-1]
out	<i>real(dp) :: melt</i>	Melting snow depth [m s-1]
out	<i>real(dp) :: prec_effect</i>	Effective precipitation depth (snow melt + rain) [m]
out	<i>real(dp) :: rain</i>	Rain precipitation depth [m]
out	<i>real(dp) :: snow</i>	Snow precipitation depth [m]

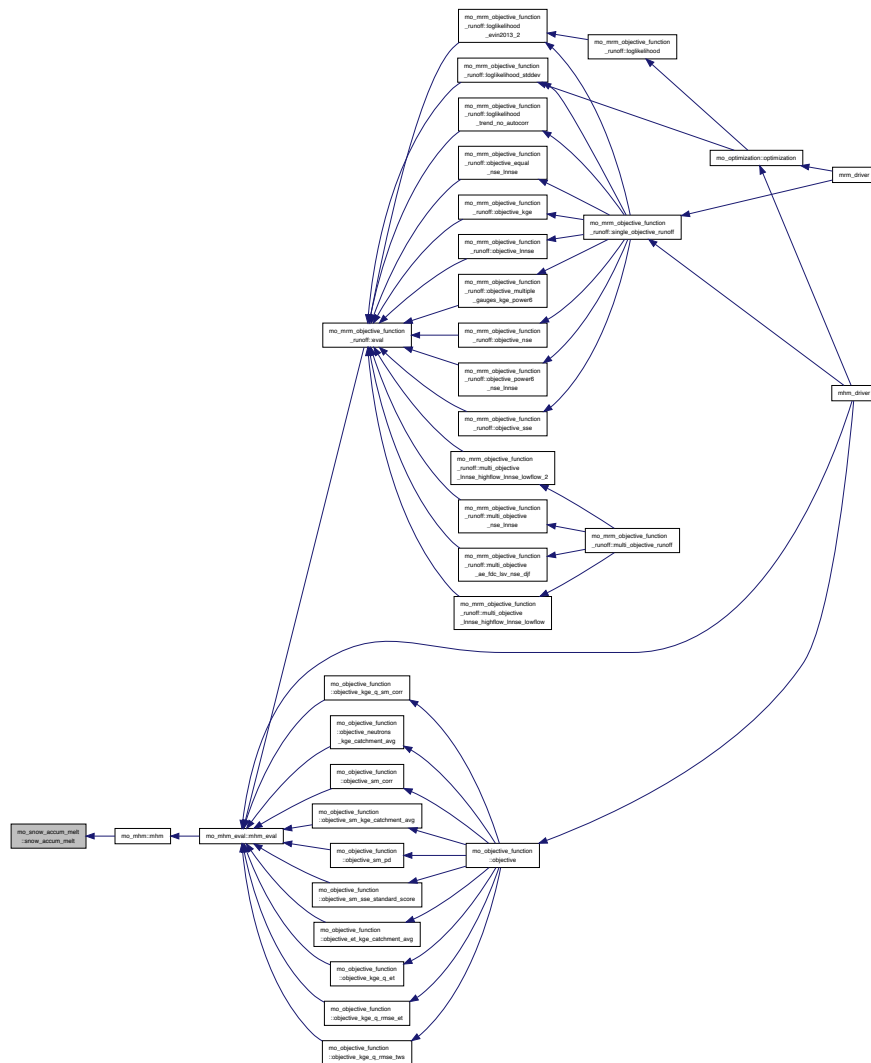
Author

Vladyslav Prykhodko

Date

Dec 2012

Referenced by mo_mhm::mhm().



- subroutine, public `read_soil_lut` (filename, iFlag_option, soilDB)
Reads the soil LUT file.
- subroutine, public `generate_soil_database` (soilDB, iFlag_option)
Generates soil database.

This module provides the routines for generating the soil database for mHM from an ASCII input file. One routine *read_soil_LUT* reads a soil LookUpTable, performs some consistency checks and returns an initial soil database. The second routine *generate_soil_database* calculates based on the initial one the proper soil database.

Authors

Juliane Mai

Date

Dec 2012

15.71.2 Function/Subroutine Documentation

15.71.2.1 generate_soil_database()

```
subroutine, public mo_soil_database::generate_soil_database (
    type(soiltype), intent(inout) soilDB,
    integer(i4), intent(in) iFlag_option )
```

Generates soil database.

Calculates the proper soil database using the initialized soil database from *read_soil_LUT*.

Parameters

in, out	<i>type(soilType) :: soilDB</i>	initialized/ proper soil database
in	<i>integer(i4) :: iFlag_option</i>	option for which kind of database to read

Author

Juliane Mai

Date

Dec 2012

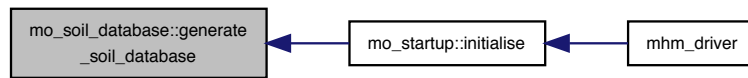
References *mo_global_variables::horizondepth_mhm*, *mo_message::message()*, *mo_mhm_constants::nodata_dp*, *mo_mhm_constants::nodata_i4*, *mo_global_variables::nsoilhorizons_mhm*, and *mo_global_variables::nsoiltypes*.

Referenced by *mo_startup::initialise()*.

Here is the call graph for this function:



Here is the caller graph for this function:



15.71.2.2 read_soil_lut()

```

subroutine, public mo_soil_database::read_soil_lut (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) iFlag_option,
    type(soiltype), intent(out) soilDB )
  
```

Reads the soil LUT file.

Reads the soil LookUpTable file and checks for consistency.

Parameters

in	<i>character(len=*) :: filename</i>	filename of the soil LUT
in	<i>integer(i4) :: iFlag_option</i>	option for which kind of database to read
out	<i>type(soilType) :: soilDB</i>	initialized soil database

Note

Soil LUT is an ASCII file.

Author

Juliane Mai

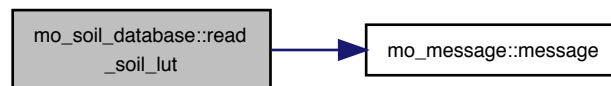
Date

Dec 2012

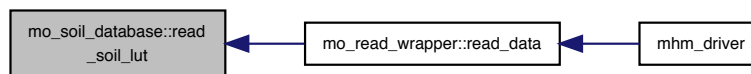
References mo_kind::dp, mo_constants::eps_dp, mo_global_variables::horizonddepth_mhm, mo_message::message(), mo_mhm_constants::nlcover_class, mo_mhm_constants::nodata_dp, mo_mhm_constants::nodata_i4, mo_global_variables::nsoilhorizons_mhm, mo_global_variables::nsoiltypes, mo_global_variables::tillagedepth, and mo_file::usoil_database.

Referenced by mo_read_wrapper::read_data().

Here is the call graph for this function:



Here is the caller graph for this function:



15.72 mo_soil_moisture Module Reference

Soil moisture of the different layers.

Functions/Subroutines

- subroutine, public [soil_moisture](#) (processCase, frac_sealed, water_thresh_sealed, pet, evap_coeff, soil_moist_sat, frac_roots, soil_moist_FC, wilting_point, soil_moist_exponen, jarvis_thresh_c1, aet_canopy, prec_effec, runoff_sealed, storage_sealed, infiltration, soil_moist, aet, aet_sealed)
Soil moisture in different soil horizons.
- elemental pure real(dp) function, private [feddes_et_reduction](#) (soil_moist, soil_moist_FC, wilting_point, frac_roots)
stress factor for reducing evapotranspiration based on actual soil moisture
- elemental pure real(dp) function, private [jarvis_et_reduction](#) (soil_moist, soil_moist_sat, wilting_point, frac_roots, jarvis_thresh_c1)
stress factor for reducing evapotranspiration based on actual soil moisture

15.72.1 Detailed Description

Soil moisture of the different layers.

Soil moisture in the different layers is calculated with infiltration as $(\theta/\theta_{sat})^\beta$

Then evapotranspiration is calculated from PET with a soil water stress factor f_{SM} either using

the Feddes equation - precessCase(1): $f_{SM} = \frac{\theta - \theta_{pwp}}{\theta_{fc} - \theta_{pwp}}$

or using the Jarvis equation - precessCase(1): $f_{SM} = \frac{1}{\theta_{stress-index-C1}} \frac{\theta - \theta_{pwp}}{\theta_{sat} - \theta_{pwp}}$.

Authors

Matthias Cuntz, Luis Samaniego

Date

Dec 2012

15.72.2 Function/Subroutine Documentation

15.72.2.1 feddes_et_reduction()

```
elemental pure real(dp) function, private mo_soil_moisture::feddes_et_reduction (
    real(dp), intent(in) soil_moist,
    real(dp), intent(in) soil_moist_FC,
    real(dp), intent(in) wilting_point,
    real(dp), intent(in) frac_roots ) [private]
```

stress factor for reducing evapotranspiration based on actual soil moisture

Potential evapotranspiration is reduced to 0 if SM is lower PWP. PET is equal fraction of roots if soil moisture is exceeding field capacity. If soil moisture is in between PWP and FC PET is reduced by fraction of roots times a stress factor.

The ET reduction factor f is estimated as

$$f = \begin{cases} f_{roots} & \text{if } \theta \geq \theta_{fc} \\ f_{roots} \cdot \frac{\theta - \theta_{pwp}}{\theta_{fc} - \theta_{pwp}} & \text{if } \theta < \theta_{fc} \\ 0 & \text{if } \theta < \theta_{pwp} \end{cases}$$

Parameters

in	<i>real(dp), intent(in) :: soil_moist</i>	Soil moisture of each horizon [mm]
in	<i>real(dp), intent(in) :: soil_moist_FC</i>	Soil moisture below which actual ET is reduced [mm]
in	<i>real(dp), intent(in) :: wilting_point</i>	Permanent wilting point
in	<i>real(dp), intent(in) :: frac_roots</i>	Fraction of Roots in soil horizon is reduced [mm]

Returns

real(dp) :: feddes_et_reduction; et reduction factor

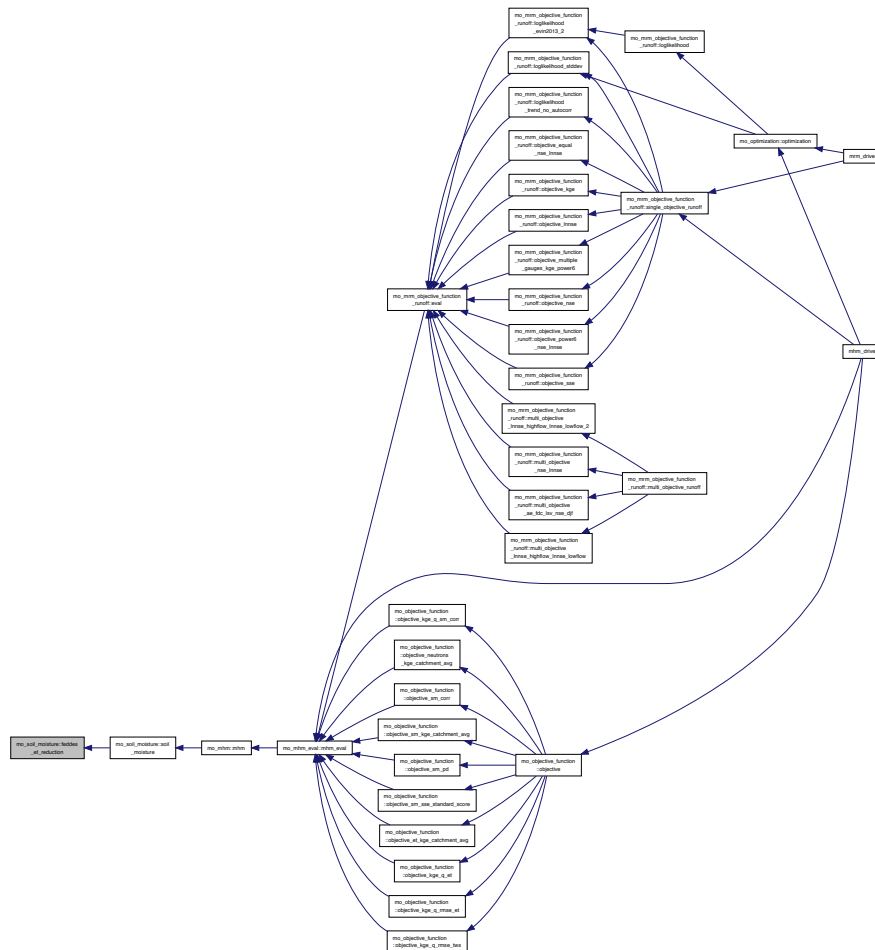
Note

Feddes, R.A., Kowalik, P., Kolinska-Malinka, K., Zaradny, H., 1976. Simulation of field water uptake by plants using a soil water dependent root extraction function. J. Hydrol. 31, 13–26. doi:10.1016/0022-1694(76)90017-2

Matthias Cuntz, Cueneyd Demirel, Matthias Zink

March 2017

Here is the caller graph for this function:



```

elemental pure real(dp) function, private mo_soil_moisture::jarvis_et_reduction (
    real(dp), intent(in)  soil_moist,
    real(dp), intent(in)  soil_moist_sat,
    real(dp), intent(in)  wilting_point,
    real(dp), intent(in)  frac_roots,
    real(dp), intent(in)  jarvis_thresh_c1 ) [private]

```

stress factor for reducing evapotranspiration based on actual soil moisture

The soil moisture stress factor is estimated based on the normalized soil water content. The normalized soil water content θ_{norm} is estimated as:

$$\theta_{norm} = \frac{\theta - \theta_{pwp}}{\theta_{sat} - \theta_{pwp}}$$

The ET reduction factor f is estimated as

$$f = \begin{cases} f_{roots} & \text{if } \theta_{norm} \geq \text{jarvis_sm_threshold_c1} \\ f_{roots} \frac{\theta_{norm}}{\text{jarvis_sm_threshold_c1}} & \text{if } \theta_{norm} < \text{jarvis_sm_threshold_c1} \end{cases}$$

Parameters

in	<i>real(dp), intent(in) :: soil_moist</i>	Soil moisture of each horizon [mm]
in	<i>real(dp), intent(in) :: soil_moist_sat</i>	saturated Soil moisture content [mm]
in	<i>real(dp), intent(in) :: wilting_point</i>	Permanent wilting point
in	<i>real(dp), intent(in) :: frac_roots</i>	Fraction of Roots in soil horizon is reduced [mm]
in	<i>real(dp), intent(in) :: jarvis_thresh_c1</i>	parameter C1 from Jarvis formulation

Returns

real(dp) :: jarvis_et_reduction; et reduction factor

Note

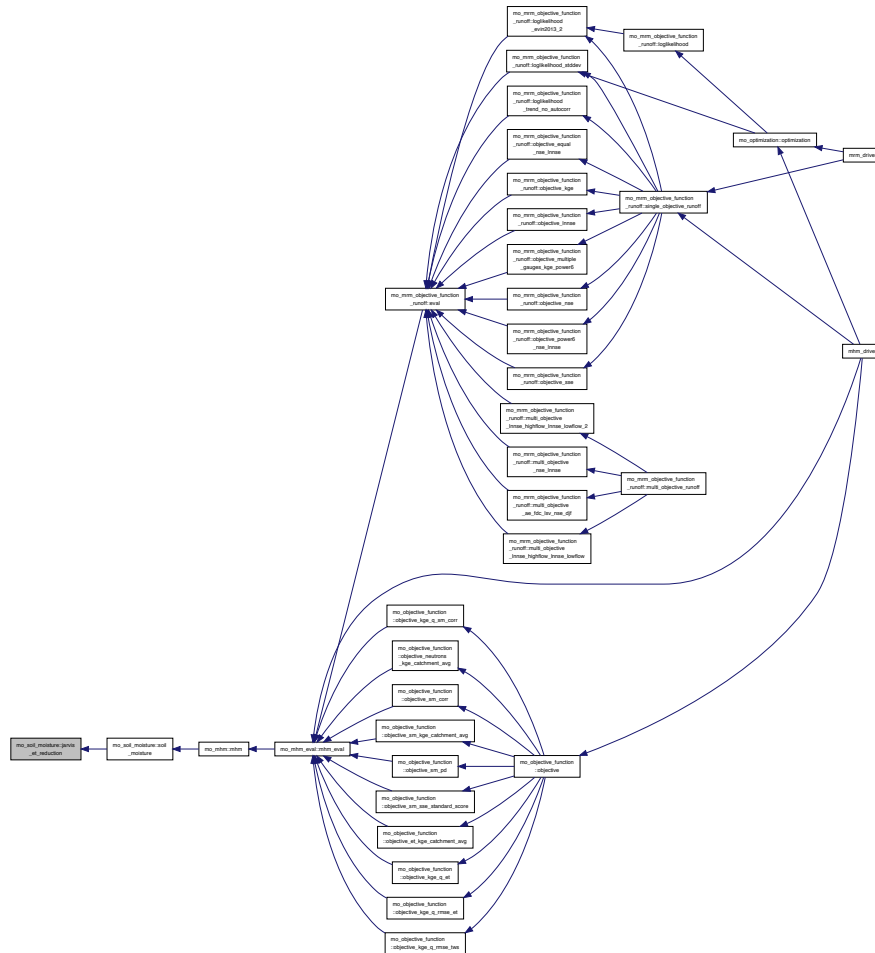
Jarvis, N.J., 1989. A simple empirical model of root water uptake. J. Hydrol. 107, 57–72. doi:10.1016/0022-1694(89)90050-4

Author

Cueneyd Demirel, Matthias Zink

March 2017

Here is the caller graph for this function:



```

subroutine, public mo_soil_moisture::soil_moisture (
    integer(i4), intent(in)  processCase,
    real(dp), intent(in)  frac_sealed,
    real(dp), intent(in)  water_thresh_sealed,
    real(dp), intent(in)  pet,
    real(dp), intent(in)  evap_coeff,
    real(dp), dimension(:), intent(in)  soil_moist_sat,
    real(dp), dimension(:), intent(in)  frac_roots,
    real(dp), dimension(:), intent(in)  soil_moist_FC,
    real(dp), dimension(:), intent(in)  wilting_point,
    real(dp), dimension(:), intent(in)  soil_moist_exponen,
    real(dp), intent(in)  jarvis_thresh_cl,
    real(dp), intent(in)  aet_canopy,

```

```

real(dp), intent(inout) prec_effec,
real(dp), intent(inout) runoff_sealed,
real(dp), intent(inout) storage_sealed,
real(dp), dimension(size(soil_moist_sat,1)), intent(inout) infiltration,
real(dp), dimension(size(soil_moist_sat,1)), intent(inout) soil_moist,
real(dp), dimension(size(soil_moist_sat,1)), intent(out) aet,
real(dp), intent(out) aet_sealed )

```

Soil moisture in different soil horizons.

Infiltration I from one layer $k - 1$ to the next k on pervious areas is calculated as (omit t)

$$I[k] = I[k - 1](\theta[k]/\theta_{sat}[k])^{\beta[k]}$$

Then soil moisture can be calculated as (omit k)

$$\theta[t] = \theta[t - 1] + I[t] - ET[t]$$

with ET (omit $[k,t]$) being

$$ET = f_{roots} \cdot f_{SM} \cdot PET$$

Parameters

in	<i>integer(i4), :: processCase</i>	1 - Feddes equation for PET reduction 2 - Jarvis equation for PET reduction 3 - Jarvis equation for PET reduction and FC dependency on root fraction coefficient
in	<i>real(dp) :: frac_sealed</i>	Fraction of sealed area
in	<i>real(dp) :: water_thresh_sealed</i>	Threshold water depth in impervious areas [mm/s]
in	<i>real(dp) :: pet</i>	Reference evapotranspiration [mm/s]
in	<i>real(dp) :: evap_coeff</i>	Evaporation coefficient for free-water surface of that current month
in	<i>real(dp), dimension(:) :: soil_moist_sat</i>	Saturation soil moisture for each horizon [mm]
in	<i>real(dp), dimension(:) :: frac_roots</i>	Fraction of Roots in soil horizon
in	<i>real(dp), dimension(:) :: soil_moist_FC</i>	Soil moisture below which actual ET is reduced [mm]
in	<i>real(dp), dimension(:) :: wilting_point</i>	Permanent wilting point for each horizon [mm]
in	<i>real(dp), dimension(:) :: soil_moist_exponen</i>	Exponential parameter to how non-linear is the soil water retention
in	<i>real(dp) :: jarvis_thresh_c1</i>	Jarvis critical value for normalized soil water content
in	<i>real(dp) :: aet_canopy</i>	Actual ET from canopy [mm/s]
in, out	<i>real(dp) :: prec_effec</i>	Effective precipitation (rain + snow melt) [mm]
in, out	<i>real(dp) :: runoff_sealed</i>	Direct runoff from impervious areas
in, out	<i>real(dp) :: storage_sealed</i>	Retention storage of impervious areas
in, out	<i>real(dp), dimension(:) :: infiltration</i>	Recharge, infiltration intensity or effective precipitation of each horizon [mm/s]
in, out	<i>real(dp), dimension(:) :: soil_moist</i>	Soil moisture of each horizon [mm]
out	<i>real(dp), dimension(:) :: aet</i>	actual ET [mm/s]
out	<i>real(dp) :: aet_sealed</i>	actual ET from free-water surfaces, i.e impervious cover [mm/s]

Author

Matthias Cuntz

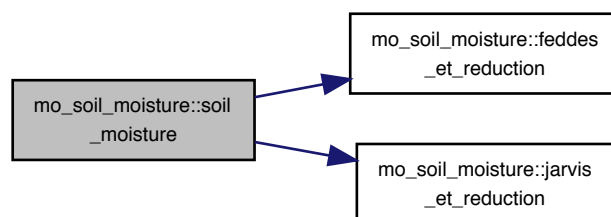
Date

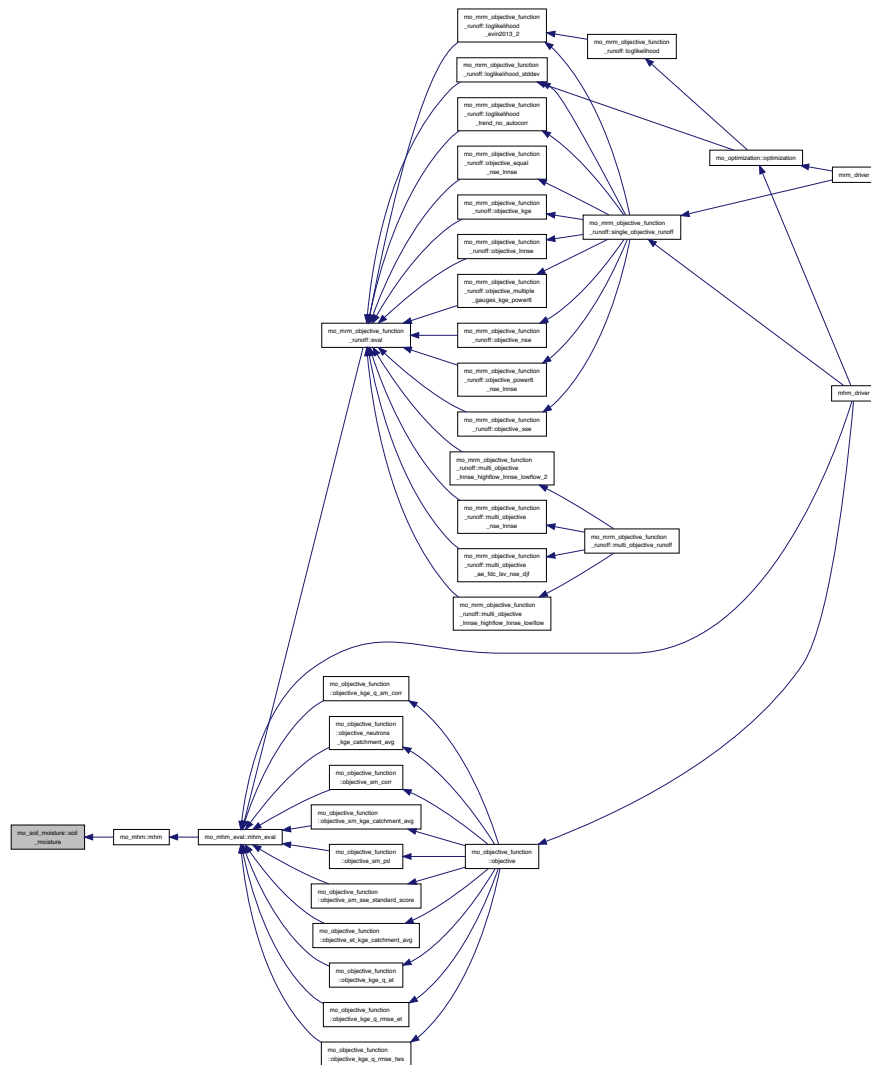
Dec 2012

References mo_constants::eps_dp, feddes_et_reduction(), and jarvis_et_reduction().

Referenced by mo_mhm::mhm().

Here is the call graph for this function:





15.73 mo_spatial_agg_disagg_forcing Module Reference

Spatial aggregation or disaggregation of meteorological input data.

Data Types

- interface `spatial_aggregation`
Spatial aggregation of meteorological variables.
- interface `spatial_disaggregation`
Spatial disaggregation of meteorological variables.

Functions/Subroutines

- subroutine `spatial_aggregation_3d` (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine `spatial_aggregation_4d` (data2, cellsize2, cellsize1, mask1, mask2, data1)

- subroutine [spatial_disaggregation_3d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [spatial_disaggregation_4d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)

15.73.1 Detailed Description

Spatial aggregation or disaggregation of meteorological input data.

This module contains two subroutines to upscale and downscale, respectively, the level-2 meteorological inputs to a required Level-1 hydrological spatial resolution.

Authors

Rohini Kumar

Date

Jan 2013

15.73.2 Function/Subroutine Documentation

15.73.2.1 [spatial_aggregation_3d\(\)](#)

```
subroutine mo_spatial_agg_disagg_forcing::spatial_aggregation_3d (
    real(dp), dimension(:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:), intent(out), allocatable data1 ) [private]
```

References `mo_mhm_constants::nodata_dp`.

15.73.2.2 [spatial_aggregation_4d\(\)](#)

```
subroutine mo_spatial_agg_disagg_forcing::spatial_aggregation_4d (
    real(dp), dimension(:,:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:,:), intent(out), allocatable data1 )
```

References `mo_mhm_constants::nodata_dp`.

15.73.2.3 [spatial_disaggregation_3d\(\)](#)

```
subroutine mo_spatial_agg_disagg_forcing::spatial_disaggregation_3d (
    real(dp), dimension(:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
```

```

logical, dimension(:,:), intent(in) mask1,
logical, dimension(:,:), intent(in) mask2,
real(dp), dimension(:,:,:), intent(out), allocatable data1 )

```

References `mo_mhm_constants::nodata_dp`.

15.73.2.4 spatial_disaggregation_4d()

```

subroutine mo_spatial_agg_disagg_forcing::spatial_disaggregation_4d (
    real(dp), dimension(:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:), intent(out), allocatable data1 )

```

References `mo_mhm_constants::nodata_dp`.

15.74 mo_spatialsimilarity Module Reference

Routines for bias insensitive comparison of spatial patterns.

Data Types

- interface [nndv](#)
Calculates the number of neighboring dominating values, a measure for spatial dissimilarity.
- interface [pd](#)
Calculates pattern dissimilarity (PD) measure.

Functions/Subroutines

- real(sp) function [nndv_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [nndv_dp](#) (mat1, mat2, mask, valid)
- real(sp) function [pd_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [pd_dp](#) (mat1, mat2, mask, valid)

15.74.1 Detailed Description

Routines for bias insensitive comparison of spatial patterns.

These routines are based on the idea that spatial similarity can be assessed by comparing the magnitude of neighboring pixels (e.g. is the neighboring pixel larger or smaller).

Author

Matthias Zink

Date

Mar 2013

15.74.2 Function/Subroutine Documentation

15.74.2.1 nndv_dp()

```
real(dp) function mo_spatialsimilarity::nndv_dp (
    real(dp), dimension(:,:), intent(in) mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

15.74.2.2 nndv_sp()

```
real(sp) function mo_spatialsimilarity::nndv_sp (
    real(sp), dimension(:,:), intent(in) mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

15.74.2.3 pd_dp()

```
real(dp) function mo_spatialsimilarity::pd_dp (
    real(dp), dimension(:,:), intent(in) mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

References mo_kind::dp.

15.74.2.4 pd_sp()

```
real(sp) function mo_spatialsimilarity::pd_sp (
    real(sp), dimension(:,:), intent(in) mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

References mo_kind::sp.

15.75 mo_standard_score Module Reference

Routines for calculating the normalization (anomaly)/standard score/z score and the deseasonalized (standard score on monthly basis) values of a time series.

Data Types

- interface [classified_standard_score](#)
Calculates the classified standard score (e.g. classes are months).

- interface [standard_score](#)

Calculates the standard score / normalization (anomaly) / z-score.

Functions/Subroutines

- real(sp) function, dimension(size(data, dim=1)) [standard_score_sp](#) (data, mask)
- real(dp) function, dimension(size(data, dim=1)) [standard_score_dp](#) (data, mask)
- real(sp) function, dimension(size(data, dim=1)) [classified_standard_score_sp](#) (data, classes, mask)
- real(dp) function, dimension(size(data, dim=1)) [classified_standard_score_dp](#) (data, classes, mask)

15.75.1 Detailed Description

Routines for calculating the normalization (anomaly)/standard score/z score and the deseasonalized (standard score on monthly basis) values of a time series.

In environmental research often the centralization and standardization are estimated for characterizing the dynamics of a signal.

Author

Matthias Zink

Date

May 2015

15.75.2 Function/Subroutine Documentation

15.75.2.1 [classified_standard_score_dp\(\)](#)

```
real(dp) function, dimension(size(data, dim=1)) mo_standard_score::classified_standard_score_dp (
    real(dp), dimension(:), intent(in) data,
    integer, dimension(:), intent(in) classes,
    logical, dimension(:), intent(in), optional mask )
```

15.75.2.2 [classified_standard_score_sp\(\)](#)

```
real(sp) function, dimension(size(data, dim=1)) mo_standard_score::classified_standard_score_sp (
    real(sp), dimension(:), intent(in) data,
    integer, dimension(:), intent(in) classes,
    logical, dimension(:), intent(in), optional mask )
```

15.75.2.3 [standard_score_dp\(\)](#)

```
real(dp) function, dimension(size(data, dim=1)) mo_standard_score::standard_score_dp (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(:), intent(in), optional mask )
```

15.75.2.4 standard_score_sp()

```
real(sp) function, dimension(size(data, dim=1)) mo_standard_score::standard_score_sp (
    real(sp), dimension(:), intent(in) data,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.76 mo_startup Module Reference

Startup procedures for mHM.

Functions/Subroutines

- subroutine, public [initialise](#) (iBasin)
Initialize main mHM variables.
- subroutine [constants_init](#) ()
Initialize mHM constants.
- subroutine [l0_check_input](#) (iBasin)
Check for errors in L0 input data.
- subroutine [l0_variable_init](#) (iBasin, soilId_isPresent)
level 0 variable initialization
- subroutine [l1_variable_init](#) (iBasin)
Level-1 variable initialization.
- subroutine [l2_variable_init](#) (iBasin)
Initialize Level-2 meteorological forcings data.

15.76.1 Detailed Description

Startup procedures for mHM.

This module initializes all variables required to run mHM. This module needs to be run only one time at the beginning of a simulation if re-starting files do not exist.

Author

Luis Samaniego, Rohini Kumar

Date

Dec 2012

15.76.2 Function/Subroutine Documentation

15.76.2.1 constants_init()

```
subroutine mo_startup::constants_init ( )
```

Initialize mHM constants.

transformation of time units & initialize constants

Author

Luis Samaniego

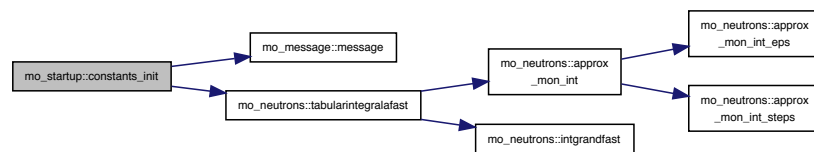
Date

Dec 2012

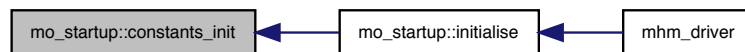
References `mo_global_variables::c2tstu`, `mo_message::message()`, `mo_global_variables::neutron_integral_afast`, `mo_global_variables::ntstepday`, `mo_common_variables::processmatrix`, `mo_neutrons::tabularintegralafast()`, and `mo_global_variables::timestep`.

Referenced by `initialise()`.

Here is the call graph for this function:



Here is the caller graph for this function:

**15.76.2.2 initialise()**

```

subroutine, public mo_startup::initialise (
    integer(i4), intent(in) iBasin )
  
```

Initialize main mHM variables.

Initialize main mHM variables for a given basin.

Calls the following procedures in this order:

- Constant initialization.
- Generate soil database.
- Checking inconsistencies input fields.

- Variable initialization at level-0.
- Variable initialization at level-1.
- Variable initialization at level-11.
- Space allocation of remaining variable/parameters.
Global variables will be used at this stage.

Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

Author

Luis Samaniego, Rohini Kumar

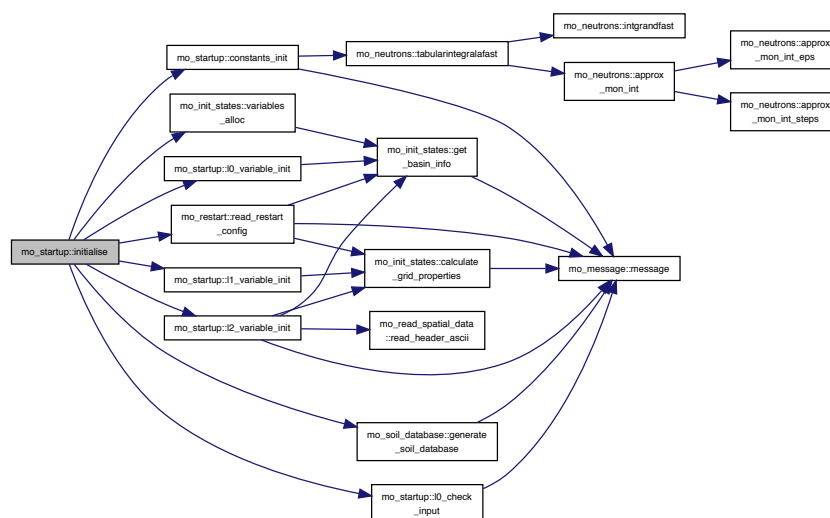
Date

Dec 2012

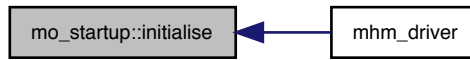
References constants_init(), mo_global_variables::dirrestartin, mo_soil_database::generate_soil_database(), mo_↵
_kind::i4, mo_global_variables::iflag_soildb, mo_global_variables::l0_basin, l0_check_input(), l0_variable_init(), l1_↵
_variable_init(), l2_variable_init(), mo_global_variables::perform_mpr, mo_global_variables::read_restart, mo_↵
restart::read_restart_config(), mo_global_variables::soildb, and mo_init_states::variables_alloc().

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.76.2.3 l0_check_input()

```

subroutine mo_startup::l0_check_input (
    integer(i4), intent(in) iBasin )
  
```

Check for errors in L0 input data.

Check for possible errors in input data (morphological and land cover) at level-0

Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

by default; when `iFlag_soilDB = 0`

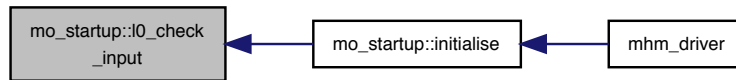
References `mo_global_variables::basin`, `mo_constants::eps_dp`, `mo_global_variables::l0_asp`, `mo_global_variables::l0_elev`, `mo_global_variables::l0_geounit`, `mo_global_variables::l0_slope`, `mo_global_variables::l0_soilid`, `mo_message::message()`, `mo_message::message_text`, `mo_mhm_constants::nodata_dp`, and `mo_mhm_constants::nodata_i4`.

Referenced by `initialise()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.76.2.4 l0_variable_init()

```

subroutine mo_startup::l0_variable_init (
    integer(i4), intent(in) iBasin,
    integer(i4), dimension(:), intent(inout), allocatable soilId_isPresent )
  
```

level 0 variable initialization

following tasks are performed for L0 data sets

- cell id & numbering
- storage of cell coordinates (row and column id)
- empirical dist. of terrain slope
- flag to determine the presence of a particular soil id in this configuration of the model run If a variable is added or removed here, then it also has to be added or removed in the subroutine config_variables_set in module [mo_restart](#) and in the subroutine set_config in module [mo_set_netcdf_restart](#)

Parameters

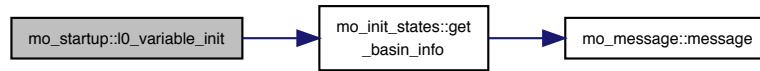
in	<i>integer(i4) :: iBasin</i>	basin id
in, out	<i>integer(i4), dimension(:) :: soilId_isPresent</i>	flag to indicate whether a given soil-id is present or not, DIMENSION [nSoilTypes]

by default; when iFlag_soilDB = 0

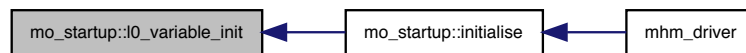
References [mo_init_states::get_basin_info\(\)](#), [mo_global_variables::l0_cellcoor](#), [mo_global_variables::l0_id](#), [mo_global_variables::l0_ncells](#), [mo_global_variables::l0_slope](#), [mo_global_variables::l0_slope_emp](#), [mo_global_variables::l0_soilid](#), [mo_global_variables::level0](#), [mo_mhm_constants::nodata_dp](#), [mo_mhm_constants::nodata_i4](#), [mo_constants::radius_earth_dp](#), and [mo_constants::twopi_dp](#).

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

Here is the call graph for this function:



Here is the caller graph for this function:



15.76.2.5 l1_variable_init()

```

subroutine mo_startup::l1_variable_init (
    integer(i4), intent(in) iBasin )
  
```

Level-1 variable initialization.

following tasks are performed for L1 datasets

- cell id & numbering
- mask creation
- storage of cell coordinates (row and column id)
- storage of four corner L0 coordinates If a variable is added or removed here, then it also has to be added or removed in the subroutine `config_variables_set` in module [mo_restart](#) and in the subroutine `set_config` in module `mo_set_netcdf_restart`

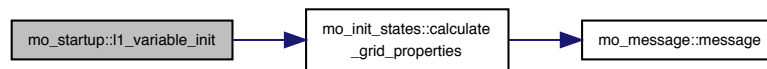
Parameters

in	<i>integer(i4) :: iBasin</i>	basin id
----	------------------------------	----------

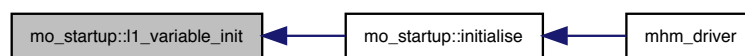
References `mo_global_variables::basin`, `mo_init_states::calculate_grid_properties()`, `mo_kind::i4`, `mo_global_variables::l0_areacell`, `mo_global_variables::l1_areacell`, `mo_global_variables::l1_cellcoor`, `mo_global_variables::l1_downbound_l0`, `mo_global_variables::l1_id`, `mo_global_variables::l1_leftbound_l0`, `mo_global_variables::l1_ncells`, `mo_global_variables::l1_ntcells_l0`, `mo_global_variables::l1_rightbound_l0`, `mo_global_variables::l1_upbound_l0`, `mo_global_variables::level0`, `mo_global_variables::level1`, `mo_global_variables::nbasins`, `mo_mhm_constants::nodata_dp`, and `mo_global_variables::resolutionhydrology`.

Referenced by `initialise()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.76.2.6 l2_variable_init()

```

subroutine mo_startup::l2_variable_init (
    integer(i4), intent(in) iBasin )
  
```

Initialize Level-2 meteorological forcings data.

following tasks are performed 1) cell id & numbering 2) mask creation 3) append variable of intrest to global ones

Parameters

in	<i>integer(i4) :: iBasin</i>	Basin Id
----	------------------------------	----------

Author

Rohini Kumar

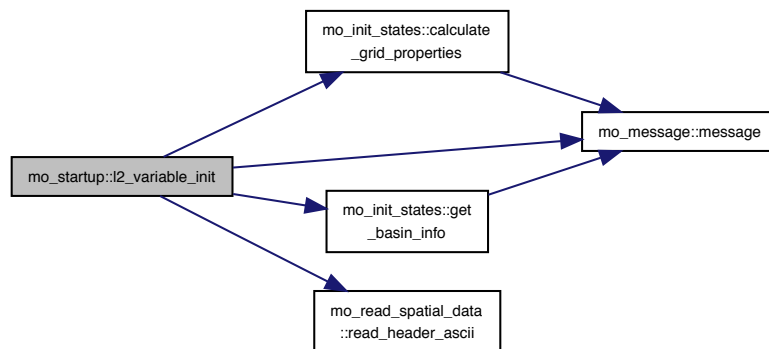
Date

Feb 2013

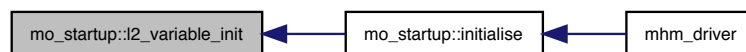
References `mo_global_variables::basin`, `mo_init_states::calculate_grid_properties()`, `mo_global_variables::dirprecipitation`, `mo_file::file_meteo_header`, `mo_init_states::get_basin_info()`, `mo_global_variables::level0`, `mo_global_variables::level2`, `mo_message::message()`, `mo_global_variables::nbasins`, `mo_mhm_constants::nodata_dp`, `mo_read_spatial_data::read_header_ascii()`, and `mo_file::umeteo_header`.

Referenced by `initialise()`.

Here is the call graph for this function:



Here is the caller graph for this function:



15.77 mo_string_utils Module Reference

String utilities.

Data Types

- interface [num2str](#)
Convert to string.
- interface [numarray2str](#)
Convert to string.

Functions/Subroutines

- `character(len(whitespaces))` function, public [compress](#) (`whiteSpaces`, `n`)

- subroutine, public `divide_string` (string, delim, strArr)
Divide string in substrings.
- logical function, public `equalstrings` (string1, string2)
- logical function, public `nonnull` (str)
Checks if string was already used.
- character(len=256) function, dimension(:), allocatable, public `splitstring` (string, delim)
- logical function, public `startswith` (string, start)
- character(len=len_trim(upper)) function, public `tolower` (upper)
Convert to lower case.
- character(len=len_trim(lower)) function, public `toupper` (lower)
- pure character(len=10) function `i42str` (nn, form)
- pure character(len=20) function `i82str` (nn, form)
- pure character(len=32) function `sp2str` (rr, form)
- pure character(len=32) function `dp2str` (rr, form)
- pure character(len=10) function `log2str` (ll, form)
- character(len=size(arr)) function `i4array2str` (arr)
- integer(i4) function, dimension(:), allocatable, public `str2num` (string)

Variables

- character(len= *), parameter, public `separator` = repeat('-', 70)

15.77.1 Detailed Description

String utilities.

This module provides string conversion and checking utilities.

Authors

Matthias Cuntz, Matthias Zink, Giovanni Dalmaso, David Schaefer

Date

Dec 2011

15.77.2 Function/Subroutine Documentation

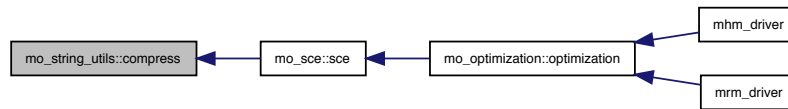
15.77.2.1 compress()

```
character(len(whitespaces)) function, public mo_string_utils::compress (
    character(len=*), intent(in) whiteSpaces,
    integer(i4), intent(out), optional n )
```

References `mo_kind::i4`.

Referenced by `mo_sce::sce()`.

Here is the caller graph for this function:



15.77.2.2 divide_string()

```

subroutine, public mo_string_utils::divide_string (
    character(len=*), intent(in) string,
    character(len=*), intent(in) delim,
    character(len=*), dimension(:), intent(out), allocatable strArr )
  
```

Divide string in substrings.

Divides a string in several substrings (array of strings) with the help of a user specified delimiter.

Parameters

in	<i>CHARACTER(len=*)</i> , <i>INTENT(IN) :: string</i>	- string to be divided
in	<i>CHARACTER(len=*)</i> , <i>INTENT(IN) :: delim</i>	- delimiter specifying places for division
out	<i>CHARACTER(len=*)</i> , <i>DIMENSION(:)</i> , <i>ALLOCATABLE</i> , <i>INTENT(OUT) :: strArr</i>	Array of substrings, has to be allocateable and is handed to the routine unallocated

Author

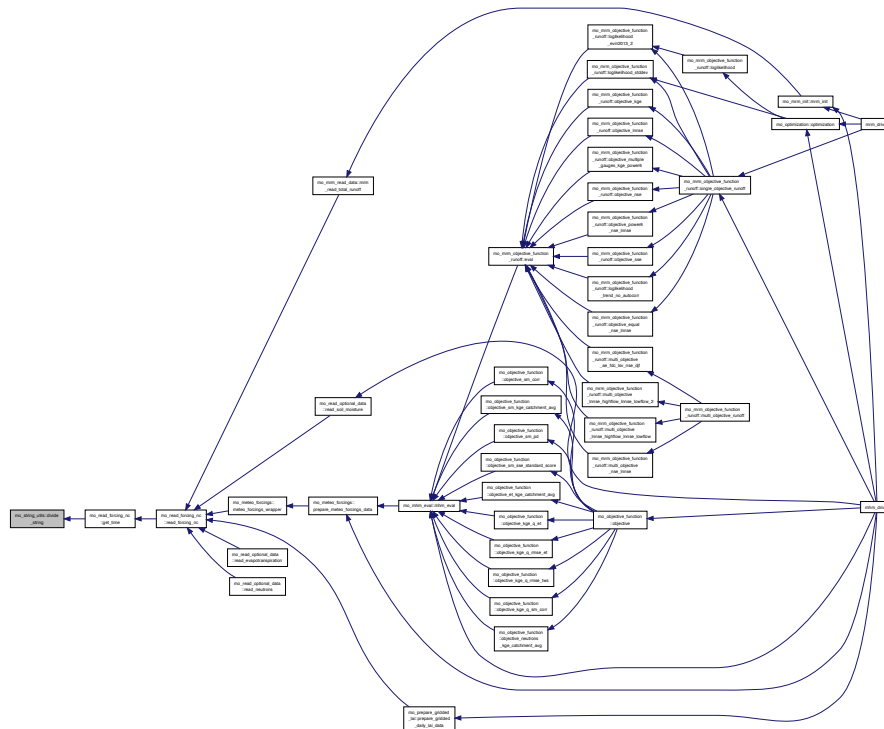
Matthias Zink

Date

Oct 2012

Referenced by mo_read_forcing_nc::get_time().

Here is the caller graph for this function:



15.77.2.3 dp2str()

```
pure character(len=32) function mo_string_utils::dp2str (
    real(dp), intent(in) rr,
    character(len=*), intent(in), optional form ) [private]
```

15.77.2.4 equalstrings()

```
logical function, public mo_string_utils::equalstrings (
    character(len=*), intent(in) string1,
    character(len=*), intent(in) string2 )
```

References str2num().

Here is the call graph for this function:



15.77.2.5 i42str()

```
pure character(len=10) function mo_string_utils::i42str (
    integer(i4), intent(in) nn,
    character(len=*), intent(in), optional form ) [private]
```

15.77.2.6 i4array2str()

```
character(len=size(arr)) function mo_string_utils::i4array2str (
    integer(i4), dimension(:), intent(in) arr ) [private]
```

15.77.2.7 i82str()

```
pure character(len=20) function mo_string_utils::i82str (
    integer(i8), intent(in) nn,
    character(len=*), intent(in), optional form ) [private]
```

15.77.2.8 log2str()

```
pure character(len=10) function mo_string_utils::log2str (
    logical, intent(in) ll,
    character(len=*), intent(in), optional form ) [private]
```

15.77.2.9 nonull()

```
logical function, public mo_string_utils::nonull (
    character(len=*), intent(in) str )
```

Checks if string was already used.

Checks if string was already used, i.e. does not contain NULL character anymore.

Parameters

in	<i>character(len=*) :: str</i>	String
----	--------------------------------	--------

Returns

logical :: used — .true.: string was already set; .false.: string still in initialised state

Author

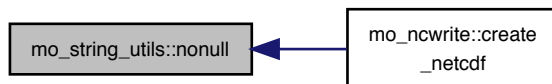
Matthias Cuntz

Date

Jan 2012

Referenced by mo_ncwrite::create_netcdf().

Here is the caller graph for this function:



15.77.2.10 sp2str()

```

pure character(len=32) function mo_string_utils::sp2str (
    real(sp), intent(in) rr,
    character(len=*), intent(in), optional form ) [private]
  
```

15.77.2.11 splitstring()

```

character(len=256) function, dimension(:), allocatable, public mo_string_utils::splitstring (
    character(len=*), intent(in) string,
    character(len=*), intent(in) delim )
  
```

References str2num().

Here is the call graph for this function:



15.77.2.12 startswith()

```

logical function, public mo_string_utils::startswith (
    character(len=*), intent(in) string,
    character(len=*), intent(in) start )
  
```

References str2num().

Here is the call graph for this function:

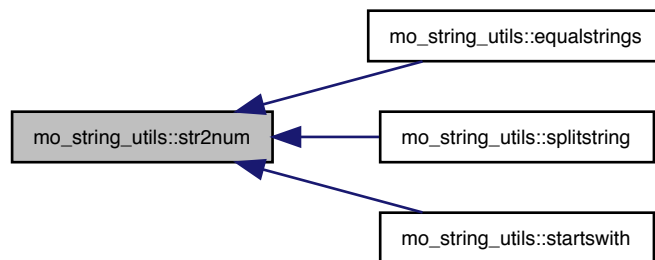


15.77.2.13 str2num()

```
integer(i4) function, dimension(:), allocatable, public mo_string_utils::str2num (
    character(len=*), intent(in) string )
```

Referenced by equalstrings(), splitstring(), and startswith().

Here is the caller graph for this function:



15.77.2.14 tolower()

```
character(len=len_trim(upper)) function, public mo_string_utils::tolower (
    character(len=*), intent(in) upper )
```

Convert to lower case.

Convert all upper case letters in string to lower case letters.

Parameters

in	<i>character(len=*) :: upper</i>	String
----	----------------------------------	--------

Returns

`character(len=len_trim(upper)) :: low` — String where all uppercase in input is converted to lowercase

Author

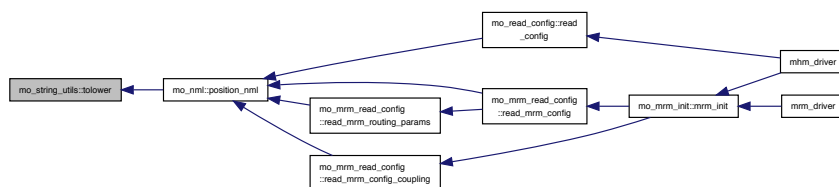
Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

Date _____

Dec 2011

Referenced by `mo_nml::position_nml()`.

Here is the caller graph for this function:

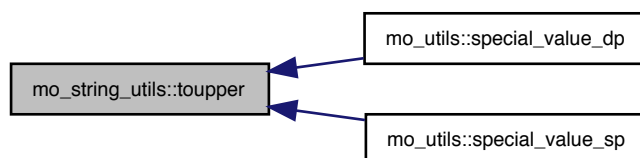


15.77.2.15 toupper()

```
character(len=len_trim(lower)) function, public mo_string_utils::toupper (
    character(len=*), intent(in) lower )
```

Referenced by `mo_utils::special_value_dp()`, and `mo_utils::special_value_sp()`.

Here is the caller graph for this function:



15.77.3 Variable Documentation

15.77.3.1 separator

```
character(len=*), parameter, public mo_string_utils::separator = repeat('-', 70)
```

Referenced by `mo_finish::finish()`, `mhm_driver()`, and `mo_mrm_init::print_startup_message()`.

15.78 mo_template Module Reference

Template for future module developments.

Data Types

- interface [mean](#)
The average.

Functions/Subroutines

- elemental pure real(dp) function, public [circum](#) (radius)
Circumference of a circle.
- real(dp) function [mean_dp](#) (dat, mask)
- real(sp) function [mean_sp](#) (dat, mask)

Variables

- real(dp), parameter, public [pi_dp](#) = 3.141592653589793238462643383279502884197_dp
Constant Pi in double precision.
- real(sp), parameter, public [pi_sp](#) = 3.141592653589793238462643383279502884197_sp
Constant Pi in single precision.
- integer(i4), parameter [itest](#) = 1

15.78.1 Detailed Description

Template for future module developments.

This module serves as a template for future model developments. It shows the module structure, the coding style, and documentation.

Please read the [Coding and documentation style](#) guide.

Authors

Matthias Cuntz, Christoph Schneider

Date

Dec 2012

15.78.2 Function/Subroutine Documentation

15.78.2.1 circum()

```
elemental pure real(dp) function, public mo_template::circum (
    real(dp), intent(in) radius )
```

Circumference of a circle.

Calculates the circumference of a circle

$$c = 2\pi r$$

Parameters

in	real(dp) :: radius	Radius
----	--------------------	--------

Returns

real(dp) :: circum — circumference of circle.

Author

Matthias Cuntz

Date

Dec 2012

References pi_dp.

15.78.2.2 mean_dp()

```
real(dp) function mo_template::mean_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.78.2.3 mean_sp()

```
real(sp) function mo_template::mean_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask ) [private]
```

15.78.3 Variable Documentation**15.78.3.1 itest**

```
integer(i4), parameter mo_template::itest =1 [private]
```

15.78.3.2 pi_dp

```
real(dp), parameter, public mo_template::pi_dp = 3.141592653589793238462643383279502884197_dp
```

Constant Pi in double precision.

Referenced by `circum()`.

15.78.3.3 pi_sp

```
real(sp), parameter, public mo_template::pi_sp = 3.141592653589793238462643383279502884197_sp
```

Constant Pi in single precision.

15.79 mo_temporal_aggregation Module Reference

Temporal aggregation for time series (averaging)

Data Types

- interface [day2mon_average](#)
Day-to-month average ([day2mon_average](#))
- interface [hour2day_average](#)
Hour-to-day average ([hour2day_average](#))

Functions/Subroutines

- subroutine [day2mon_average_dp](#) (daily_data, yearS, monthS, dayS, mon_avg, misval, rm_misval)
- subroutine [hour2day_average_dp](#) (hourly_data, yearS, monthS, dayS, hourS, day_avg, misval, rm_misval)

15.79.1 Detailed Description

Temporal aggregation for time series (averaging)

This module does temporal aggregation (averaging) of time series

Authors

Oldrich Rakovec, Rohini Kumar

Date

October 2015

15.79.2 Function/Subroutine Documentation

15.79.2.1 day2mon_average_dp()

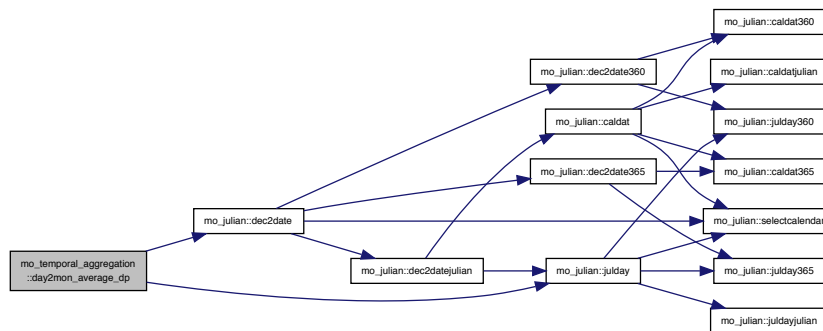
```

subroutine mo_temporal_aggregation::day2mon_average_dp (
    real(dp), dimension(:), intent(in) daily_data,
    integer(i4), intent(in) yearS,
    integer(i4), intent(in) monthS,
    integer(i4), intent(in) dayS,
    real(dp), dimension(:), intent(inout), allocatable mon_avg,
    real(dp), intent(in), optional misval,
    logical, intent(in), optional rm_misval ) [private]

```

References mo_julian::dec2date(), mo_constants::eps_dp, and mo_julian::julday().

Here is the call graph for this function:



15.79.2.2 hour2day_average_dp()

```

subroutine mo_temporal_aggregation::hour2day_average_dp (
    real(dp), dimension(:), intent(in) hourly_data,
    integer(i4), intent(in) yearS,
    integer(i4), intent(in) monthS,
    integer(i4), intent(in) dayS,
    integer(i4), intent(in) hourS,
    real(dp), dimension(:), intent(inout), allocatable day_avg,
    real(dp), intent(in), optional misval,
    logical, intent(in), optional rm_misval ) [private]

```

15.80 mo_temporal_disagg_forcing Module Reference

Temporal disaggregation of daily input values.

Functions/Subroutines

- elemental pure subroutine, public [temporal_disagg_forcing](#) (isday, ntimesteps_day, prec_day, pet_day, temp↔_day, fday_prec, fday_pet, fday_temp, fnight_prec, fnight_pet, fnight_temp, temp_weights, pet_weights, pre↔_weights, read_meteo_weights, prec, pet, temp)

Temporally distribute daily mean forcings onto time step.

15.80.1 Detailed Description

Temporal disaggregation of daily input values.

Calculate actual values for precipitation, PET and temperature from daily mean inputs

Note

There is not PET correction for aspect in this routine. Use `pet * fasp` before or after the routine.

Authors

Matthias Cuntz

Date

Dec 2012

15.80.2 Function/Subroutine Documentation

15.80.2.1 `temporal_disagg_forcing()`

```

elemental pure subroutine, public mo_temporal_disagg_forcing::temporal_disagg_forcing (
    logical, intent(in) isday,
    real(dp), intent(in) ntimesteps_day,
    real(dp), intent(in) prec_day,
    real(dp), intent(in) pet_day,
    real(dp), intent(in) temp_day,
    real(dp), intent(in) fday_prec,
    real(dp), intent(in) fday_pet,
    real(dp), intent(in) fday_temp,
    real(dp), intent(in) fnight_prec,
    real(dp), intent(in) fnight_pet,
    real(dp), intent(in) fnight_temp,
    real(dp), intent(in) temp_weights,
    real(dp), intent(in) pet_weights,
    real(dp), intent(in) pre_weights,
    logical, intent(in) read_meteo_weights,
    real(dp), intent(out) prec,
    real(dp), intent(out) pet,
    real(dp), intent(out) temp )

```

Temporally distribute daily mean forcings onto time step.

Calculates actual precipitation, PET and temperature from daily mean inputs.

Precipitation and PET are distributed with predefined factors onto the day.

Temperature gets a predefined amplitude added on day and subtracted at night.

Alternatively, weights for each hour and month can be given and disaggregation is using these as factors for PET and temperature. Precipitation is distributed uniformly.

Parameters

in	<i>logical :: isday</i>	is day or night
in	<i>real(dp) :: ntimesteps_day</i>	# of time steps per day
in	<i>real(dp) :: prec_day</i>	Daily mean precipitation [mm/s]
in	<i>real(dp) :: pet_day</i>	Daily mean ET [mm/s]

Parameters

in	<i>real(dp) :: temp_day</i>	Daily mean air temperature [K]
in	<i>real(dp) :: fday_prec</i>	Daytime fraction of precipitation
in	<i>real(dp) :: fday_pet</i>	Daytime fraction of PET
in	<i>real(dp) :: fday_temp</i>	Daytime air temparture increase
in	<i>real(dp) :: fnight_prec</i>	Daytime fraction of precipitation
in	<i>real(dp) :: fnight_pet</i>	Daytime fraction of PET
in	<i>real(dp) :: fnight_temp</i>	Daytime air temparture increase
out	<i>real(dp) :: prec</i>	Actual precipitation [mm/s]
out	<i>real(dp) :: pet</i>	Reference ET [mm/s]
out	<i>real(dp) :: temp</i>	Air temperature [K]

Note

There is no PET correction for aspect in this routine. Use `pet * fasp` before or after the routine.
Example:

```
call temporal_disagg_forcing((tt <= 6) .or. ((tt>=19) .and. (tt<=24)), &
    real(NTSTEPDAY, dp), prec_day(k), pet_day(k), temp_day(k), &
    fpre(month,2), fpet(month,2), ftem(month,2), &
    fpre(month,1), fpet(month,1), ftem(month,1), &
    tavg_weight(k,month,hour), pet_weights(k,month,hour), read_meteo_weights, &
    prec(k), pet(k), temp(k))
pet(k) = pet(k) * fasp(k)
```

Author

Matthias Cuntz

Date

Dec 2012

Referenced by `mo_mhm::mhm()`.

[illegible]

Timing routines.

- subroutine, public `timer_check` (timer)
Check a timer.
- subroutine, public `timer_clear` (timer)
Reset a timer.
- real(sp) function, public `timer_get` (timer)
Return a timer.
- subroutine, public `timer_print` (timer)
Print a timer.
- subroutine, public `timer_start` (timer)
Start a timer.

- subroutine, public `timer_stop` (timer)
Stop a timer.
- subroutine, public `timers_init`
Initialise timer module.

Variables

- integer(i4), parameter, public `max_timers` = 99
max number of timers allowed
- integer(i4), save, public `cycles_max`
max value of clock allowed by system
- real(sp), save, public `clock_rate`
clock_rate in seconds for each cycle
- integer(i4), dimension(`max_timers`), save, public `cycles1`
cycle number at start for each timer
- integer(i4), dimension(`max_timers`), save, public `cycles2`
cycle number at stop for each timer
- real(sp), dimension(`max_timers`), save, public `cputime`
accumulated cpu time in each timer
- character(len=8), dimension(`max_timers`), save, public `status`
timer status string

15.81.1 Detailed Description

Timing routines.

This module uses F90 cpu time routines to allowing setting of multiple CPU timers.

Authors

Matthias Cuntz - from `timers.f` (c) the Regents of the University of Californi

Date

Dec 2012

15.81.2 Function/Subroutine Documentation

15.81.2.1 `timer_check()`

```
subroutine, public mo_timer::timer_check (
    integer(i4), intent(in) timer )
```

Check a timer.

This routine checks a given timer. This is primarily used to periodically accumulate time in the timer to prevent timer cycles from wrapping around `max_cycles`.

Parameters

in	<code>integer(i4) :: timer</code>	timer number
----	-----------------------------------	--------------

Author

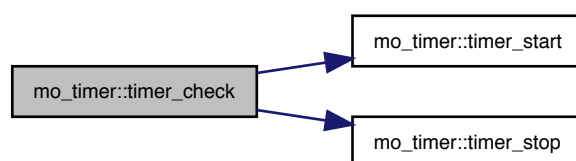
Matthias Cuntz

Date

Aug 2012

References `status`, `timer_start()`, and `timer_stop()`.

Here is the call graph for this function:

**15.81.2.2 timer_clear()**

```
subroutine, public mo_timer::timer_clear (
    integer(i4), intent(in), optional timer )
```

Reset a timer.

This routine resets a given timer or all timers to 0.

Parameters

in	<i>integer(i4), optional :: timer</i>	timer number if given. If missing, all timers will be reset.
----	---------------------------------------	---

Author

Matthias Cuntz

Date

Aug 2012

References `cputime`.

15.81.2.3 timer_get()

```
real(sp) function, public mo_timer::timer_get (
    integer(i4), intent(in) timer )
```

Return a timer.

This routine returns the result of a given timer. This can be called instead of timer_print so that the calling routine can print it in desired format.

Parameters

in	<i>integer(i4) :: timer</i>	timer number
----	-----------------------------	--------------

Author

Matthias Cuntz

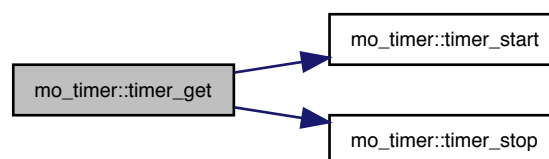
Date

Aug 2012

References cputime, status, timer_start(), and timer_stop().

Referenced by mhm_driver(), mrm_driver(), mo_optimization::optimization(), mo_meteo_forcings::prepare_meteo_forcings_data(), mo_read_optional_data::read_evapotranspiration(), mo_read_optional_data::read_neutrons(), and mo_read_optional_data::read_soil_moisture().

Here is the call graph for this function:



The diagram illustrates the dependency structure of the 'ms' package. It features a central 'ms_driver' module at the top, which depends on a wide range of other modules. Below it, 'ms_optimization' and 'ms_objective_function' are key components. The middle section contains numerous 'ms_obj' modules, each representing a specific objective function. The bottom section contains 'ms_util' modules, which are utility functions. The graph shows a dense network of dependencies, with many modules having multiple incoming and outgoing arrows. The nodes are labeled with their full names, including the package name 'ms' and the module name. The edges represent the dependencies between these modules.

```
subroutine, public mo_timer::timer_print (
    integer(i4), intent(in) timer )
```

This routine prints the accumulated cpu time in given timer.

in	<i>integer(i4) :: timer</i>	timer number
----	-----------------------------	--------------

Author

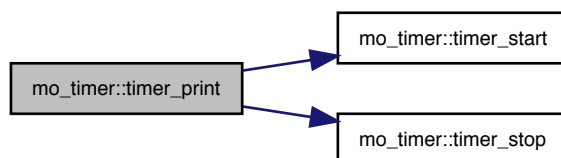
Matthias Cuntz

Date

Aug 2012

References `cputime`, `status`, `timer_start()`, and `timer_stop()`.

Here is the call graph for this function:

**15.81.2.5 timer_start()**

```
subroutine, public mo_timer::timer_start (  
    integer(i4), intent(in) timer )
```

Start a timer.

This routine starts a given timer.

Parameters

in	<i>integer(i4) :: timer</i>	timer number
----	-----------------------------	--------------

Author

Matthias Cuntz

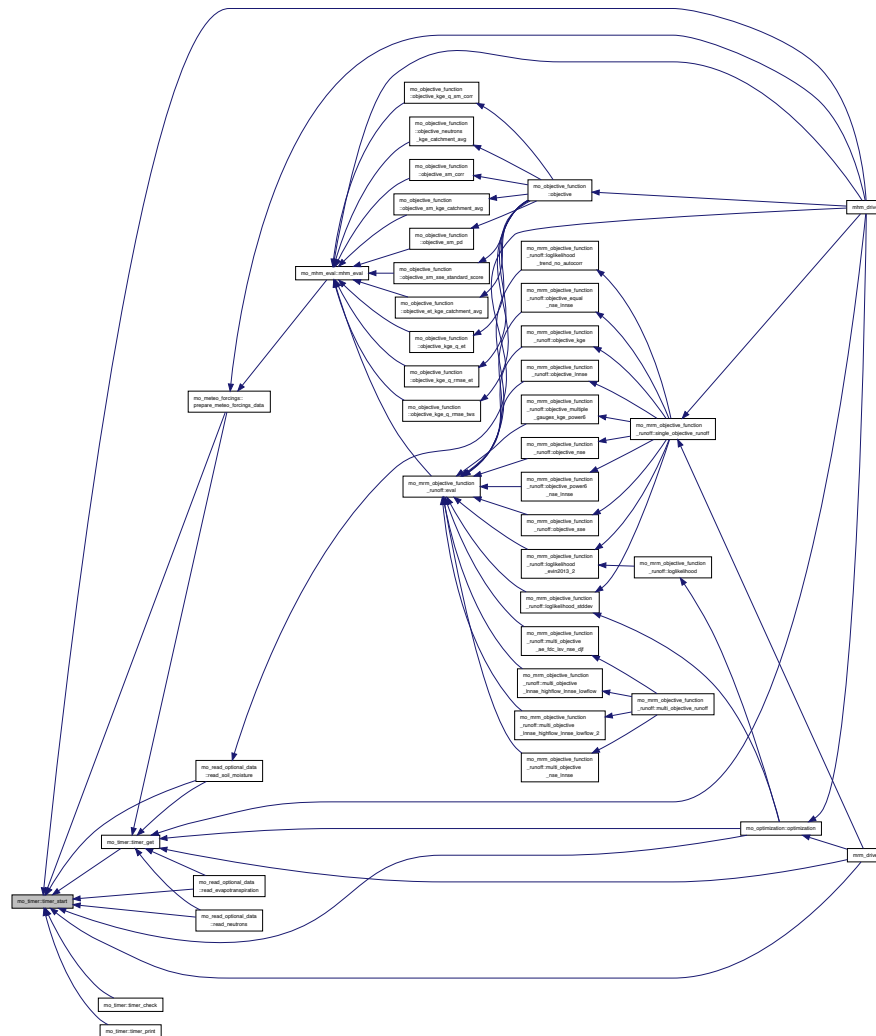
Date

Aug 2012

References cycles1, and status.

Referenced by mhm_driver(), mrm_driver(), mo_optimization::optimization(), mo_meteo_forcings::prepare_meteo_forcings_data(), mo_read_optional_data::read_evapotranspiration(), mo_read_optional_data::read_neutrons(), mo_read_optional_data::read_soil_moisture(), timer_check(), timer_get(), and timer_print().

Here is the caller graph for this function:



15.81.2.6 timer_stop()

```
subroutine, public mo_timer::timer_stop (
    integer(i4), intent(in) timer )
```

Stop a timer.

This routine stops a given timer.

Parameters

in	<code>integer(i4) :: timer</code>	timer number
----	-----------------------------------	--------------

Author

Matthias Cuntz

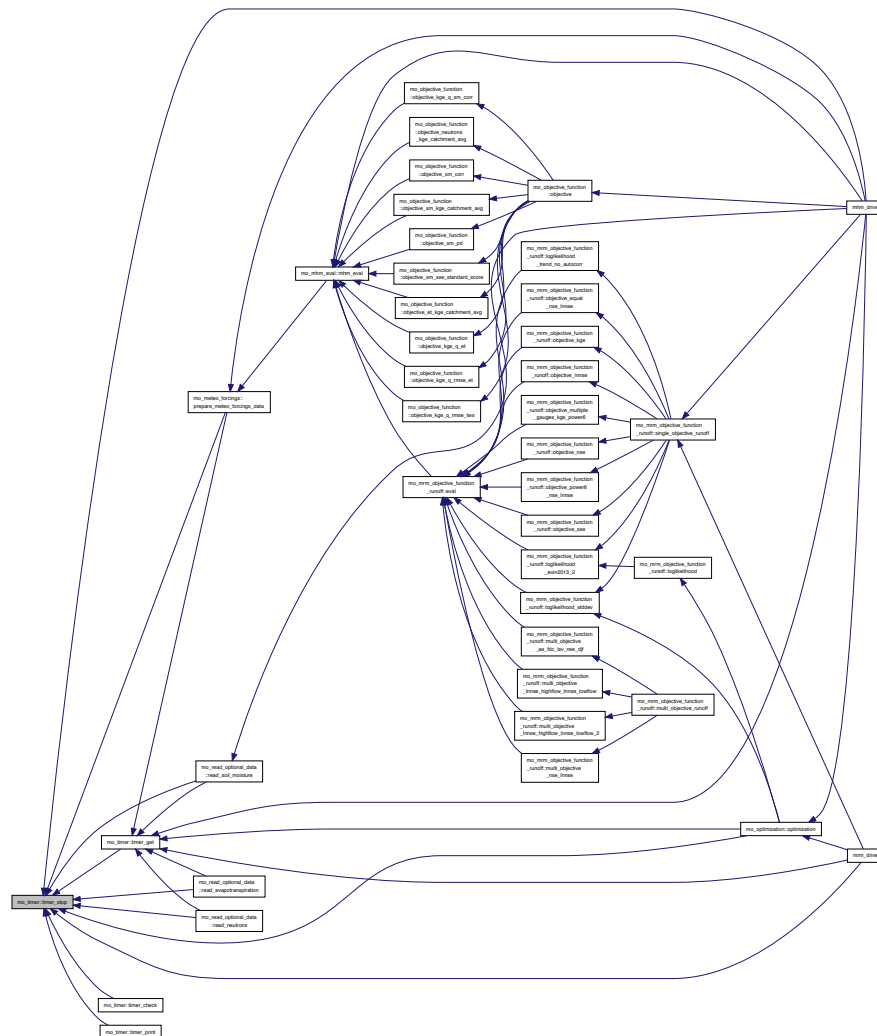
Date

Aug 2012

References clock_rate, cputime, cycles1, cycles2, cycles_max, and status.

Referenced by mhm_driver(), mrm_driver(), mo_optimization::optimization(), mo_meteo_forcings::prepare_meteo_forcings_data(), mo_read_optional_data::read_evapotranspiration(), mo_read_optional_data::read_neutrons(), mo_read_optional_data::read_soil_moisture(), timer_check(), timer_get(), and timer_print().

Here is the caller graph for this function:



15.81.2.7 timers_init()

```
subroutine, public mo_timer::timers_init ( )
```

Initialise timer module.

This routine initializes some machine parameters necessary for computing cpu time from F90 intrinsics.

Author

Matthias Cuntz

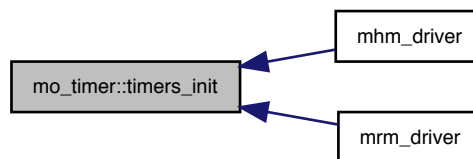
Date

Aug 2012

References clock_rate, cputime, cycles1, cycles2, cycles_max, and status.

Referenced by mhm_driver(), and mrm_driver().

Here is the caller graph for this function:



15.81.3 Variable Documentation

15.81.3.1 clock_rate

```
real(sp), save, public mo_timer::clock_rate
```

clock_rate in seconds for each cycle

Referenced by timer_stop(), and timers_init().

15.81.3.2 cputime

```
real(sp), dimension(max_timers), save, public mo_timer::cputime
```

accumulated cpu time in each timer

Referenced by timer_clear(), timer_get(), timer_print(), timer_stop(), and timers_init().

15.81.3.3 cycles1

```
integer(i4), dimension(max_timers), save, public mo_timer::cycles1
```

cycle number at start for each timer

Referenced by timer_start(), timer_stop(), and timers_init().

15.81.3.4 cycles2

```
integer(i4), dimension(max_timers), save, public mo_timer::cycles2
```

cycle number at stop for each timer

Referenced by timer_stop(), and timers_init().

15.81.3.5 cycles_max

```
integer(i4), save, public mo_timer::cycles_max
```

max value of clock allowed by system

Referenced by timer_stop(), and timers_init().

15.81.3.6 max_timers

```
integer(i4), parameter, public mo_timer::max_timers = 99
```

max number of timers allowed

15.81.3.7 status

```
character(len=8), dimension(max_timers), save, public mo_timer::status
```

timer status string

Referenced by timer_check(), timer_get(), timer_print(), timer_start(), timer_stop(), and timers_init().

15.82 mo_upscaling_operators Module Reference

Module containing upscaling operators.

Functions/Subroutines

- integer(i4) function, dimension(size(l1_upper_rowid_cell, 1)), public [majority_statistics](#) (nClass, L1_upper_rowid_cell, L1_lower_rowid_cell, L1_left_colonId_cell, L1_right_colonId_cell, L0_fineScale_2D_data)
majority statistics
- real(dp) function, dimension(size(l0upbound_inLx, 1)), public [l0_fractionalcover_in_Lx](#) (dataIn0, classId, mask0, L0upBound_inLx, L0downBound_inLx, L0leftBound_inLx, L0rightBound_inLx, nTCells0_inLx)
fractional coverage of a given class of L0 fields in Lx field (Lx = L1 or L11)

- `real(dp) function, dimension(size(nl0_cells_in_l1_cell, 1)), public upscale_arithmetic_mean (nl0_cells_in↵
_l1_cell, L1_upper_rowld_cell, L1_lower_rowld_cell, L1_left_colonld_cell, L1_right_colonld_cell, L0_cellld,
mask0, nodata_value, L0_fineScale_data)`
arithmetic mean
- `real(dp) function, dimension(size(nl0_cells_in_l1_cell, 1)), public upscale_harmonic_mean (nl0_cells_in↵
_l1_cell, L1_upper_rowld_cell, L1_lower_rowld_cell, L1_left_colonld_cell, L1_right_colonld_cell, L0_cellld,
mask0, nodata_value, L0_fineScale_data)`
harmonic mean
- `real(dp) function, dimension(size(l1_upper_rowld_cell, 1)), public upscale_geometric_mean (L1_upper_↵
rowld_cell, L1_lower_rowld_cell, L1_left_colonld_cell, L1_right_colonld_cell, mask0, nodata_value, L0_↵
fineScale_data)`
geometric mean

15.82.1 Detailed Description

Module containing upscaling operators.

This module provides the routines for upscaling_operators.

Authors

Giovanni Dalmasso, Rohini Kumar

Date

Dec 2012

15.82.2 Function/Subroutine Documentation

15.82.2.1 l0_fractionalcover_in_lx()

```
real(dp) function, dimension( size(l0upbound_inlx,1) ), public mo_upscaling_operators::l0_↵
fractionalcover_in_lx (
    integer(i4), dimension(:), intent(in) dataIn0,
    integer(i4), intent(in) classId,
    logical, dimension(:,:), intent(in) mask0,
    integer(i4), dimension(:), intent(in) L0upBound_inLx,
    integer(i4), dimension(:), intent(in) L0downBound_inLx,
    integer(i4), dimension(:), intent(in) L0leftBound_inLx,
    integer(i4), dimension(:), intent(in) L0rightBound_inLx,
    integer(i4), dimension(:), intent(in) nTCells0_inLx )
```

fractional coverage of a given class of L0 fields in Lx field (Lx = L1 or L11)

Fractional coverage of a given class of L0 fields in Lx field (Lx = L1 or L11). For example, this routine can be used for calculating the karstic fraction.

Parameters

in	<i>integer(i4), dimension(:,:) :: dataIn0</i>	input fields at finer scale
in	<i>integer(i4) :: classId</i>	class id for which fraction has to be estimated
in	<i>logical, dimension(:,:) :: mask0</i>	finer scale L0 mask
in	<i>integer(i4), dimension(:) :: L0upBound_inLx</i>	row start at finer L0 scale
in	<i>integer(i4), dimension(:) :: L0downBound_inLx</i>	row end at finer L0 scale

in	<i>integer(i4), dimension(:) :: L0leftBound_inLx</i>	col start at finer L0 scale
in	<i>integer(i4), dimension(:) :: L0rightBound_inLx</i>	col end at finer L0 scale
in	<i>integer(i4), dimension(:) :: nTCells0_inLx</i>	total number of valid L0 cells in a given Lx cell

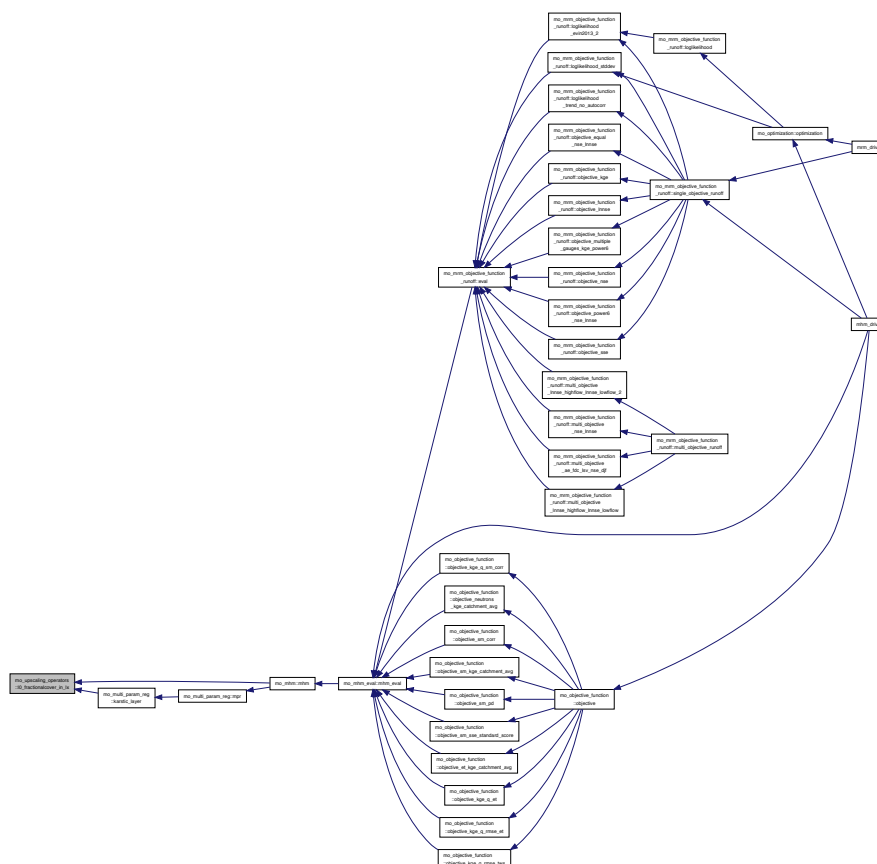
real(dp) :: L0_fractionalCover_in_Lx(:) — packed 1D fraction coverage (Lx) of given class id

Rohini Kumar

Feb 2013

Referenced by `mo_multi_param_reg::karstic_layer()`, and `mo_mhm::mhm()`.

Here is the caller graph for this function:



15.82.2.2 majority_statistics()

```
integer(i4) function, dimension( size(l1_upper_rowId_cell,1) ), public mo_upscaling_operators←
::majority_statistics (
    integer(i4), intent(in) nClass,
    integer(i4), dimension(:), intent(in) L1_upper_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_lower_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_left_colonId_cell,
    integer(i4), dimension(:), intent(in) L1_right_colonId_cell,
    integer(i4), dimension(:, :), intent(in) L0_fineScale_2D_data )
```

majority statistics

upscale grid L0_fineScale_2D_data based on a majority statistics

Parameters

in	<i>integer(i4) :: nClass</i>	number of classes
in	<i>integer(i4) :: L1_upper_rowId_cell(:)</i>	upper row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_lower_rowId_cell(:)</i>	lower row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_left_colonId_cell(:)</i>	left colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_right_colonId_cell(:)</i>	right colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L0_fineScale_2D_data(:, :)</i>	high resolution data

Returns

integer(i4) :: majority_statistics(:) — Upscaled variable based on majority.

Note

Input values must be floating points.

Author

Giovanni Dalmasso, Rohini Kumar

Date

Dec 2012

15.82.2.3 upscale_arithmetic_mean()

```
real(dp) function, dimension( size(nl0_cells_in_l1_cell,1) ), public mo_upscaling_operators←
::upscale_arithmetic_mean (
    integer(i4), dimension(:), intent(in) nL0_cells_in_L1_cell,
    integer(i4), dimension(:), intent(in) L1_upper_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_lower_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_left_colonId_cell,
    integer(i4), dimension(:), intent(in) L1_right_colonId_cell,
    integer(i4), dimension(:), intent(in) L0_cellId,
    logical, dimension(:, :), intent(in) mask0,
    real(dp), intent(in) nodata_value,
    real(dp), dimension(:), intent(in) L0_fineScale_data )
```

arithmetic mean

upscaling of level-0 grid data to level-1 using arithmetic mean

Parameters

in	<i>integer(i4) :: nL0_cells_in_L1_cell(:)</i>	number of level-0 cells within a level-1 cell
in	<i>integer(i4) :: L1_upper_rowld_cell(:)</i>	upper row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_lower_rowld_cell(:)</i>	lower row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_left_colonld_cell(:)</i>	left colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_right_colonld_cell(:)</i>	right colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L0_cellld(:, :)</i>	cell ID at level-0
in	<i>logical :: mask0(:, :)</i>	mask at Level 0
in	<i>real(dp) :: nodata_value</i>	no data value
in	<i>real(dp) :: L0_fineScale_data(:, :)</i>	high resolution data

Returns

real(dp) :: upscale_arithmetic_mean(:) — Upscaled variable from L0 to L1 using arithmetic mean

Author

Giovanni Dalmaso, Rohini Kumar

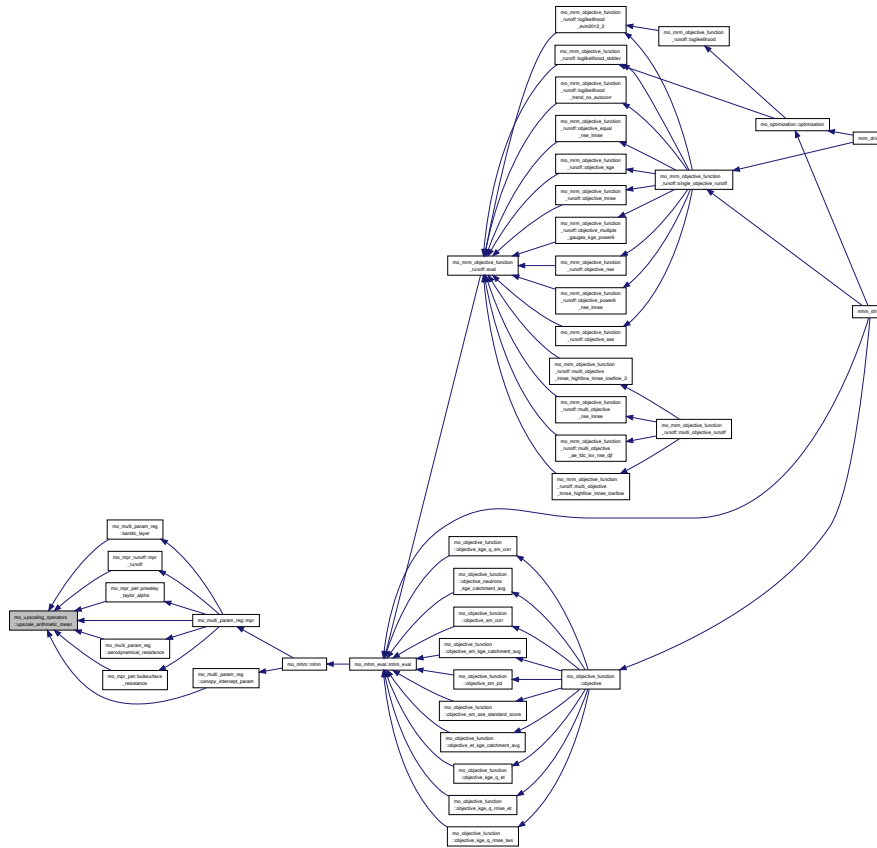
Date

Dec 2012

References *mo_kind::i4*.

Referenced by *mo_multi_param_reg::aerodynamical_resistance()*, *mo_mpr_pet::bulksurface_resistance()*, *mo_multi_param_reg::canopy_intercept_param()*, *mo_multi_param_reg::karstic_layer()*, *mo_multi_param_reg::mpr()*, *mo_mpr_runoff::mpr_runoff()*, and *mo_mpr_pet::priestley_taylor_alpha()*.

Here is the caller graph for this function:



15.82.2.4 upscale_geometric_mean()

```
real(dp) function, dimension( size(l1_upper_rowId_cell,1) ), public mo_upscaling_operators←
::upscale_geometric_mean (
    integer(i4), dimension(:), intent(in) L1_upper_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_lower_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_left_colonId_cell,
    integer(i4), dimension(:), intent(in) L1_right_colonId_cell,
    logical, dimension(:,:), intent(in) mask0,
    real(dp), intent(in) nodata_value,
    real(dp), dimension(:), intent(in) L0_fineScale_data )
```

geometric mean

upscaling of level-0 grid data to level-1 using geometric mean

Parameters

in	<i>integer(i4) :: L1_upper_rowId_cell(:)</i>	upper row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_lower_rowId_cell(:)</i>	lower row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_left_colonId_cell(:)</i>	left colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_right_colonId_cell(:)</i>	right colon boundary (level-0) of a level-1 cell
in	<i>logical :: mask0(:,:)</i>	mask at Level 0

Parameters

in	<i>real(dp) :: nodata_value</i>	no data value
in, out	<i>real(dp)</i>	:: L0_fineScale_2D_data(:,:) high resolution data

Returns

real(dp) :: upscale_geometric_mean(:) — Upscaled variable from L0 to L1 using geometric mean

Author

Giovanni Dalmasso, Rohini Kumar

Date

Dec 2012

15.82.2.5 upscale_harmonic_mean()

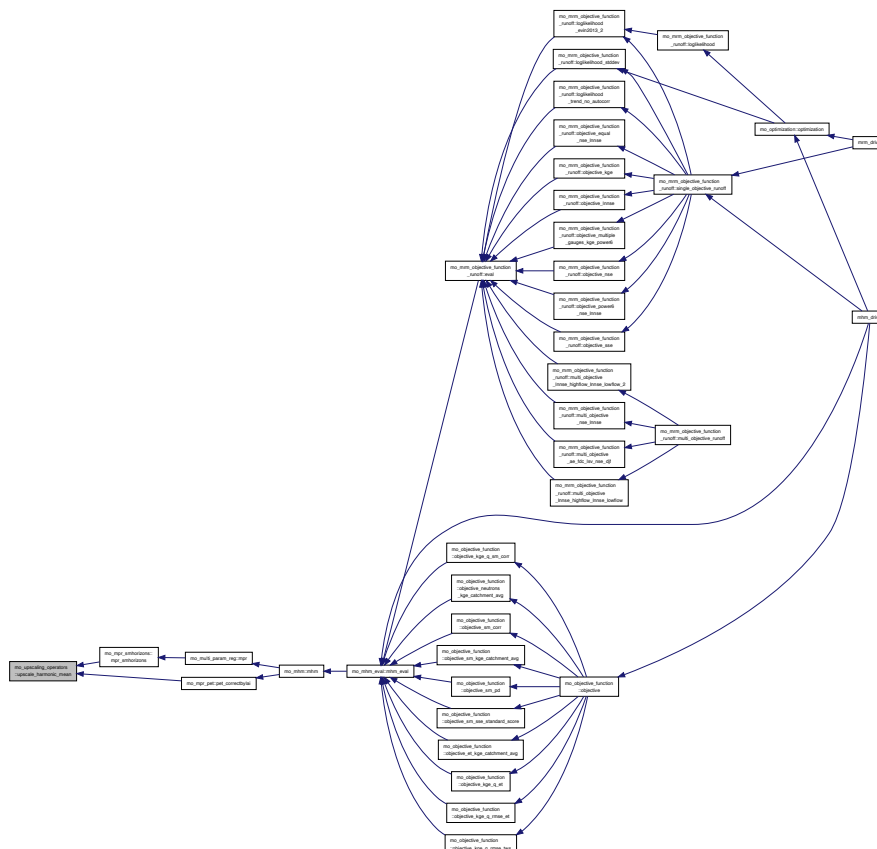
```
real(dp) function, dimension( size(nl0_cells_in_l1_cell,1) ), public mo_upscaling_operators←
::upscale_harmonic_mean (
    integer(i4), dimension(:), intent(in) nl0_cells_in_L1_cell,
    integer(i4), dimension(:), intent(in) L1_upper_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_lower_rowId_cell,
    integer(i4), dimension(:), intent(in) L1_left_colonId_cell,
    integer(i4), dimension(:), intent(in) L1_right_colonId_cell,
    integer(i4), dimension(:), intent(in) L0_cellId,
    logical, dimension(:,:), intent(in) mask0,
    real(dp), intent(in) nodata_value,
    real(dp), dimension(:), intent(in) L0_fineScale_data )
```

harmonic mean

upscaling of level-0 grid data to level-1 using harmonic mean

Parameters

in	<i>integer(i4) :: nl0_cells_in_L1_cell(:)</i>	number of level-0 cells within a level-1 cell
in	<i>integer(i4) :: L1_upper_rowId_cell(:)</i>	upper row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_lower_rowId_cell(:)</i>	lower row boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_left_colonId_cell(:)</i>	left colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L1_right_colonId_cell(:)</i>	right colon boundary (level-0) of a level-1 cell
in	<i>integer(i4) :: L0_cellId(:,:)</i>	cell ID at level-0
in	<i>logical :: mask0(:,:)</i>	mask at Level 0
in	<i>real(dp) :: nodata_value</i>	no data value
in	<i>real(dp) :: L0_fineScale_2D_data(:,:)</i>	high resolution data



Data Types

- interface [eq](#)
- interface [equal](#)
 - Comparison of real values.*
- interface [ge](#)
- interface [greaterequal](#)
- interface [is_finite](#)
 - .true. if not IEEE Inf, IEEE NaN, nor IEEE Inf nor IEEE NaN, respectively.*
- interface [is_nan](#)
- interface [is_normal](#)
- interface [le](#)
- interface [lesserequal](#)
- interface [locate](#)
 - Find closest values in a monotonic series, returns the indexes.*
- interface [ne](#)
- interface [notequal](#)
- interface [special_value](#)
 - Special IEEE values.*
- interface [swap](#)
 - Swap to values or two elements in array.*

Functions/Subroutines

- elemental pure logical function [equal_dp](#) (a, b)
- elemental pure logical function [equal_sp](#) (a, b)
- elemental pure logical function [greaterequal_dp](#) (a, b)
- elemental pure logical function [greaterequal_sp](#) (a, b)
- elemental pure logical function [lesserequal_dp](#) (a, b)
- elemental pure logical function [lesserequal_sp](#) (a, b)
- elemental pure logical function [notequal_dp](#) (a, b)
- elemental pure logical function [notequal_sp](#) (a, b)
- elemental pure logical function [is_finite_dp](#) (a)
- elemental pure logical function [is_finite_sp](#) (a)
- elemental pure logical function [is_nan_dp](#) (a)
- elemental pure logical function [is_nan_sp](#) (a)
- elemental pure logical function [is_normal_dp](#) (a)
- elemental pure logical function [is_normal_sp](#) (a)
- integer(i4) function [locate_0d_dp](#) (x, y)
- integer(i4) function [locate_0d_sp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [locate_1d_dp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [locate_1d_sp](#) (x, y)
- elemental pure subroutine [swap_xy_dp](#) (x, y)
- elemental pure subroutine [swap_xy_sp](#) (x, y)
- elemental pure subroutine [swap_xy_i4](#) (x, y)
- subroutine [swap_vec_dp](#) (x, i1, i2)
- subroutine [swap_vec_sp](#) (x, i1, i2)
- subroutine [swap_vec_i4](#) (x, i1, i2)
- real(dp) function [special_value_dp](#) (x, ieee)
- real(sp) function [special_value_sp](#) (x, ieee)

15.83.1 Detailed Description

General utilities for the CHS library.

This module provides general utilities such as comparisons of two reals.

Authors

Matthias Cuntz, Juliane Mai

Date

Feb 2014

15.83.2 Function/Subroutine Documentation

15.83.2.1 equal_dp()

```
elemental pure logical function mo_utils::equal_dp (  
    real(dp), intent(in) a,  
    real(dp), intent(in) b ) [private]
```

15.83.2.2 equal_sp()

```
elemental pure logical function mo_utils::equal_sp (  
    real(sp), intent(in) a,  
    real(sp), intent(in) b ) [private]
```

15.83.2.3 greaterequal_dp()

```
elemental pure logical function mo_utils::greaterequal_dp (  
    real(dp), intent(in) a,  
    real(dp), intent(in) b ) [private]
```

15.83.2.4 greaterequal_sp()

```
elemental pure logical function mo_utils::greaterequal_sp (  
    real(sp), intent(in) a,  
    real(sp), intent(in) b ) [private]
```

15.83.2.5 is_finite_dp()

```
elemental pure logical function mo_utils::is_finite_dp (  
    real(dp), intent(in) a ) [private]
```

15.83.2.6 is_finite_sp()

```
elemental pure logical function mo_utils::is_finite_sp (  
    real(sp), intent(in) a ) [private]
```

15.83.2.7 is_nan_dp()

```
elemental pure logical function mo_utils::is_nan_dp (  
    real(dp), intent(in) a ) [private]
```

15.83.2.8 is_nan_sp()

```
elemental pure logical function mo_utils::is_nan_sp (  
    real(sp), intent(in) a ) [private]
```

15.83.2.9 is_normal_dp()

```
elemental pure logical function mo_utils::is_normal_dp (  
    real(dp), intent(in) a ) [private]
```

15.83.2.10 is_normal_sp()

```
elemental pure logical function mo_utils::is_normal_sp (  
    real(sp), intent(in) a ) [private]
```

15.83.2.11 lesserequal_dp()

```
elemental pure logical function mo_utils::lesserequal_dp (  
    real(dp), intent(in) a,  
    real(dp), intent(in) b ) [private]
```

15.83.2.12 lesserequal_sp()

```
elemental pure logical function mo_utils::lesserequal_sp (  
    real(sp), intent(in) a,  
    real(sp), intent(in) b ) [private]
```

15.83.2.13 locate_0d_dp()

```
integer(i4) function mo_utils::locate_0d_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), intent(in) y ) [private]
```

15.83.2.14 locate_0d_sp()

```
integer(i4) function mo_utils::locate_0d_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), intent(in) y ) [private]
```

15.83.2.15 locate_1d_dp()

```
integer(i4) function, dimension(:), allocatable mo_utils::locate_1d_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y ) [private]
```

15.83.2.16 locate_1d_sp()

```
integer(i4) function, dimension(:), allocatable mo_utils::locate_1d_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y ) [private]
```

15.83.2.17 notequal_dp()

```
elemental pure logical function mo_utils::notequal_dp (  
    real(dp), intent(in) a,  
    real(dp), intent(in) b ) [private]
```

15.83.2.18 notequal_sp()

```
elemental pure logical function mo_utils::notequal_sp (  
    real(sp), intent(in) a,  
    real(sp), intent(in) b ) [private]
```

15.83.2.19 special_value_dp()

```
real(dp) function mo_utils::special_value_dp (  
    real(dp), intent(in) x,  
    character(len=*), intent(in) ieee ) [private]
```

References mo_string_utils::toupper().

Here is the call graph for this function:



15.83.2.20 special_value_sp()

```
real(sp) function mo_utils::special_value_sp (  
    real(sp), intent(in) x,  
    character(len=*), intent(in) ieee ) [private]
```

References mo_string_utils::toupper().

Here is the call graph for this function:



15.83.2.21 swap_vec_dp()

```
subroutine mo_utils::swap_vec_dp (  
    real(dp), dimension(:), intent(inout) x,  
    integer(i4), intent(in) i1,  
    integer(i4), intent(in) i2 ) [private]
```

15.83.2.22 swap_vec_i4()

```
subroutine mo_utils::swap_vec_i4 (  
    integer(i4), dimension(:), intent(inout) x,  
    integer(i4), intent(in) i1,  
    integer(i4), intent(in) i2 ) [private]
```

15.83.2.23 swap_vec_sp()

```
subroutine mo_utils::swap_vec_sp (
    real(sp), dimension(:), intent(inout) x,
    integer(i4), intent(in) i1,
    integer(i4), intent(in) i2 ) [private]
```

15.83.2.24 swap_xy_dp()

```
elemental pure subroutine mo_utils::swap_xy_dp (
    real(dp), intent(inout) x,
    real(dp), intent(inout) y ) [private]
```

15.83.2.25 swap_xy_i4()

```
elemental pure subroutine mo_utils::swap_xy_i4 (
    integer(i4), intent(inout) x,
    integer(i4), intent(inout) y ) [private]
```

15.83.2.26 swap_xy_sp()

```
elemental pure subroutine mo_utils::swap_xy_sp (
    real(sp), intent(inout) x,
    real(sp), intent(inout) y ) [private]
```

15.84 mo_write_ascii Module Reference

Module to write ascii file output.

Functions/Subroutines

- subroutine, public [write_configfile](#) ()
This modules writes the results of the configuration into an ASCII-file.
- subroutine, public [write_optifile](#) (best_OF, best_paramSet, param_names)
Write briefly final optimization results.
- subroutine, public [write_optinamelist](#) (processMatrix, parameters, maskpara, parameters_name)
Write final, optimized parameter set in a namelist format.

15.84.1 Detailed Description

Module to write ascii file output.

Module to write ascii file output.

Writing model output to ASCII should be the exception. Therefore, output is written usually as NetCDF and only:

- (1) The configuration file of mHM,
- (2) the final parameter set after optimization, and
- (3) the simulated vs. observed daily discharge

is written in ASCII file format to allow for a quick assurance of proper model runs.

Authors

Christoph Schneider, Juliane Mai, Luis Samaniego

Date

May 2013

15.84.2 Function/Subroutine Documentation**15.84.2.1 write_configfile()**

```
subroutine, public mo_write_ascii::write_configfile ( )
```

This module writes the results of the configuration into an ASCII-file.

Author

Christoph Schneider

Date

May 2013

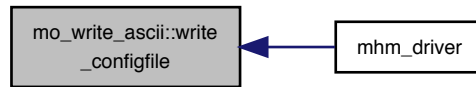
References mo_global_variables::basin, mo_mrm_global_variables::basin_mrm, mo_global_variables::dirconfigout, mo_mrm_global_variables::dirgauges, mo_global_variables::dircover, mo_global_variables::dirmorpho, mo_global_variables::dirout, mo_global_variables::dirprecipitation, mo_global_variables::dirreferenceet, mo_global_variables::dirrestartout, mo_global_variables::dirtemperature, mo_kind::dp, mo_global_variables::evalper, mo_file::file_config, mo_mrm_global_variables::gauge, mo_common_variables::global_parameters, mo_common_variables::global_parameters_name, mo_kind::i4, mo_global_variables::iflag_cordinate_sys, mo_mrm_global_variables::inflowgauge, mo_global_variables::l0_ncells, mo_mrm_global_variables::l11_fromn, mo_mrm_global_variables::l11_id, mo_mrm_global_variables::l11_label, mo_mrm_global_variables::l11_length, mo_mrm_global_variables::l11_ncells, mo_mrm_global_variables::l11_netperm, mo_mrm_global_variables::l11_rorder, mo_mrm_global_variables::l11_slope, mo_mrm_global_variables::l11_ton, mo_mrm_global_variables::l1_areacell, mo_mrm_global_variables::l1_l11_id, mo_global_variables::l1_ncells, mo_global_variables::lcfilename, mo_global_variables::lcyetid, mo_message::message(), mo_global_variables::nbasins, mo_mrm_global_variables::ngauestotal, mo_mrm_global_variables::ninflowgauestotal, mo_mhm_constants::nodata_dp, mo_common_variables::processmatrix, mo_global_variables::read_restart, mo_global_variables::resolutionhydrology, mo_mrm_global_variables::resolutionrouting, mo_global_variables::simper, mo_global_variables::timestep, mo_file::uconfig, mo_file::version, mo_global_variables::warmper, and mo_global_variables::write_restart.

Referenced by mhm_driver().

Here is the call graph for this function:



Here is the caller graph for this function:



15.84.2.2 write_optifile()

```

subroutine, public mo_write_ascii::write_optifile (
    real(dp), intent(in) best_OF,
    real(dp), dimension(:), intent(in) best_paramSet,
    character(len=*), dimension(:), intent(in) param_names )
  
```

Write briefly final optimization results.

Write overall best objective function and the best optimized parameter set to a file_opti.

Parameters

in	<i>real(dp) :: best_OF</i>	best objective function value as returned by the optimization routine
in	<i>real(dp), dimension(:) :: best_paramSet</i>	best associated global parameter set Called only when optimize is .TRUE.

Author

David Schaefer

Date

July 2013

References mo_global_variables::dirconfigout, mo_file::file_opti, mo_message::message(), and mo_file::uopti.

Here is the call graph for this function:



15.84.2.3 write_optinamelist()

```
subroutine, public mo_write_ascii::write_optinamelist (
    integer(i4), dimension(nprocesses, 3), intent(in) processMatrix,
    real(dp), dimension(:, :), intent(in) parameters,
    logical, dimension(size(parameters,1)), intent(in) maskpara,
    character(len=*), dimension(size(parameters,1)), intent(in) parameters_name )
```

Write final, optimized parameter set in a namelist format.

Write final, optimized parameter set in a namelist format. Only parameters of processes which were switched on are written to the namelist. All others are discarded.

Parameters

in	<i>integer(i4) :: processMatrix(:, :)</i>	information on which process is switched on and off
in	<i>real(dp) :: parameters(:, :)</i>	information about parameter (min, max, opti)
in	<i>logical :: maskpara(:)</i>	information which parameter where optimized
in	<i>character(len=*) :: parameters_name(:)</i>	clear names of parameters Called only when optimize is .TRUE.

Author

Juliane Mai

Date

Dec 2013

References mo_global_variables::dirconfigout, mo_file::file_opti_nml, mo_message::message(), mo_common_variables::nprocesses, and mo_file::uopti_nml.

Here is the call graph for this function:



15.85 mo_write_fluxes_states Module Reference

Creates NetCDF output for different fluxes and state variables of mHM.

Data Types

- interface [outputdataset](#)
- interface [outputvariable](#)

Functions/Subroutines

- type([outputvariable](#)) function [newoutputvariable](#) (nc, name, dtype, dims, ncells, mask, avg)
Initialize OutputVariable.
- subroutine [updatevariable](#) (self, data)
Update OutputVariable.
- subroutine [writevariabletimestep](#) (self, timestep)
Write timestep to file.
- type([outputdataset](#)) function, public [newoutputdataset](#) (ibasin, mask1)
Initialize OutputDataset.
- subroutine [updatedataset](#) (self, sidx, eidx, L1_fSealed, L1_fNotSealed, L1_inter, L1_snowPack, L1_soil↔
Moist, L1_soilMoistSat, L1_sealSTW, L1_unsatSTW, L1_satSTW, L1_neutrons, L1_pet, L1_aETSoil, L1↔
_aETCanopy, L1_aETSealed, L1_total_runoff, L1_runoffSeal, L1_fastRunoff, L1_slowRunoff, L1_baseflow,
L1_percol, L1_infilSoil, L1_preEffect)
Update all variables.
- subroutine [writetimestep](#) (self, timestep)
Write all accumulated data.
- subroutine [close](#) (self)
Close the file.
- type([ncdataset](#)) function [createoutputfile](#) (ibasin)
Create and initialize output file.
- subroutine [writevariableattributes](#) (var, long_name, unit)
Write output variable attributes.
- subroutine [mapcoordinates](#) (ibasin, level, y, x)
Generate map coordinates.
- subroutine [geocoordinates](#) (ibasin, level, lat, lon)
Generate geographic coordinates.
- character(16) function [fluxesunit](#) (ibasin)
Generate a unit string.

15.85.1 Detailed Description

Creates NetCDF output for different fluxes and state variables of mHM.

NetCDF is first initialized and later on variables are put to the NetCDF.

Authors

Matthias Zink

Date

Apr 2013

15.85.2 Function/Subroutine Documentation

15.85.2.1 close()

```
subroutine mo_write_fluxes_states::close (
    class(outputdataset) self ) [private]
```

Close the file.

Close the file associated with variable of type(OutputDataset)

Author

Rohini Kumar & Stephan Thober

Date

August 2013

References mo_global_variables::dirout, and mo_message::message().

Here is the call graph for this function:



15.85.2.2 createoutputfile()

```
type(ncdataset) function mo_write_fluxes_states::createoutputfile (
    integer(i4), intent(in) ibasin )
```

Create and initialize output file.

Create output file, write all non-dynamic variables and global attributes for the given basin.

Parameters

in	<i>integer(i4) :: ibasin</i>	-> basin id
in	<i>logical, target :: mask1(:, :)</i>	-> level1 mask

Author

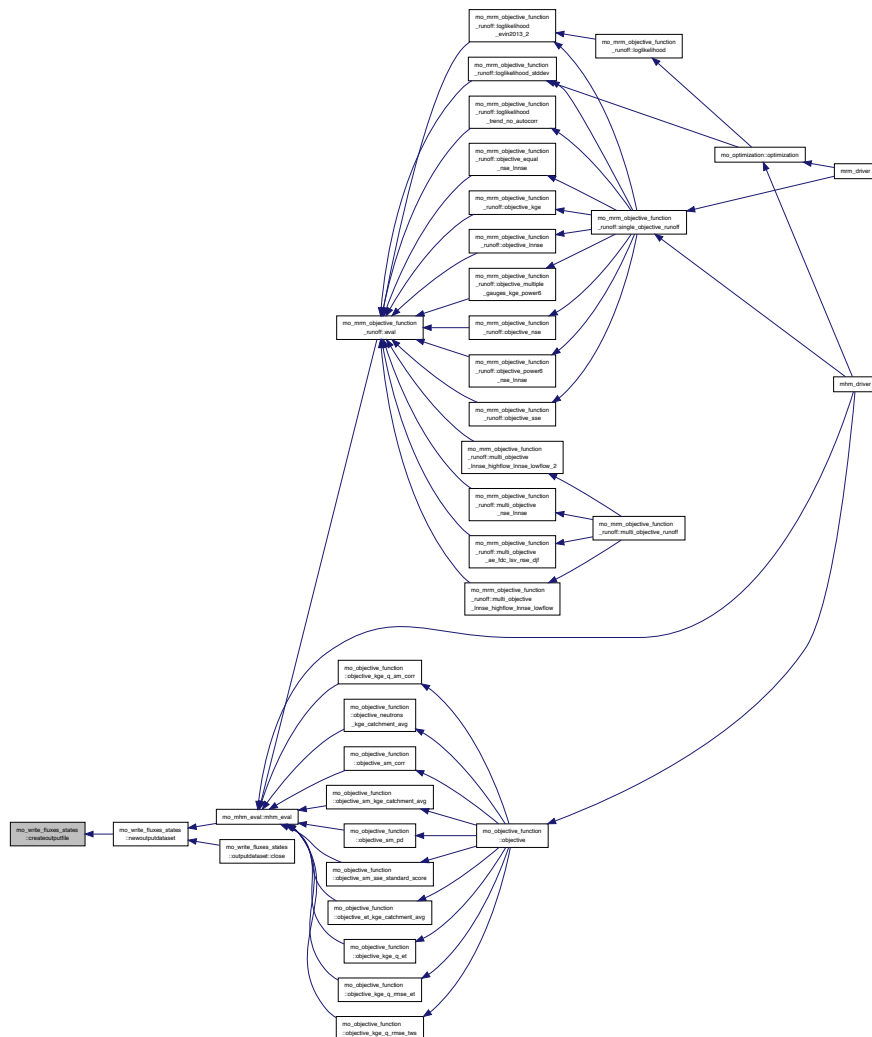
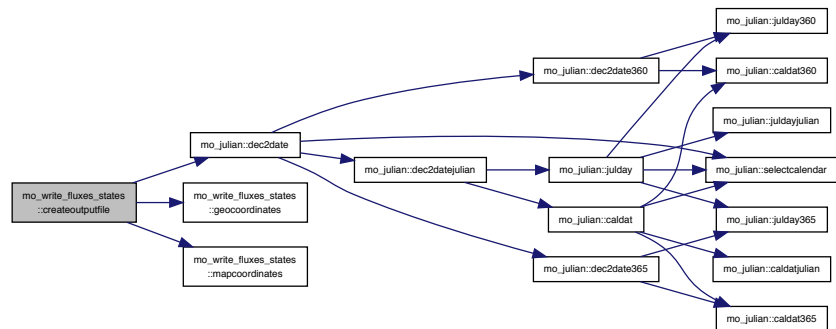
David Schaefer

Date

June 2015

References mo_julian::dec2date(), mo_global_variables::dirout, mo_global_variables::evalper, geocoordinates(), mo_global_variables::level1, mapcoordinates(), and mo_file::version.

Here is the call graph for this function:



15.85.2.3 fluxesunit()

```
character(16) function mo_write_fluxes_states::fluxesunit (
    integer(i4), intent(in) ibasin )
```

Generate a unit string.

Generate the unit string for the output variable netcdf attribute based on modeling timestep

Parameters

in	<i>integer(i4) :: iBasin</i>	-> basin id
----	------------------------------	-------------

Author

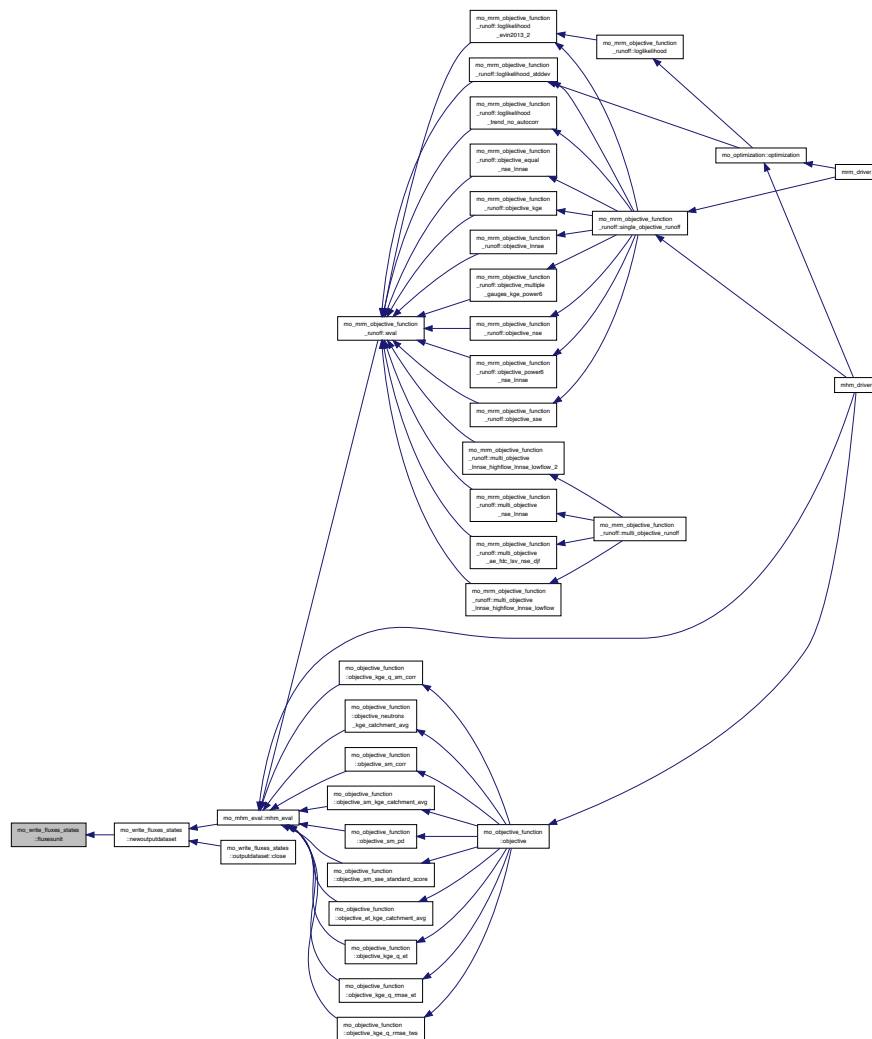
David Schaefer

Date

June 2015

References `mo_global_variables::ntstepday`, `mo_global_variables::simper`, `mo_global_variables::timestep`, and `mo_global_variables::timestep_model_outputs`.

Referenced by `newoutputdataset()`.



in	<i>integer(i4) :: iBasin</i>	-> basin number
in	<i>type(gridGeoRef) :: level</i>	-> grid reference
out	<i>real(dp) :: lat(:, :)</i>	-> lat-coordinates
out	<i>real(dp) :: lon(:, :)</i>	-> lon-coorindates

Author

Matthias Zink

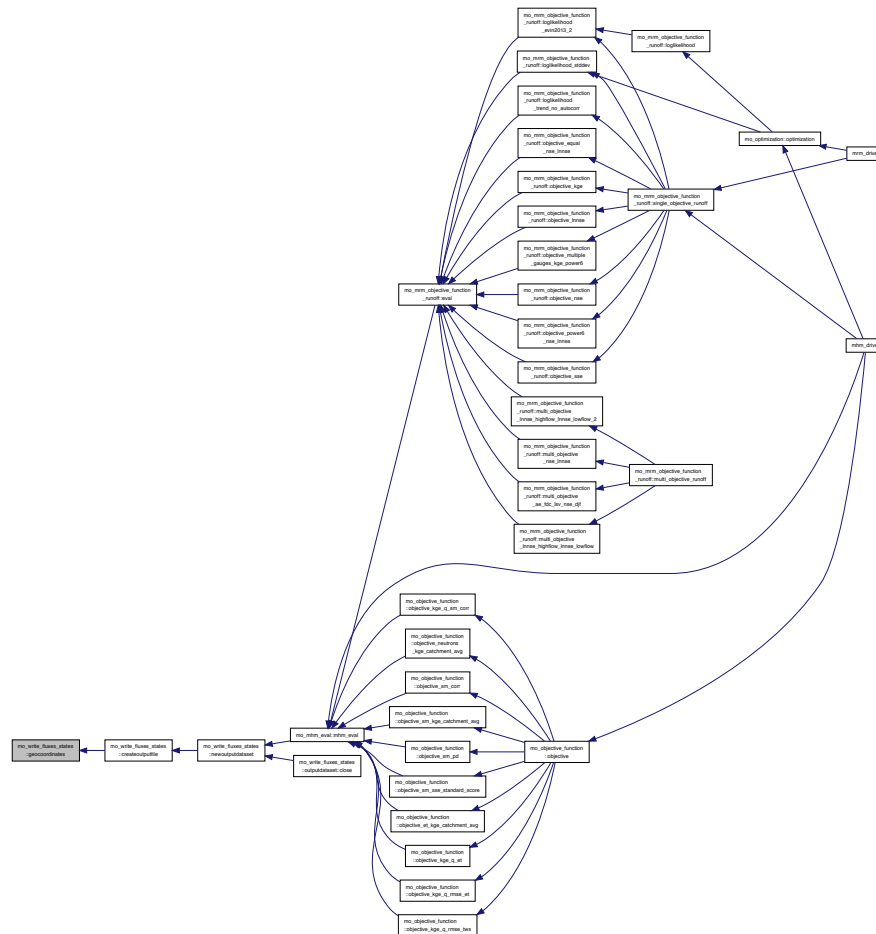
Date

Apr 2013

References `mo_global_variables::l1_rect_latitude`, and `mo_global_variables::l1_rect_longitude`.

Referenced by `createoutputfile()`.

Here is the caller graph for this function:



15.85.2.5 mapcoordinates()

```
subroutine mo_write_fluxes_states::mapcoordinates (
  integer(i4), intent(in) ibasin,
  type(gridgeoref), intent(in) level,
  real(dp), dimension(:), intent(out), allocatable y,
  real(dp), dimension(:), intent(out), allocatable x ) [private]
```

Generate map coordinates.

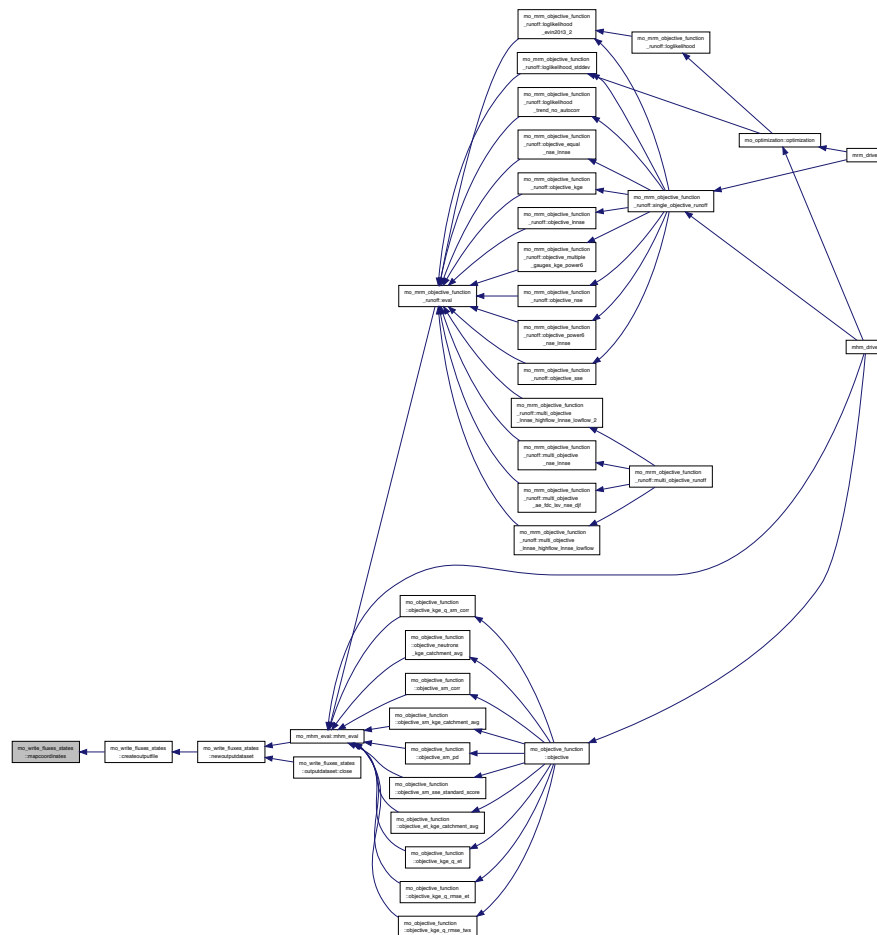
Generate map coordinate arrays for given basin and level

in	<i>integer(i4) :: iBasin</i>	-> basin number
in	<i>type(geoGridRef) :: level</i>	-> grid reference
out	<i>real(:) :: y(:)</i>	-> y-coordinates
out	<i>real(dp) :: x(:)</i>	-> x-coorindates

Matthias Zink

Apr 2013

Here is the caller graph for this function:



```
type(outputdataset) function, public mo_write_fluxes_states::newoutputdataset (
```

```
integer(i4), intent(in) ibasin,
logical, dimension(:,:), target mask1 )
```

Initialize OutputDataset.

Create and initialize the output file. If new a new output variable needs to be written, this is the first of two procedures to change (second: updateDataset)

Parameters

in	<i>integer(i4) :: ibasin</i>	-> basin id
in	<i>logical :: mask1</i>	-> L1 mask to reconstruct the data

Author

Matthias Zink

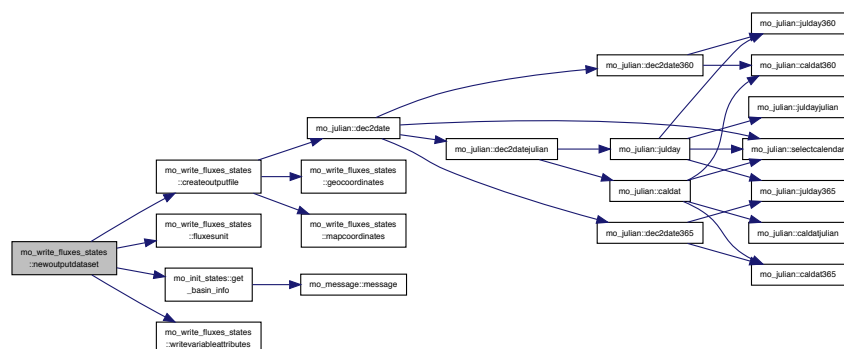
Date

Apr 2013

References createoutputfile(), fluxesunit(), mo_init_states::get_basin_info(), mo_global_variables::nsoilhorizons_↔ mhm, mo_global_variables::outputflxstate, and writevariableattributes().

Referenced by mo_write_fluxes_states::outputdataset::close(), and mo_mhm_eval::mhm_eval().

Here is the call graph for this function:



```

type(outputvariable) function mo_write_fluxes_states::newoutputvariable (
  type(ncdataset), intent(in) nc,
  character(*), intent(in) name,
  character(*), intent(in) dtype,
  character(16), dimension(3), intent(in) dims,
  integer(i4), intent(in) ncells,
  logical, dimension(:, :), intent(in), target mask,
  logical, intent(in), optional avg ) [private]

```

Generated on December 1, 2017

Parameters

in	<i>type(NcDataset) :: nc</i>	-> NcDataset which contains the variable
in	<i>integer(i4) :: ncells</i>	-> number of cells in basin
in	<i>logical, target :: mask(:, :)</i>	-> mask to reconstruct data
in	<i>logical, optional :: avg</i>	-> average the data before writing

Author

David Schaefer

Date

June 2015

15.85.2.8 updatedataset()

```

subroutine mo_write_fluxes_states::updatedataset (
    class(outputdataset), intent(inout), target self,
    integer(i4), intent(in) sidx,
    integer(i4), intent(in) eidx,
    real(dp), dimension(:), intent(in) L1_fSealed,
    real(dp), dimension(:), intent(in) L1_fNotSealed,
    real(dp), dimension(:), intent(in) L1_inter,
    real(dp), dimension(:), intent(in) L1_snowPack,
    real(dp), dimension(:, :), intent(in) L1_soilMoist,
    real(dp), dimension(:, :), intent(in) L1_soilMoistSat,
    real(dp), dimension(:), intent(in) L1_sealSTW,
    real(dp), dimension(:), intent(in) L1_unsatSTW,
    real(dp), dimension(:), intent(in) L1_satSTW,
    real(dp), dimension(:), intent(in) L1_neutrons,
    real(dp), dimension(:), intent(in) L1_pet,
    real(dp), dimension(:, :), intent(in) L1_aETSoil,
    real(dp), dimension(:), intent(in) L1_aETCanopy,
    real(dp), dimension(:), intent(in) L1_aETSealed,
    real(dp), dimension(:), intent(in) L1_total_runoff,
    real(dp), dimension(:), intent(in) L1_runoffSeal,
    real(dp), dimension(:), intent(in) L1_fastRunoff,
    real(dp), dimension(:), intent(in) L1_slowRunoff,
    real(dp), dimension(:), intent(in) L1_baseflow,
    real(dp), dimension(:), intent(in) L1_percol,
    real(dp), dimension(:, :), intent(in) L1_infilSoil,
    real(dp), dimension(:), intent(in) L1_preEffect )

```

Update all variables.

Call the type bound procedure `updateVariable` for all output variables. If a new output variable needs to be written, this is the second of two procedures to change (first: `newOutputDataset`)

Parameters

in	<i>sidx</i>	-> start index of the basin related data in L1_* arguments
in	<i>eidx</i>	-> end index of the basin related data in L1_* arguments
in	<i>L1_fSealed</i>	

Parameters

in	<i>L1_fNotSealed</i>	
in	<i>L1_inter</i>	
in	<i>L1_snowPack</i>	
in	<i>L1_soilMoist</i>	
in	<i>L1_soilMoistSat</i>	
in	<i>L1_sealSTW</i>	
in	<i>L1_unsatSTW</i>	
in	<i>L1_satSTW</i>	
in	<i>L1_neutrons</i>	
in	<i>L1_pet</i>	
in	<i>L1_aETSoil</i>	
in	<i>L1_aETCanopy</i>	
in	<i>L1_aETSealed</i>	
in	<i>L1_total_runoff</i>	
in	<i>L1_runoffSeal</i>	
in	<i>L1_fastRunoff</i>	
in	<i>L1_slowRunoff</i>	
in	<i>L1_baseflow</i>	
in	<i>L1_percol</i>	
in	<i>L1_infilSoil</i>	
in	<i>L1_preEffect</i>	

Author

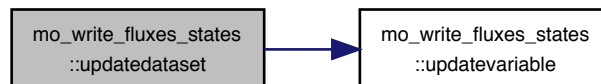
Matthias Zink

Date

Apr 2013

References mo_global_variables::nsoilhorizons_mhm, mo_global_variables::outputflxstate, and updatevariable().

Here is the call graph for this function:



15.85.2.9 updatevariable()

```

subroutine mo_write_fluxes_states::updatevariable (
  class(outputvariable), intent(inout) self,
  real(dp), dimension(:), intent(in) data ) [private]
  
```

Update OutputVariable.

Add the array given as actual argument to the derived type's component 'data'

Parameters

in	<i>type(NcDataset) :: nc</i>	-> NcDataset which contains the variable
in	<i>integer(i4) :: ncells</i>	-> number of cells in basin
in	<i>logical, target :: mask(:, :)</i>	-> mask to reconstruct data
in	<i>logical, optional :: avg</i>	-> average the data before writing

Author

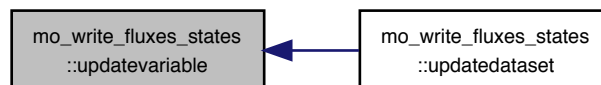
David Schaefer

Date

June 2015

Referenced by `updatedataset()`.

Here is the caller graph for this function:



15.85.2.10 writetimestep()

```

subroutine mo_write_fluxes_states::writetimestep (
    class(outputdataset), intent(inout), target self,
    integer(i4), intent(in) timestep )
  
```

Write all accumulated data.

Write all accumulated and potentially averaged data to disk.

Parameters

in	<i>integer(i4) :: timestep</i>	The model timestep to write
----	--------------------------------	-----------------------------

Author

David Schaefer

Date

June 2015

15.85.2.11 writevariableattributes()

```
subroutine mo_write_fluxes_states::writevariableattributes (
    type(outputvariable), intent(in) var,
    character(*), intent(in) long_name,
    character(*), intent(in) unit )
```

Write output variable attributes.

Parameters

in	<i>type(OutputVariable) :: var</i>	
in	<i>character(*) :: long_name</i>	-> variable name
in	<i>character(*) :: unit</i>	-> physical unit

Author

David Schaefer

Date

June 2015

References mo_mhm_constants::nodata_dp.

Referenced by newoutputdataset().

Date

June 2015

References mo_mhm_constants::nodata_dp.

15.86 mo_xor4096 Module Reference

Data Types

- interface [get_timeseed](#)
- interface [xor4096](#)
- interface [xor4096g](#)

Functions/Subroutines

- subroutine [get_timeseed_i4_0d](#) (seed)
- subroutine [get_timeseed_i4_1d](#) (seed)
- subroutine [get_timeseed_i8_0d](#) (seed)
- subroutine [get_timeseed_i8_1d](#) (seed)
- subroutine [xor4096s_0d](#) (seed, SingleIntegerRN, save_state)
- subroutine [xor4096s_1d](#) (seed, SingleIntegerRN, save_state)
- subroutine [xor4096f_0d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096f_1d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096l_0d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [xor4096l_1d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [xor4096d_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [xor4096d_1d](#) (seed, DoubleRealRN, save_state)
- subroutine [xor4096gf_0d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096gf_1d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096gd_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [xor4096gd_1d](#) (seed, DoubleRealRN, save_state)

Variables

- integer(i4), parameter, public [n_save_state](#) = 132_i4
Dimension of vector saving the state of a stream.

15.86.1 Function/Subroutine Documentation

15.86.1.1 [get_timeseed_i4_0d\(\)](#)

```
subroutine mo_xor4096::get_timeseed_i4_0d (
    integer(i4), intent(inout) seed ) [private]
```

15.86.1.2 get_timeseed_i4_1d()

```
subroutine mo_xor4096::get_timeseed_i4_1d (
    integer(i4), dimension(:), intent(inout) seed ) [private]
```

15.86.1.3 get_timeseed_i8_0d()

```
subroutine mo_xor4096::get_timeseed_i8_0d (
    integer(i8), intent(inout) seed ) [private]
```

References mo_kind::i8.

15.86.1.4 get_timeseed_i8_1d()

```
subroutine mo_xor4096::get_timeseed_i8_1d (
    integer(i8), dimension(:), intent(inout) seed ) [private]
```

References mo_kind::i8.

15.86.1.5 xor4096d_0d()

```
subroutine mo_xor4096::xor4096d_0d (
    integer(i8), intent(in) seed,
    real(dp), intent(out) DoubleRealRN,
    integer(i8), dimension(n_save_state), intent(inout), optional save_state ) [private]
```

References n_save_state.

15.86.1.6 xor4096d_1d()

```
subroutine mo_xor4096::xor4096d_1d (
    integer(i8), dimension(:), intent(in) seed,
    real(dp), dimension(size(seed,1)), intent(out) DoubleRealRN,
    integer(i8), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state ) [private]
```

References n_save_state.

15.86.1.7 xor4096f_0d()

```
subroutine mo_xor4096::xor4096f_0d (
    integer(i4), intent(in) seed,
    real(sp), intent(out) SingleRealRN,
    integer(i4), dimension(n_save_state), intent(inout), optional save_state ) [private]
```

References n_save_state.

15.86.1.8 xor4096f_1d()

```

subroutine mo_xor4096::xor4096f_1d (
    integer(i4), dimension(:), intent(in) seed,
    real(sp), dimension(size(seed)), intent(out) SingleRealRN,
    integer(i4), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state ) [private]

```

References n_save_state.

15.86.1.9 xor4096gd_0d()

```

subroutine mo_xor4096::xor4096gd_0d (
    integer(i8), intent(in) seed,
    real(dp), intent(out) DoubleRealRN,
    integer(i8), dimension(n_save_state), intent(inout), optional save_state ) [private]

```

References n_save_state.

15.86.1.10 xor4096gd_1d()

```

subroutine mo_xor4096::xor4096gd_1d (
    integer(i8), dimension(:), intent(in) seed,
    real(dp), dimension(size(seed)), intent(out) DoubleRealRN,
    integer(i8), dimension(size(seed),n_save_state), intent(inout), optional save_↵
state ) [private]

```

References n_save_state.

15.86.1.11 xor4096gf_0d()

```

subroutine mo_xor4096::xor4096gf_0d (
    integer(i4), intent(in) seed,
    real(sp), intent(out) SingleRealRN,
    integer(i4), dimension(n_save_state), intent(inout), optional save_state ) [private]

```

References n_save_state.

15.86.1.12 xor4096gf_1d()

```

subroutine mo_xor4096::xor4096gf_1d (
    integer(i4), dimension(:), intent(in) seed,
    real(sp), dimension(size(seed)), intent(out) SingleRealRN,
    integer(i4), dimension(size(seed),n_save_state), intent(inout), optional save_↵
state ) [private]

```

References n_save_state.

15.86.1.13 xor4096l_0d()

```
subroutine mo_xor4096::xor4096l_0d (
    integer(i8), intent(in) seed,
    integer(i8), intent(out) DoubleIntegerRN,
    integer(i8), dimension(n_save_state), intent(inout), optional save_state ) [private]
```

References `n_save_state`.

15.86.1.14 xor4096l_1d()

```
subroutine mo_xor4096::xor4096l_1d (
    integer(i8), dimension(:), intent(in) seed,
    integer(i8), dimension(size(seed,1)), intent(out) DoubleIntegerRN,
    integer(i8), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state ) [private]
```

References `n_save_state`.

15.86.1.15 xor4096s_0d()

```
subroutine mo_xor4096::xor4096s_0d (
    integer(i4), intent(in) seed,
    integer(i4), intent(out) SingleIntegerRN,
    integer(i4), dimension(n_save_state), intent(inout), optional save_state ) [private]
```

References `n_save_state`.

15.86.1.16 xor4096s_1d()

```
subroutine mo_xor4096::xor4096s_1d (
    integer(i4), dimension(:), intent(in) seed,
    integer(i4), dimension(size(seed,1)), intent(out) SingleIntegerRN,
    integer(i4), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state ) [private]
```

References `n_save_state`.

15.86.2 Variable Documentation**15.86.2.1 n_save_state**

```
integer(i4), parameter, public mo_xor4096::n_save_state = 132_i4
```

Dimension of vector saving the state of a stream.

Referenced by `mo_sce::cce()`, `mo_sce::getpnt()`, `mo_mcmc::mcmc_dp()`, `mo_mcmc::mcmc_stddev_dp()`, `mo_↵_sce::sce()`, `xor4096d_0d()`, `xor4096d_1d()`, `xor4096f_0d()`, `xor4096f_1d()`, `xor4096gd_0d()`, `xor4096gd_1d()`, `xor4096gf_0d()`, `xor4096gf_1d()`, `xor4096l_0d()`, `xor4096l_1d()`, `xor4096s_0d()`, and `xor4096s_1d()`.

Chapter 16

Data Type Documentation

16.1 mo_moment::absdev Interface Reference

Public Member Functions

- real(sp) function [absdev_sp](#) (dat, mask)
- real(dp) function [absdev_dp](#) (dat, mask)

16.1.1 Member Function/Subroutine Documentation

16.1.1.1 absdev_dp()

```
real(dp) function mo_moment::absdev::absdev_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

16.1.1.2 absdev_sp()

```
real(sp) function mo_moment::absdev::absdev_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.2 mo_anneal::anneal Interface Reference

anneal

Public Member Functions

- real(dp) function, dimension(size(para, 1)) [anneal_dp](#) (cost, para, prange, prange_func, temp, Dt, nITERmax, Len, nST, eps, acc, seeds, printflag, maskpara, weight, changeParaMode, reflectionFlag, pertubFlexFlag, maxit, undef_funcval, tmp_file, funcbest, history)

16.2.1 Detailed Description

anneal

Optimizes a user provided cost function using the Simulated Annealing strategy.

Parameters

in	<i>INTERFACE :: cost_dp</i>	interface calculating the cost function at a given point
in	<i>REAL(DP), DIMENSION(:) :: para</i>	initial parameter set
in	<i>REAL(DP), DIMENSION(size(para),2), optional :: prange</i>	lower and upper bound per parameter
in	<i>INTERFACE, optional :: prange_func</i>	interface calculating the feasible range for a parameter at a certain point, if ranges are variable
in	<i>REAL(DP), optional :: temp</i>	initial temperature DEFAULT: Get_Temperature
in	<i>REAL(DP), optional :: DT</i>	geometrical decreement of temperature $0.7 < DT < 0.999$ DEFAULT: 0.9_dp
in	<i>INTEGER(I4), optional :: nITERmax</i>	maximal number of iterations will be increased by 10% if stopping criteria of acceptance ratio or epsilon decreement of cost function is not fullfilled DEFAULT: 1000_i4
in	<i>INTEGER(I4), optional :: LEN</i>	Length of Markov Chain DEFAULT: MAX(250_i4, size(para,1))
in	<i>INTEGER(I4), optional :: nST</i>	Number of consecutive LEN steps DEFAULT: 5_i4
in	<i>REAL(DP), optional :: eps</i>	Stopping criteria of epsilon decreement of cost function DEFAULT: 0.0001_dp
in	<i>REAL(DP), optional :: acc</i>	Stopping criteria for Acceptance Ratio $acc \leq 0.1_dp$ DEFAULT: 0.1_dp
in	<i>INTEGER(I4/I8), DIMENSION(3), optional :: seeds</i>	Seeds of random numbers used for random parameter set generation DEFAULT: dependent on current time
in	<i>LOGICAL, optional :: printflag</i>	If .true. detailed command line output is written DEFAULT: .false.
in	<i>LOGICAL, DIMENSION(size(para)), optional :: maskpara</i>	maskpara(i) = .true. -> parameter is optimized maskpara(i) = .false. -> parameter is discarded from optimiztaion DEFAULT: .true.
in	<i>REAL(DP), DIMENSION(size(para)), optional :: weight</i>	vector of weights per parameter: gives the frequency of parameter to be chosen for optimization (will be scaled to a CDF internally) eg. [1,2,1] -> parameter 2 is chosen twice as often as parameter 1 and 2 DEFAULT: weight = 1.0_dp
in	<i>INTEGER(I4), optional :: changeParaMode</i>	which and how many param.are changed in one step 1 = one parameter 2 = all parameter 3 = neighborhood parameter DEFAULT: 1_i4

Parameters

in	<i>LOGICAL, optional :: reflectionFlag</i>	if new parameter values are Gaussian distributed and reflected (.true.) or uniform in range (.false.) DEFAULT: .false.
in	<i>LOGICAL, optional :: pertubFlexFlag</i>	if perturbation of Gaussian distributed parameter values is constant at 0.2 (.false.) or depends on dR (.true.) DEFAULT: .true.
in	<i>LOGICAL, optional :: maxit</i>	maximizing (.true.) or minimizing (.false.) a function DEFAULT: .false. (minimization)
in	<i>REAL(DP), optional :: undef_funcval</i>	objective function value defining invalid model output, e.g. -999.0_dp
in	<i>character(len=*) , optional :: tmp_file</i>	file with temporal output
out	<i>REAL(DP), optional :: funcbest</i>	minimized value of cost function
out	<i>"REAL(DP),DIMENSION(:,),ALLOCATABLE,LE,optional</i>	<i>:: history"</i> returns a vector of achieved objective

Returns

real(dp) :: parabest(size(para)) — Parameter set minimizing the cost function.

Note

Either fixed parameter range (prange) OR flexible parameter range (function interface prange_func) has to be given in calling sequence.
Only double precision version available.
If single precision is needed not only DP has to be replaced by SP but also I8 of save_state (random number variables) has to be replaced by I4.
ParaChangeMode > 1 is not applied in GetTemperature. For Temperature estimation always only one single parameter is changed (ParaChangeMode=1) which should give theoretically always the best estimate.
Cost and prange_func are user defined functions. See interface definition.

16.2.2 Member Function/Subroutine Documentation

16.2.2.1 anneal_dp()

```
real(dp) function, dimension(size(para,1)) mo_anneal::anneal::anneal_dp (
    cost,
    real(dp), dimension(:), intent(in) para,
    real(dp), dimension(size(para,1),2), intent(in), optional prange,
    optional prange_func,
    real(dp), intent(in), optional temp,
    real(dp), intent(in), optional Dt,
    integer(i4), intent(in), optional nITERmax,
    integer(i4), intent(in), optional Len,
    integer(i4), intent(in), optional nST,
    real(dp), intent(in), optional eps,
    real(dp), intent(in), optional acc,
    integer(i8), dimension(3), intent(in), optional seeds,
    logical, intent(in), optional printflag,
```

```

logical, dimension(size(para,1)), intent(in), optional maskpara,
real(dp), dimension(size(para,1)), intent(in), optional weight,
integer(i4), intent(in), optional changeParaMode,
logical, intent(in), optional reflectionFlag,
logical, intent(in), optional pertubFlexFlag,
logical, intent(in), optional maxit,
real(dp), intent(in), optional undef_funcval,
character(len=*), intent(in), optional tmp_file,
real(dp), intent(out), optional funcbest,
real(dp), dimension(:, :), intent(out), optional, allocatable history )

```

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

- [mo_anneal.f90](#)

16.3 mo_append::append Interface Reference

Append (rows) scalars, vectors, and matrixes onto existing array.

Public Member Functions

- subroutine [append_i4_v_s](#) (vec1, sca2)
- subroutine [append_i4_v_v](#) (vec1, vec2)
- subroutine [append_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_i8_v_s](#) (vec1, sca2)
- subroutine [append_i8_v_v](#) (vec1, vec2)
- subroutine [append_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_i8_3d](#) (mat1, mat2, fill_value)
- subroutine [append_sp_v_s](#) (vec1, sca2)
- subroutine [append_sp_v_v](#) (vec1, vec2)
- subroutine [append_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_sp_3d](#) (mat1, mat2, fill_value)
- subroutine [append_dp_v_s](#) (vec1, sca2)
- subroutine [append_dp_v_v](#) (vec1, vec2)
- subroutine [append_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_dp_3d](#) (mat1, mat2, fill_value)
- subroutine [append_char_v_s](#) (vec1, sca2)
- subroutine [append_char_v_v](#) (vec1, vec2)
- subroutine [append_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_char_3d](#) (mat1, mat2, fill_value)
- subroutine [append_lgt_v_s](#) (vec1, sca2)
- subroutine [append_lgt_v_v](#) (vec1, vec2)
- subroutine [append_lgt_m_m](#) (mat1, mat2, fill_value)
- subroutine [append_lgt_3d](#) (mat1, mat2, fill_value)

16.3.1 Detailed Description

Append (rows) scalars, vectors, and matrixes onto existing array.

Appends one input to the rows of another, i.e. append on the first dimension.

The input might be a scalar, a vector or a matrix.

Possibilities are:

- (1) append scalar to vector
- (2) append vector to vector
- (3) append matrix to matrix

Parameters

in	<i>input2</i>	values to append. Can be INTEGER(I4/I8), REAL(SP/DP), or CHARACTER(len=*) and also scalar, DIMENSION(:), or DIMENSION(:, :) Also includes 3d version, where the append is always done along the first dimension. If not scalar then the columns have to agree with input1.
in, out	<i>allocatable :: input1</i>	array to be appended to. Can be INTEGER(I4/I8), REAL(SP/DP), or CHARACTER(len=*). Must be DIMENSION(:) or DIMENSION(:, :), and allocatable. If input2 is not scalar then it must be size(input1,2) = size(input2,2).

Author

Juliane Mai

Date

Aug 2012

16.3.2 Member Function/Subroutine Documentation

16.3.2.1 append_char_3d()

```
subroutine mo_append::append::append_char_3d (
    character(len=*), dimension(:, :, :), intent(inout), allocatable mat1,
    character(len=*), dimension(:, :, :), intent(in) mat2,
    character(len=*), intent(in), optional fill_value )
```

16.3.2.2 append_char_m_m()

```
subroutine mo_append::append::append_char_m_m (
    character(len=*), dimension(:, :), intent(inout), allocatable mat1,
    character(len=*), dimension(:, :), intent(in) mat2,
    character(len=*), intent(in), optional fill_value )
```

16.3.2.3 append_char_v_s()

```
subroutine mo_append::append::append_char_v_s (
    character(len=*), dimension(:), intent(inout), allocatable vec1,
    character(len=*), intent(in) sca2 )
```

16.3.2.4 append_char_v_v()

```
subroutine mo_append::append::append_char_v_v (
    character(len=*), dimension(:), intent(inout), allocatable vec1,
    character(len=*), dimension(:), intent(in) vec2 )
```

16.3.2.5 append_dp_3d()

```

subroutine mo_append::append::append_dp_3d (
    real(dp), dimension(:,:,:), intent(inout), allocatable mat1,
    real(dp), dimension(:,:,:), intent(in) mat2,
    real(dp), intent(in), optional fill_value )

```

16.3.2.6 append_dp_m_m()

```

subroutine mo_append::append::append_dp_m_m (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    real(dp), intent(in), optional fill_value )

```

16.3.2.7 append_dp_v_s()

```

subroutine mo_append::append::append_dp_v_s (
    real(dp), dimension(:), intent(inout), allocatable vec1,
    real(dp), intent(in) sca2 )

```

16.3.2.8 append_dp_v_v()

```

subroutine mo_append::append::append_dp_v_v (
    real(dp), dimension(:), intent(inout), allocatable vec1,
    real(dp), dimension(:), intent(in) vec2 )

```

16.3.2.9 append_i4_m_m()

```

subroutine mo_append::append::append_i4_m_m (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:,:), intent(in) mat2,
    integer(i4), intent(in), optional fill_value )

```

16.3.2.10 append_i4_v_s()

```

subroutine mo_append::append::append_i4_v_s (
    integer(i4), dimension(:), intent(inout), allocatable vec1,
    integer(i4), intent(in) sca2 )

```

16.3.2.11 append_i4_v_v()

```

subroutine mo_append::append::append_i4_v_v (
    integer(i4), dimension(:), intent(inout), allocatable vec1,
    integer(i4), dimension(:), intent(in) vec2 )

```

16.3.2.12 append_i8_3d()

```
subroutine mo_append::append::append_i8_3d (  
    integer(i8), dimension(:,:,:), intent(inout), allocatable mat1,  
    integer(i8), dimension(:,:,:), intent(in) mat2,  
    integer(i8), intent(in), optional fill_value )
```

16.3.2.13 append_i8_m_m()

```
subroutine mo_append::append::append_i8_m_m (  
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,  
    integer(i8), dimension(:,:), intent(in) mat2,  
    integer(i8), intent(in), optional fill_value )
```

16.3.2.14 append_i8_v_s()

```
subroutine mo_append::append::append_i8_v_s (  
    integer(i8), dimension(:), intent(inout), allocatable vec1,  
    integer(i8), intent(in) sca2 )
```

16.3.2.15 append_i8_v_v()

```
subroutine mo_append::append::append_i8_v_v (  
    integer(i8), dimension(:), intent(inout), allocatable vec1,  
    integer(i8), dimension(:), intent(in) vec2 )
```

16.3.2.16 append_lgt_3d()

```
subroutine mo_append::append::append_lgt_3d (  
    logical, dimension(:,:,:), intent(inout), allocatable mat1,  
    logical, dimension(:,:,:), intent(in) mat2,  
    logical, intent(in), optional fill_value )
```

16.3.2.17 append_lgt_m_m()

```
subroutine mo_append::append::append_lgt_m_m (  
    logical, dimension(:,:), intent(inout), allocatable mat1,  
    logical, dimension(:,:), intent(in) mat2,  
    logical, intent(in), optional fill_value )
```

16.3.2.18 append_lgt_v_s()

```
subroutine mo_append::append::append_lgt_v_s (
    logical, dimension(:), intent(inout), allocatable vec1,
    logical, intent(in) sca2 )
```

16.3.2.19 append_lgt_v_v()

```
subroutine mo_append::append::append_lgt_v_v (
    logical, dimension(:), intent(inout), allocatable vec1,
    logical, dimension(:), intent(in) vec2 )
```

16.3.2.20 append_sp_3d()

```
subroutine mo_append::append::append_sp_3d (
    real(sp), dimension(:,:,:), intent(inout), allocatable mat1,
    real(sp), dimension(:,:,:), intent(in) mat2,
    real(sp), intent(in), optional fill_value )
```

16.3.2.21 append_sp_m_m()

```
subroutine mo_append::append::append_sp_m_m (
    real(sp), dimension(:,:), intent(inout), allocatable mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    real(sp), intent(in), optional fill_value )
```

16.3.2.22 append_sp_v_s()

```
subroutine mo_append::append::append_sp_v_s (
    real(sp), dimension(:), intent(inout), allocatable vec1,
    real(sp), intent(in) sca2 )
```

16.3.2.23 append_sp_v_v()

```
subroutine mo_append::append::append_sp_v_v (
    real(sp), dimension(:), intent(inout), allocatable vec1,
    real(sp), dimension(:), intent(in) vec2 )
```

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

- [mo_append.f90](#)

16.4 mo_corr::arth Interface Reference

Private Member Functions

- `real(sp)` function, dimension(n) [arth_sp](#) (first, increment, n)
- `real(dp)` function, dimension(n) [arth_dp](#) (first, increment, n)
- `integer(i4)` function, dimension(n) [arth_i4](#) (first, increment, n)

16.4.1 Member Function/Subroutine Documentation

16.4.1.1 arth_dp()

```
real(dp) function, dimension(n) mo_corr::arth::arth_dp (  
    real(dp), intent(in) first,  
    real(dp), intent(in) increment,  
    integer(i4), intent(in) n ) [private]
```

16.4.1.2 arth_i4()

```
integer(i4) function, dimension(n) mo_corr::arth::arth_i4 (  
    integer(i4), intent(in) first,  
    integer(i4), intent(in) increment,  
    integer(i4), intent(in) n ) [private]
```

16.4.1.3 arth_sp()

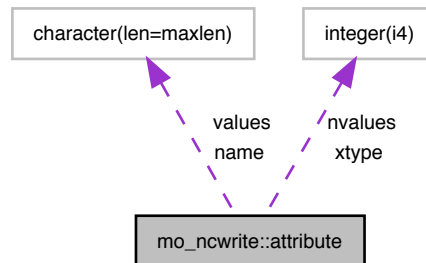
```
real(sp) function, dimension(n) mo_corr::arth::arth_sp (  
    real(sp), intent(in) first,  
    real(sp), intent(in) increment,  
    integer(i4), intent(in) n ) [private]
```

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

- [mo_corr.f90](#)

16.5 mo_ncwrite::attribute Type Reference

Collaboration diagram for mo_ncwrite::attribute:



Public Attributes

- character(len=[maxlen](#)) [name](#)
- integer(i4) [xtype](#)
- integer(i4) [nvalues](#)
- character(len=[maxlen](#)) [values](#)

16.5.1 Member Data Documentation

16.5.1.1 name

```
character (len=maxlen) mo_ncwrite::attribute::name
```

16.5.1.2 nvalues

```
integer(i4) mo_ncwrite::attribute::nvalues
```

16.5.1.3 values

```
character (len=maxlen) mo_ncwrite::attribute::values
```

16.5.1.4 xtype

```
integer(i4) mo_ncwrite::attribute::xtype
```

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

- [mo_ncwrite.f90](#)

16.6 mo_corr::autocoeffk Interface Reference

Public Member Functions

- real(sp) function [autocoeffk_sp](#) (x, k, mask)
- real(dp) function [autocoeffk_dp](#) (x, k, mask)
- real(dp) function, dimension(size(k)) [autocoeffk_1d_dp](#) (x, k, mask)
- real(sp) function, dimension(size(k)) [autocoeffk_1d_sp](#) (x, k, mask)

16.6.1 Member Function/Subroutine Documentation

16.6.1.1 autocoeffk_1d_dp()

```
real(dp) function, dimension(size(k)) mo_corr::autocoeffk::autocoeffk_1d_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.6.1.2 autocoeffk_1d_sp()

```
real(sp) function, dimension(size(k)) mo_corr::autocoeffk::autocoeffk_1d_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.6.1.3 autocoeffk_dp()

```
real(dp) function mo_corr::autocoeffk::autocoeffk_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.6.1.4 autocoeffk_sp()

```
real(sp) function mo_corr::autocoeffk::autocoeffk_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_corr.f90](#)

16.7 mo_corr::autocorr Interface Reference

Public Member Functions

- real(sp) function [autocorr_sp](#) (x, k, mask)
- real(dp) function [autocorr_dp](#) (x, k, mask)
- real(sp) function, dimension(size(k)) [autocorr_1d_sp](#) (x, k, mask)
- real(dp) function, dimension(size(k)) [autocorr_1d_dp](#) (x, k, mask)

16.7.1 Member Function/Subroutine Documentation

16.7.1.1 autocorr_1d_dp()

```
real(dp) function, dimension(size(k)) mo_corr::autocorr::autocorr_1d_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.7.1.2 autocorr_1d_sp()

```
real(sp) function, dimension(size(k)) mo_corr::autocorr::autocorr_1d_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.7.1.3 autocorr_dp()

```
real(dp) function mo_corr::autocorr::autocorr_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.7.1.4 autocorr_sp()

```
real(sp) function mo_corr::autocorr::autocorr_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_corr.f90](#)

16.8 mo_moment::average Interface Reference

Public Member Functions

- real(sp) function [average_sp](#) (dat, mask)
- real(dp) function [average_dp](#) (dat, mask)

16.8.1 Member Function/Subroutine Documentation

16.8.1.1 average_dp()

```
real(dp) function mo_moment::average::average_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

16.8.1.2 average_sp()

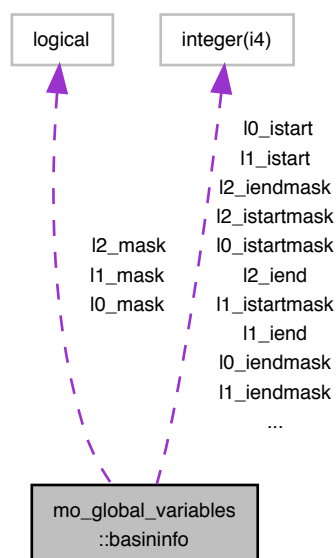
```
real(sp) function mo_moment::average::average_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.9 mo_global_variables::basininfo Type Reference

Collaboration diagram for mo_global_variables::basininfo:



Public Attributes

- integer(i4), dimension(:), allocatable [l0_istart](#)
- integer(i4), dimension(:), allocatable [l0_iend](#)
- integer(i4), dimension(:), allocatable [l0_istartmask](#)
- integer(i4), dimension(:), allocatable [l0_iendmask](#)
- logical, dimension(:), pointer [l0_mask](#)
- integer(i4), dimension(:), allocatable [l1_istart](#)
- integer(i4), dimension(:), allocatable [l1_iend](#)
- integer(i4), dimension(:), allocatable [l1_istartmask](#)
- integer(i4), dimension(:), allocatable [l1_iendmask](#)
- logical, dimension(:), allocatable [l1_mask](#)
- integer(i4), dimension(:), allocatable [l2_istart](#)
- integer(i4), dimension(:), allocatable [l2_iend](#)
- integer(i4), dimension(:), allocatable [l2_istartmask](#)
- integer(i4), dimension(:), allocatable [l2_iendmask](#)
- logical, dimension(:), allocatable [l2_mask](#)

16.9.1 Member Data Documentation

16.9.1.1 l0_iend

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l0_iend

16.9.1.2 l0_iendmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l0_iendmask

16.9.1.3 l0_istart

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l0_istart

16.9.1.4 l0_istartmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l0_istartmask

16.9.1.5 l0_mask

logical, dimension(:), pointer mo_global_variables::basininfo::l0_mask

16.9.1.6 l1_iend

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l1_iend

16.9.1.7 l1_iendmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l1_iendmask

16.9.1.8 l1_istart

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l1_istart

16.9.1.9 l1_istartmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l1_istartmask

16.9.1.10 l1_mask

logical, dimension(:), allocatable mo_global_variables::basininfo::l1_mask

16.9.1.11 l2_iend

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l2_iend

16.9.1.12 l2_iendmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l2_iendmask

16.9.1.13 l2_istart

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l2_istart

16.9.1.14 l2_istartmask

integer(i4), dimension(:), allocatable mo_global_variables::basininfo::l2_istartmask

16.9.1.15 l2_mask

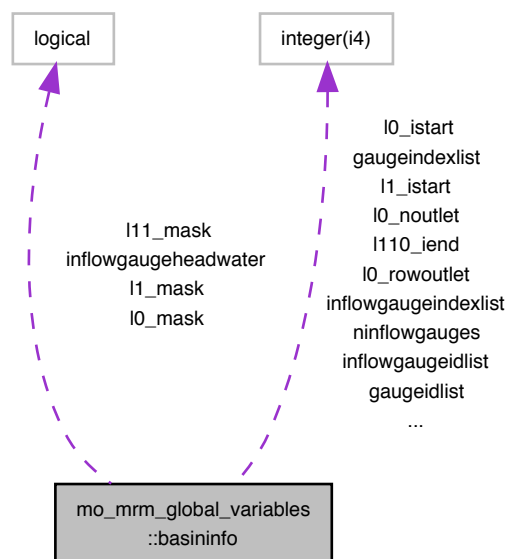
```
logical, dimension(:), allocatable mo_global_variables::basininfo::l2_mask
```

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

- [mo_global_variables.f90](#)

16.10 mo_mrm_global_variables::basininfo Type Reference

Collaboration diagram for mo_mrm_global_variables::basininfo:



Public Attributes

- integer(i4), dimension(:), allocatable [ngauges](#)
- integer(i4), dimension(:, :), allocatable [gaugeidlist](#)
- integer(i4), dimension(:, :), allocatable [gaugeindexlist](#)
- integer(i4), dimension(:, :), allocatable [gaugenodelist](#)
- integer(i4), dimension(:), allocatable [ninflowgauges](#)
- integer(i4), dimension(:, :), allocatable [inflowgaugeidlist](#)
- integer(i4), dimension(:, :), allocatable [inflowgaugeindexlist](#)
- integer(i4), dimension(:, :), allocatable [inflowgaugenodelist](#)
- logical, dimension(:, :), allocatable [inflowgaugeheadwater](#)
- integer(i4), dimension(:), allocatable [l0_noutlet](#)
- integer(i4), dimension(:, :), allocatable [l0_rowoutlet](#)
- integer(i4), dimension(:, :), allocatable [l0_coloutlet](#)
- integer(i4), dimension(:), allocatable [l0_istart](#)
- integer(i4), dimension(:), allocatable [l0_iend](#)
- integer(i4), dimension(:), allocatable [l0_istartmask](#)

- integer(i4), dimension(:), allocatable [l0_iendmask](#)
- logical, dimension(:), pointer [l0_mask](#)
- integer(i4), dimension(:), allocatable [l1_istart](#)
- integer(i4), dimension(:), allocatable [l1_iend](#)
- integer(i4), dimension(:), allocatable [l1_istartmask](#)
- integer(i4), dimension(:), allocatable [l1_iendmask](#)
- logical, dimension(:), allocatable [l1_mask](#)
- integer(i4), dimension(:), allocatable [l11_istart](#)
- integer(i4), dimension(:), allocatable [l11_iend](#)
- integer(i4), dimension(:), allocatable [l11_istartmask](#)
- integer(i4), dimension(:), allocatable [l11_iendmask](#)
- logical, dimension(:), allocatable [l11_mask](#)
- integer(i4), dimension(:), allocatable [l110_istart](#)
- integer(i4), dimension(:), allocatable [l110_iend](#)

16.10.1 Member Data Documentation

16.10.1.1 gaugeidlist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::gaugeidlist

16.10.1.2 gaugeindexlist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::gaugeindexlist

16.10.1.3 gaugenodelist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::gaugenodelist

16.10.1.4 inflowgaugeheadwater

logical, dimension(:,:), allocatable mo_mrm_global_variables::basininfo::inflowgaugeheadwater

16.10.1.5 inflowgaugeidlist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::inflowgaugeidlist

16.10.1.6 inflowgaugeindexlist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::inflowgaugeindexlist

16.10.1.7 inflowgaugenodelist

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::inflowgaugenodelist

16.10.1.8 l0_coloutlet

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::l0_coloutlet

16.10.1.9 l0_iend

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l0_iend

16.10.1.10 l0_iendmask

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l0_iendmask

16.10.1.11 l0_istart

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l0_istart

16.10.1.12 l0_istartmask

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l0_istartmask

16.10.1.13 l0_mask

logical, dimension(:), pointer mo_mrm_global_variables::basininfo::l0_mask

16.10.1.14 l0_noutlet

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l0_noutlet

16.10.1.15 l0_rowoutlet

integer(i4), dimension(:,:), allocatable mo_mrm_global_variables::basininfo::l0_rowoutlet

16.10.1.16 l110_iend

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l110_iend

16.10.1.17 l110_istart

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l110_istart

16.10.1.18 l11_iend

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l11_iend

16.10.1.19 l11_iendmask

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l11_iendmask

16.10.1.20 l11_istart

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l11_istart

16.10.1.21 l11_istartmask

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l11_istartmask

16.10.1.22 l11_mask

logical, dimension(:), allocatable mo_mrm_global_variables::basininfo::l11_mask

16.10.1.23 l1_iend

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l1_iend

16.10.1.24 l1_iendmask

integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l1_iendmask

16.10.1.25 l1_istart

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l1_istart
```

16.10.1.26 l1_istartmask

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::l1_istartmask
```

16.10.1.27 l1_mask

```
logical, dimension(:), allocatable mo_mrm_global_variables::basininfo::l1_mask
```

16.10.1.28 ngauges

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::ngauges
```

16.10.1.29 ninflowgauges

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::basininfo::ninflowgauges
```

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

- [mo_mrm_global_variables.f90](#)

16.11 mo_errormeasures::bias Interface Reference**Public Member Functions**

- real(sp) function [bias_sp_1d](#) (x, y, mask)
- real(dp) function [bias_dp_1d](#) (x, y, mask)
- real(sp) function [bias_sp_2d](#) (x, y, mask)
- real(dp) function [bias_dp_2d](#) (x, y, mask)
- real(sp) function [bias_sp_3d](#) (x, y, mask)
- real(dp) function [bias_dp_3d](#) (x, y, mask)

16.11.1 Member Function/Subroutine Documentation**16.11.1.1 bias_dp_1d()**

```
real(dp) function mo_errormeasures::bias::bias_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.11.1.2 bias_dp_2d()

```
real(dp) function mo_errormeasures::bias::bias_dp_2d (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.11.1.3 bias_dp_3d()

```
real(dp) function mo_errormeasures::bias::bias_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

16.11.1.4 bias_sp_1d()

```
real(sp) function mo_errormeasures::bias::bias_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.11.1.5 bias_sp_2d()

```
real(sp) function mo_errormeasures::bias::bias_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.11.1.6 bias_sp_3d()

```
real(sp) function mo_errormeasures::bias::bias_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.12 mo_moment::central_moment Interface Reference

Public Member Functions

- real(sp) function [central_moment_sp](#) (x, r, mask)
- real(dp) function [central_moment_dp](#) (x, r, mask)

16.12.1 Member Function/Subroutine Documentation

16.12.1.1 `central_moment_dp()`

```
real(dp) function mo_moment::central_moment::central_moment_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), intent(in) r,
    logical, dimension(:), intent(in), optional mask )
```

16.12.1.2 `central_moment_sp()`

```
real(sp) function mo_moment::central_moment::central_moment_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), intent(in) r,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.13 `mo_moment::central_moment_var` Interface Reference

Public Member Functions

- real(sp) function [central_moment_var_sp](#) (x, r, mask)
- real(dp) function [central_moment_var_dp](#) (x, r, mask)

16.13.1 Member Function/Subroutine Documentation

16.13.1.1 `central_moment_var_dp()`

```
real(dp) function mo_moment::central_moment_var::central_moment_var_dp (
    real(dp), dimension(:), intent(in) x,
    integer(i4), intent(in) r,
    logical, dimension(:), intent(in), optional mask )
```

16.13.1.2 `central_moment_var_sp()`

```
real(sp) function mo_moment::central_moment_var::central_moment_var_sp (
    real(sp), dimension(:), intent(in) x,
    integer(i4), intent(in) r,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.14 mo_standard_score::classified_standard_score Interface Reference

Calculates the classified standard score (e.g. classes are months).

Public Member Functions

- real(sp) function, dimension(size(data, dim=1)) [classified_standard_score_sp](#) (data, classes, mask)
- real(dp) function, dimension(size(data, dim=1)) [classified_standard_score_dp](#) (data, classes, mask)

16.14.1 Detailed Description

Calculates the classified standard score (e.g. classes are months).

In statistics, the standard score is the (signed) number of standard deviations an observation or datum is above the mean. Thus, a positive standard score indicates a datum above the mean, while a negative standard score indicates a datum below the mean. It is a dimensionless quantity obtained by subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. This conversion process is called standardizing or normalizing (however, "normalizing" can refer to many types of ratios).

Standard scores are also called z-values, z-scores, normal scores, and standardized variables; the use of "Z" is because the normal distribution is also known as the "Z distribution". They are most frequently used to compare a sample to a standard normal deviate, though they can be defined without assumptions of normality (Wikipedia, May 2015).

In this particular case the standard score is calculated for means and standard deviations derived from classes of the time series. Such classes could be for example months. Thus, the output would be a deseasonalized time series.

$$classified_standard_score = \frac{x_i - \mu_{c_{x_i}}}{\sigma_{c_{x_i}}}$$

where x_i is an element of class c_{x_i} . x is a population, $\mu_{c_{x_i}}$ is the mean of all members of a class c_{x_i} and $\sigma_{c_{x_i}}$ its standard deviation.

If an optimal mask is given, the calculations are over those locations that correspond to true values in the mask. data can be single or double precision. The result will have the same numerical precision.

Parameters

in	<i>integer, dimension(:) :: classes</i>	classes to categorize data (e.g. months)
in	<i>real(sp/dp), dimension(:) :: data</i>	data to calculate the standard score for
in	<i>logical, dimension(:), optional :: mask</i>	indication which cells to use for calculation If present, only those locations in mask having true values in mask are evaluated.

Returns

real(sp/dp) :: [classified_standard_score](#) — classified standard score (e.g. deseasonalized time series)

Author

Matthias Zink

Date

May 2015

16.14.2 Member Function/Subroutine Documentation

16.14.2.1 `classified_standard_score_dp()`

```
real(dp) function, dimension(size(data, dim=1)) mo_standard_score::classified_standard_score←
::classified_standard_score_dp (
    real(dp), dimension(:), intent(in) data,
    integer, dimension(:), intent(in) classes,
    logical, dimension(:), intent(in), optional mask )
```

16.14.2.2 `classified_standard_score_sp()`

```
real(sp) function, dimension(size(data, dim=1)) mo_standard_score::classified_standard_score←
::classified_standard_score_sp (
    real(sp), dimension(:), intent(in) data,
    integer, dimension(:), intent(in) classes,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_standard_score.f90](#)

16.15 `mo_corr::corr` Interface Reference

Public Member Functions

- `real(sp) function, dimension(size(data1)) corr_sp (data1, data2, nadjust, nhigh, nwin)`
- `real(dp) function, dimension(size(data1)) corr_dp (data1, data2, nadjust, nhigh, nwin)`

16.15.1 Member Function/Subroutine Documentation

16.15.1.1 `corr_dp()`

```
real(dp) function, dimension(size(data1)) mo_corr::corr::corr_dp (
    real(dp), dimension(:), intent(in) data1,
    real(dp), dimension(:), intent(in) data2,
    integer(i4), intent(out), optional nadjust,
    integer(i4), intent(in), optional nhigh,
    integer(i4), intent(in), optional nwin )
```

16.15.1.2 `corr_sp()`

```
real(sp) function, dimension(size(data1)) mo_corr::corr::corr_sp (
    real(sp), dimension(:), intent(in) data1,
    real(sp), dimension(:), intent(in) data2,
    integer(i4), intent(out), optional nadjust,
    integer(i4), intent(in), optional nhigh,
    integer(i4), intent(in), optional nwin )
```

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

- [mo_corr.f90](#)

16.16 mo_moment::correlation Interface Reference

Public Member Functions

- real(sp) function [correlation_sp](#) (x, y, mask)
- real(dp) function [correlation_dp](#) (x, y, mask)

16.16.1 Member Function/Subroutine Documentation

16.16.1.1 correlation_dp()

```
real(dp) function mo_moment::correlation::correlation_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.16.1.2 correlation_sp()

```
real(sp) function mo_moment::correlation::correlation_sp (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.17 mo_moment::covariance Interface Reference

Public Member Functions

- real(sp) function [covariance_sp](#) (x, y, mask)
- real(dp) function [covariance_dp](#) (x, y, mask)

16.17.1 Member Function/Subroutine Documentation

16.17.1.1 covariance_dp()

```
real(dp) function mo_moment::covariance::covariance_dp (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.17.1.2 covariance_sp()

```
real(sp) function mo_moment::covariance::covariance_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.18 mo_corr::crosscoeffk Interface Reference

Public Member Functions

- real(sp) function [crosscoeffk_sp](#) (x, y, k, mask)
- real(dp) function [crosscoeffk_dp](#) (x, y, k, mask)

16.18.1 Member Function/Subroutine Documentation

16.18.1.1 crosscoeffk_dp()

```
real(dp) function mo_corr::crosscoeffk::crosscoeffk_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.18.1.2 crosscoeffk_sp()

```
real(sp) function mo_corr::crosscoeffk::crosscoeffk_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_corr.f90](#)

16.19 mo_corr::crosscorr Interface Reference

Public Member Functions

- real(sp) function [crosscorr_sp](#) (x, y, k, mask)
- real(dp) function [crosscorr_dp](#) (x, y, k, mask)

16.19.1 Member Function/Subroutine Documentation

16.19.1.1 crosscorr_dp()

```
real(dp) function mo_corr::crosscorr::crosscorr_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

16.19.1.2 crosscorr_sp()

```
real(sp) function mo_corr::crosscorr::crosscorr_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    integer(i4), intent(in) k,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_corr.f90](#)

16.20 mo_orderpack::ctrper Interface Reference**Public Member Functions**

- subroutine [d_ctrper](#) (XDONT, PCLS)
- subroutine [r_ctrper](#) (XDONT, PCLS)
- subroutine [i_ctrper](#) (XDONT, PCLS)

16.20.1 Member Function/Subroutine Documentation**16.20.1.1 d_ctrper()**

```
subroutine mo_orderpack::ctrper::d_ctrper (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    real(kind=dp), intent(in) PCLS )
```

16.20.1.2 i_ctrper()

```
subroutine mo_orderpack::ctrper::i_ctrper (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    real(kind=sp), intent(in) PCLS )
```

16.20.1.3 r_ctrper()

```
subroutine mo_orderpack::ctrper::r_ctrper (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    real(kind=sp), intent(in) PCLS )
```

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

- [mo_orderpack.f90](#)

16.21 mo_temporal_aggregation::day2mon_average Interface Reference

Day-to-month average ([day2mon_average](#))

Public Member Functions

- subroutine [day2mon_average_dp](#) (daily_data, yearS, monthS, dayS, mon_avg, misval, rm_misval)

16.21.1 Detailed Description

Day-to-month average ([day2mon_average](#))

converts daily time series to monthly

Parameters

in	<i>real(sp/dp) :: daily_data(:)</i>	array of daily time series
in	<i>integer(i4) :: year</i>	year of the starting time
in	<i>integer(i4) :: month</i>	month of the starting time
in	<i>integer(i4) :: day</i>	day of the starting time
in	<i>real(sp/dp) :: mon_average(:)</i>	array of monthly averaged values
in	<i>real(sp/dp) :: misval</i>	missing value definition
in	<i>logical :: rm_misval</i>	switch to exclude missing values

Note

Author

Oldrich Rakovec, Rohini Kumar

Date

Oct 2015

16.21.2 Member Function/Subroutine Documentation

16.21.2.1 day2mon_average_dp()

```
subroutine mo_temporal_aggregation::day2mon_average::day2mon_average_dp (
    real(dp), dimension(:), intent(in) daily_data,
    integer(i4), intent(in) yearS,
    integer(i4), intent(in) monthS,
```

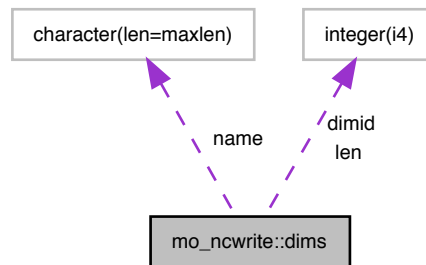
```
integer(i4), intent(in) dayS,
real(dp), dimension(:), intent(inout), allocatable mon_avg,
real(dp), intent(in), optional misval,
logical, intent(in), optional rm_misval )
```

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

- [mo_temporal_aggregation.f90](#)

16.22 mo_ncwrite::dims Type Reference

Collaboration diagram for mo_ncwrite::dims:



Public Attributes

- `character(len=maxlen)` [name](#)
- `integer(i4)` [len](#)
- `integer(i4)` [dimid](#)

16.22.1 Member Data Documentation

16.22.1.1 dimid

```
integer(i4) mo_ncwrite::dims::dimid
```

16.22.1.2 len

```
integer(i4) mo_ncwrite::dims::len
```

16.22.1.3 name

```
character (len=maxlen) mo_ncwrite::dims::name
```

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

- [mo_ncwrite.f90](#)

16.23 mo_ncwrite::dump_netcdf Interface Reference

Public Member Functions

- subroutine [dump_netcdf_1d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_1d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_1d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_2d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_3d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_4d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [dump_netcdf_5d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)

16.23.1 Member Function/Subroutine Documentation

16.23.1.1 dump_netcdf_1d_dp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_1d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.2 dump_netcdf_1d_i4()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_1d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.3 dump_netcdf_1d_sp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_1d_sp (  
    character(len=*), intent(in) filename,  
    real(sp), dimension(:), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.4 dump_netcdf_2d_dp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_2d_dp (  
    character(len=*), intent(in) filename,  
    real(dp), dimension(:, :), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.5 dump_netcdf_2d_i4()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_2d_i4 (  
    character(len=*), intent(in) filename,  
    integer(i4), dimension(:, :), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.6 dump_netcdf_2d_sp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_2d_sp (  
    character(len=*), intent(in) filename,  
    real(sp), dimension(:, :), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.7 dump_netcdf_3d_dp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_3d_dp (  
    character(len=*), intent(in) filename,  
    real(dp), dimension(:, :, :), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.8 dump_netcdf_3d_i4()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_3d_i4 (  
    character(len=*), intent(in) filename,  
    integer(i4), dimension(:,:,:), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.9 dump_netcdf_3d_sp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_3d_sp (  
    character(len=*), intent(in) filename,  
    real(sp), dimension(:,:,:), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.10 dump_netcdf_4d_dp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_4d_dp (  
    character(len=*), intent(in) filename,  
    real(dp), dimension(:,:,:,:), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.11 dump_netcdf_4d_i4()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_4d_i4 (  
    character(len=*), intent(in) filename,  
    integer(i4), dimension(:,:,:,:), intent(in) arr,  
    logical, intent(in), optional append,  
    logical, intent(in), optional lfs,  
    logical, intent(in), optional netcdf4,  
    integer(i4), intent(in), optional deflate_level )
```

16.23.1.12 dump_netcdf_4d_sp()

```
subroutine mo_ncwrite::dump_netcdf::dump_netcdf_4d_sp (  
    character(len=*), intent(in) filename,  
    real(sp), dimension(:,:,:,:), intent(in) arr,  
    logical, intent(in), optional append,
```

```

logical, intent(in), optional lfs,
logical, intent(in), optional netcdf4,
integer(i4), intent(in), optional deflate_level )

```

16.23.1.13 dump_netcdf_5d_dp()

```

subroutine mo_ncwrite::dump_netcdf::dump_netcdf_5d_dp (
    character(len=*), intent(in) filename,
    real(dp), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level )

```

16.23.1.14 dump_netcdf_5d_i4()

```

subroutine mo_ncwrite::dump_netcdf::dump_netcdf_5d_i4 (
    character(len=*), intent(in) filename,
    integer(i4), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level )

```

16.23.1.15 dump_netcdf_5d_sp()

```

subroutine mo_ncwrite::dump_netcdf::dump_netcdf_5d_sp (
    character(len=*), intent(in) filename,
    real(sp), dimension(:,:,:,,:), intent(in) arr,
    logical, intent(in), optional append,
    logical, intent(in), optional lfs,
    logical, intent(in), optional netcdf4,
    integer(i4), intent(in), optional deflate_level )

```

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

- [mo_ncwrite.f90](#)

16.24 mo_utils::eq Interface Reference

Public Member Functions

- elemental pure logical function [equal_sp](#) (a, b)
- elemental pure logical function [equal_dp](#) (a, b)

16.24.1 Member Function/Subroutine Documentation

16.24.1.1 equal_dp()

```
elemental pure logical function mo_utils::eq::equal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.24.1.2 equal_sp()

```
elemental pure logical function mo_utils::eq::equal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.25 mo_utils::equal Interface Reference

Comparison of real values.

Public Member Functions

- elemental pure logical function [equal_sp](#) (a, b)
- elemental pure logical function [equal_dp](#) (a, b)

16.25.1 Detailed Description

Comparison of real values.

Compares two reals if they are numerically equal or not, i.e. equal:

$$\left| \frac{a-b}{b} \right| < \epsilon$$

Parameters

in	<i>real(sp/dp) :: a</i>	First number to compare
in	<i>real(sp/dp) :: b</i>	Second number to compare

Returns

real(sp/dp) :: equal — $a == b$ logically true or false

Authors

Matthias Cuntz, Juliane Mai

Date

Feb 2014

16.25.2 Member Function/Subroutine Documentation

16.25.2.1 equal_dp()

```
elemental pure logical function mo_utils::equal::equal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.25.2.2 equal_sp()

```
elemental pure logical function mo_utils::equal::equal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.26 mo_orderpack::fndnth Interface Reference

Public Member Functions

- real(kind=dp) function [d_fndnth](#) (XDONT, NORD)
- real(kind=sp) function [r_fndnth](#) (XDONT, NORD)
- integer(kind=i4) function [i_fndnth](#) (XDONT, NORD)

16.26.1 Member Function/Subroutine Documentation

16.26.1.1 d_fndnth()

```
real(kind=dp) function mo_orderpack::fndnth::d_fndnth (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.26.1.2 i_fndnth()

```
integer(kind=i4) function mo_orderpack::fndnth::i_fndnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.26.1.3 `r_fndnth()`

```
real(kind=sp) function mo_orderpack::fndnth::r_fndnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.27 `mo_corr::four1` Interface Reference

Private Member Functions

- subroutine [four1_sp](#) (data, isign)
- subroutine [four1_dp](#) (data, isign)

16.27.1 Member Function/Subroutine Documentation

16.27.1.1 `four1_dp()`

```
subroutine mo_corr::four1::four1_dp (
    complex(dpc), dimension (:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

16.27.1.2 `four1_sp()`

```
subroutine mo_corr::four1::four1_sp (
    complex(spc), dimension (:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]
```

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

- [mo_corr.f90](#)

16.28 `mo_corr::fourrow` Interface Reference

Private Member Functions

- subroutine [fourrow_sp](#) (data, isign)
- subroutine [fourrow_dp](#) (data, isign)

16.28.1 Member Function/Subroutine Documentation

16.28.1.1 fourrow_dp()

```

subroutine mo_corr::fourrow::fourrow_dp (
    complex(dp), dimension(:,:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]

```

16.28.1.2 fourrow_sp()

```

subroutine mo_corr::fourrow::fourrow_sp (
    complex(spc), dimension(:,:), intent(inout) data,
    integer(i4), intent(in) isign ) [private]

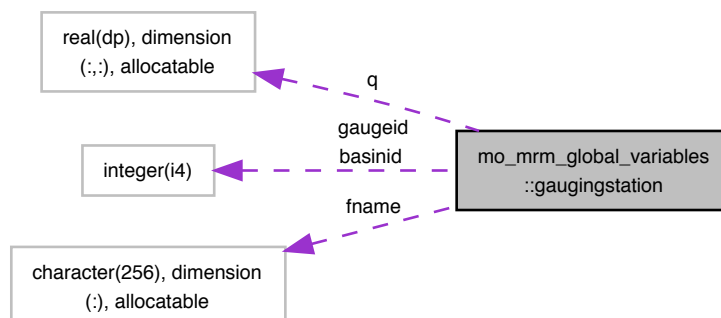
```

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

- [mo_corr.f90](#)

16.29 mo_mrm_global_variables::gaugingstation Type Reference

Collaboration diagram for mo_mrm_global_variables::gaugingstation:



Public Attributes

- integer(i4), dimension(:), allocatable [basinid](#)
- integer(i4), dimension(:), allocatable [gaugeid](#)
- character(256), dimension(:), allocatable [fname](#)
- real(dp), dimension(:,:), allocatable [q](#)

16.29.1 Member Data Documentation

16.29.1.1 basinid

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::gaugingstation::basinid
```

16.29.1.2 fname

```
character(256), dimension(:), allocatable mo_mrm_global_variables::gaugingstation::fname
```

16.29.1.3 gaugeid

```
integer(i4), dimension(:), allocatable mo_mrm_global_variables::gaugingstation::gaugeid
```

16.29.1.4 q

```
real(dp), dimension(:, :), allocatable mo_mrm_global_variables::gaugingstation::q
```

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

- [mo_mrm_global_variables.f90](#)

16.30 mo_utils::ge Interface Reference**Public Member Functions**

- elemental pure logical function [greater_equal_sp](#) (a, b)
- elemental pure logical function [greater_equal_dp](#) (a, b)

16.30.1 Member Function/Subroutine Documentation**16.30.1.1 greater_equal_dp()**

```
elemental pure logical function mo_utils::ge::greater_equal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.30.1.2 greater_equal_sp()

```
elemental pure logical function mo_utils::ge::greater_equal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.31 mo_anneal::generate_neighborhood_weight Interface Reference

Private Member Functions

- subroutine [generate_neighborhood_weight_dp](#) (truepara, cum_weight, save_state_xor, iTotalCounter, nIT←ERmax, neighborhood)

16.31.1 Member Function/Subroutine Documentation

16.31.1.1 generate_neighborhood_weight_dp()

```
subroutine mo_anneal::generate_neighborhood_weight::generate_neighborhood_weight_dp (
    integer(i4), dimension(:), intent(in) truepara,
    real(dp), dimension(:), intent(in) cum_weight,
    integer(i8), dimension(n_save_state), intent(inout) save_state_xor,
    integer(i4), intent(in) iTotalCounter,
    integer(i4), intent(in) nITERmax,
    logical, dimension(size(cum_weight)), intent(out) neighborhood ) [private]
```

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

- [mo_anneal.f90](#)

16.32 mo_ncread::get_ncvar Interface Reference

Public Member Functions

- subroutine [get_ncvar_0d_sp](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_0d_dp](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_1d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_0d_i4](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_0d_i1](#) (Filename, VarName, Dat, fid)
- subroutine [get_ncvar_1d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_2d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_3d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_4d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [get_ncvar_5d_i1](#) (Filename, VarName, Dat, start, a_count, fid)

16.32.1 Member Function/Subroutine Documentation

16.32.1.1 `get_ncvar_0d_dp()`

```
subroutine mo_ncread::get_ncvar::get_ncvar_0d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), intent(inout) Dat,
    integer(i4), intent(in), optional fid )
```

16.32.1.2 `get_ncvar_0d_i1()`

```
subroutine mo_ncread::get_ncvar::get_ncvar_0d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(l), intent(inout) Dat,
    integer(i4), intent(in), optional fid )
```

16.32.1.3 `get_ncvar_0d_i4()`

```
subroutine mo_ncread::get_ncvar::get_ncvar_0d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), intent(inout) Dat,
    integer(i4), intent(in), optional fid )
```

16.32.1.4 `get_ncvar_0d_sp()`

```
subroutine mo_ncread::get_ncvar::get_ncvar_0d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), intent(inout) Dat,
    integer(i4), intent(in), optional fid )
```

16.32.1.5 `get_ncvar_1d_dp()`

```
subroutine mo_ncread::get_ncvar::get_ncvar_1d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )
```

16.32.1.6 get_ncvar_1d_i1()

```

subroutine mo_ncread::get_ncvar::get_ncvar_1d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.7 get_ncvar_1d_i4()

```

subroutine mo_ncread::get_ncvar::get_ncvar_1d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.8 get_ncvar_1d_sp()

```

subroutine mo_ncread::get_ncvar::get_ncvar_1d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.9 get_ncvar_2d_dp()

```

subroutine mo_ncread::get_ncvar::get_ncvar_2d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.10 get_ncvar_2d_i1()

```

subroutine mo_ncread::get_ncvar::get_ncvar_2d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.11 get_ncvar_2d_i4()

```

subroutine mo_ncread::get_ncvar::get_ncvar_2d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.12 get_ncvar_2d_sp()

```

subroutine mo_ncread::get_ncvar::get_ncvar_2d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.13 get_ncvar_3d_dp()

```

subroutine mo_ncread::get_ncvar::get_ncvar_3d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:, :, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.14 get_ncvar_3d_i1()

```

subroutine mo_ncread::get_ncvar::get_ncvar_3d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:, :, :), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.15 get_ncvar_3d_i4()

```

subroutine mo_ncread::get_ncvar::get_ncvar_3d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:, :, :), intent(inout), allocatable Dat,

```

```
integer(i4), dimension(:), intent(in), optional start,
integer(i4), dimension(:), intent(in), optional a_count,
integer(i4), intent(in), optional fid )
```

16.32.1.16 get_ncvar_3d_sp()

```
subroutine mo_ncread::get_ncvar::get_ncvar_3d_sp (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  real(sp), dimension(:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid )
```

16.32.1.17 get_ncvar_4d_dp()

```
subroutine mo_ncread::get_ncvar::get_ncvar_4d_dp (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  real(dp), dimension(:,:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid )
```

16.32.1.18 get_ncvar_4d_i1()

```
subroutine mo_ncread::get_ncvar::get_ncvar_4d_i1 (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  integer(1), dimension(:,:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid )
```

16.32.1.19 get_ncvar_4d_i4()

```
subroutine mo_ncread::get_ncvar::get_ncvar_4d_i4 (
  character(len=*), intent(in) Filename,
  character(len=*), intent(in) VarName,
  integer(i4), dimension(:,:,:,:), intent(inout), allocatable Dat,
  integer(i4), dimension(:), intent(in), optional start,
  integer(i4), dimension(:), intent(in), optional a_count,
  integer(i4), intent(in), optional fid )
```

16.32.1.20 get_ncvar_4d_sp()

```
subroutine mo_ncread::get_ncvar::get_ncvar_4d_sp (
```

```

character(len=*), intent(in) Filename,
character(len=*), intent(in) VarName,
real(sp), dimension(:,:,:), intent(inout), allocatable Dat,
integer(i4), dimension(:), intent(in), optional start,
integer(i4), dimension(:), intent(in), optional a_count,
integer(i4), intent(in), optional fid )

```

16.32.1.21 `get_ncvar_5d_dp()`

```

subroutine mo_ncread::get_ncvar::get_ncvar_5d_dp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(dp), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.22 `get_ncvar_5d_i1()`

```

subroutine mo_ncread::get_ncvar::get_ncvar_5d_i1 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(1), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.23 `get_ncvar_5d_i4()`

```

subroutine mo_ncread::get_ncvar::get_ncvar_5d_i4 (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    integer(i4), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

16.32.1.24 `get_ncvar_5d_sp()`

```

subroutine mo_ncread::get_ncvar::get_ncvar_5d_sp (
    character(len=*), intent(in) Filename,
    character(len=*), intent(in) VarName,
    real(sp), dimension(:,:,:), intent(inout), allocatable Dat,
    integer(i4), dimension(:), intent(in), optional start,
    integer(i4), dimension(:), intent(in), optional a_count,
    integer(i4), intent(in), optional fid )

```

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

- [mo_ncread.f90](#)

16.33 mo_xor4096::get_timeseed Interface Reference

Public Member Functions

- subroutine [get_timeseed_i4_0d](#) (seed)
- subroutine [get_timeseed_i4_1d](#) (seed)
- subroutine [get_timeseed_i8_0d](#) (seed)
- subroutine [get_timeseed_i8_1d](#) (seed)

16.33.1 Member Function/Subroutine Documentation

16.33.1.1 [get_timeseed_i4_0d\(\)](#)

```
subroutine mo_xor4096::get_timeseed::get_timeseed_i4_0d (
    integer(i4), intent(inout) seed )
```

16.33.1.2 [get_timeseed_i4_1d\(\)](#)

```
subroutine mo_xor4096::get_timeseed::get_timeseed_i4_1d (
    integer(i4), dimension(:), intent(inout) seed )
```

16.33.1.3 [get_timeseed_i8_0d\(\)](#)

```
subroutine mo_xor4096::get_timeseed::get_timeseed_i8_0d (
    integer(i8), intent(inout) seed )
```

16.33.1.4 [get_timeseed_i8_1d\(\)](#)

```
subroutine mo_xor4096::get_timeseed::get_timeseed_i8_1d (
    integer(i8), dimension(:), intent(inout) seed )
```

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

- [mo_xor4096.f90](#)

16.34 mo_anneal::gettemperature Interface Reference

GetTemperature.

Public Member Functions

- real(dp) function [gettemperature_dp](#) (paraset, cost, acc_goal, prange, prange_func, samplesize, maskpara, seeds, printflag, weight, maxit, undef_funcval)

16.34.1 Detailed Description

GetTemperature.

Determines an initial temperature for Simulated Annealing achieving

Parameters

in	<i>REAL(DP), DIMENSION(:) :: paraset</i>	initial (valid) parameter set
in	<i>INTERFACE :: cost_dp</i>	interface calculating the cost function at a given point
in	<i>REAL(DP) :: acc_goal</i>	Acceptance Ratio which has to be achieved
in	<i>REAL(DP), DIMENSION(size(para),2), optional :: prange</i>	lower and upper bound per parameter
in	<i>INTERFACE, optional :: prange_func</i>	interface calculating the feasible range for a parameter at a certain point, if ranges are variable
in	<i>INTEGER(I4), optional :: samplesize</i>	number of iterations the estimation of temperature is based on DEFAULT: Max(20_i4*n,250_i4)
in	<i>LOGICAL, DIMENSION(size(para)), optional :: maskpara</i>	maskpara(i) = .true. -> parameter is optimized maskpara(i) = .false. -> parameter is discarded from optimiztaion DEFAULT: .true.
in	<i>INTEGER(I4/I8), DIMENSION(2), optional :: seeds</i>	Seeds of random numbers used for random parameter set generation DEFAULT: dependent on current time
in	<i>LOGICAL, optional :: printflag</i>	If .true. detailed command line output is written DEFAULT: .false.
in	<i>REAL(DP), DIMENSION(size(para,1)), optional :: weight</i>	vector of weights per parameter gives the frequency of parameter to be chosen for optimization (will be scaled to a CDF internally) eg. [1,2,1] -> parameter 2 is chosen twice as often as parameter 1 and 2
in	<i>LOGICAL, optional :: maxit</i>	minimizing (.false.) or maximizing (.true.) a function DEFAULT: .false. (minimization)
in	<i>REAL(DP), optional :: undef_funcval</i>	objective function value defining invalid model output, e.g. -999.0_dp

Returns

real(dp) :: temperature — Temperature achieving a certain acceptance ratio in Simulated Annealing

Note

Either fixed parameter range (prange) OR flexible parameter range (function interface prange_func) has to be given in calling sequence.

Only double precision version available. If single precision is needed not only DP has to be replaced by SP but also I8 of save_state (random number variables) has to be replaced by I4.

ParaChangeMode > 1 is not applied in GetTemperature. For Temperature estimation always only one single parameter is changed (ParaChangeMode=1) which should give theoretically always the best estimate.

Cost and prange_func are user defined functions. See interface definition.

16.34.2 Member Function/Subroutine Documentation

16.34.2.1 gettemperature_dp()

```
real(dp) function mo_anneal::gettemperature::gettemperature_dp (
    real(dp), dimension(:), intent(in) paraset,
    cost,
    real(dp), intent(in) acc_goal,
    real(dp), dimension(size(paraset,1),2), intent(in), optional prange,
    optional prange_func,
    integer(i4), intent(in), optional samplesize,
    logical, dimension(size(paraset,1)), intent(in), optional maskpara,
    integer(i8), dimension(2), intent(in), optional seeds,
    logical, intent(in), optional printflag,
    real(dp), dimension(size(paraset,1)), intent(in), optional weight,
    logical, intent(in), optional maxit,
    real(dp), intent(in), optional undef_funcval )
```

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

- [mo_anneal.f90](#)

16.35 mo_utils::greaterequal Interface Reference

Public Member Functions

- elemental pure logical function [greaterequal_sp](#) (a, b)
- elemental pure logical function [greaterequal_dp](#) (a, b)

16.35.1 Member Function/Subroutine Documentation

16.35.1.1 greaterequal_dp()

```
elemental pure logical function mo_utils::greaterequal::greaterequal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.35.1.2 greaterequal_sp()

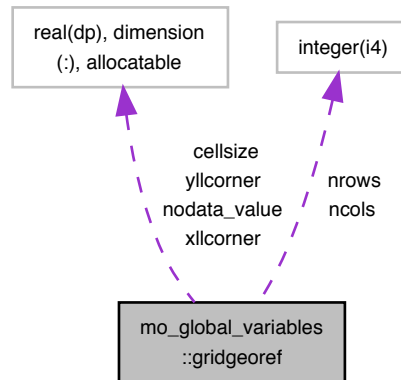
```
elemental pure logical function mo_utils::greaterequal::greaterequal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.36 mo_global_variables::gridgeoref Type Reference

Collaboration diagram for mo_global_variables::gridgeoref:



Public Attributes

- integer(i4), dimension(:), allocatable [ncols](#)
- integer(i4), dimension(:), allocatable [nrows](#)
- real(dp), dimension(:), allocatable [xllcorner](#)
- real(dp), dimension(:), allocatable [yllcorner](#)
- real(dp), dimension(:), allocatable [cellsize](#)
- real(dp), dimension(:), allocatable [nodata_value](#)

16.36.1 Member Data Documentation

16.36.1.1 cellsize

`real(dp), dimension(:), allocatable mo_global_variables::gridgeoref::cellsize`

16.36.1.2 ncols

`integer(i4), dimension(:), allocatable mo_global_variables::gridgeoref::ncols`

16.36.1.3 nodata_value

`real(dp), dimension(:), allocatable mo_global_variables::gridgeoref::nodata_value`

16.36.1.4 nrows

```
integer(i4), dimension(:), allocatable mo_global_variables::gridgeoref::nrows
```

16.36.1.5 xllcorner

```
real(dp), dimension(:), allocatable mo_global_variables::gridgeoref::xllcorner
```

16.36.1.6 yllcorner

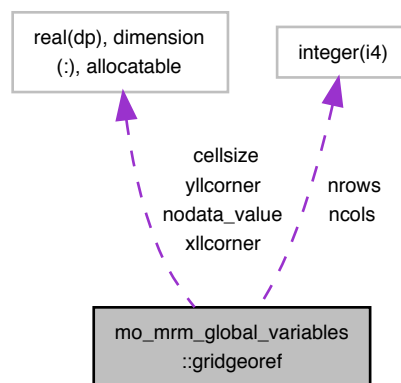
```
real(dp), dimension(:), allocatable mo_global_variables::gridgeoref::yllcorner
```

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

- [mo_global_variables.f90](#)

16.37 mo_mrm_global_variables::gridgeoref Type Reference

Collaboration diagram for mo_mrm_global_variables::gridgeoref:



Public Attributes

- integer(i4), dimension(:), allocatable [ncols](#)
- integer(i4), dimension(:), allocatable [nrows](#)
- real(dp), dimension(:), allocatable [xllcorner](#)
- real(dp), dimension(:), allocatable [yllcorner](#)
- real(dp), dimension(:), allocatable [cellsize](#)
- real(dp), dimension(:), allocatable [nodata_value](#)

16.37.1 Member Data Documentation

16.37.1.1 cellsize

`real(dp), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::cellsize`

16.37.1.2 ncols

`integer(i4), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::ncols`

16.37.1.3 nodata_value

`real(dp), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::nodata_value`

16.37.1.4 nrows

`integer(i4), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::nrows`

16.37.1.5 xllcorner

`real(dp), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::xllcorner`

16.37.1.6 yllcorner

`real(dp), dimension(:), allocatable mo_mrm_global_variables::gridgeoref::yllcorner`

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

- [mo_mrm_global_variables.f90](#)

16.38 mo_temporal_aggregation::hour2day_average Interface Reference

Hour-to-day average ([hour2day_average](#))

Public Member Functions

- subroutine [hour2day_average_dp](#) (hourly_data, yearS, monthS, dayS, hourS, day_avg, misval, rm_misval)

16.38.1 Detailed Description

Hour-to-day average ([hour2day_average](#))

converts hourly time series to daily

Parameters

in	<i>real(sp/dp) :: hourly_data(:)</i>	array of hourly time series
in	<i>integer(i4) :: year</i>	year of the starting time
in	<i>integer(i4) :: month</i>	month of the starting time
in	<i>integer(i4) :: day</i>	day of the starting time
in	<i>integer(i4) :: hour</i>	hour of the starting time
in	<i>real(sp/dp) :: day_average(:)</i>	array of daily averaged values
in	<i>real(sp/dp) :: misval</i>	missing value definition
in	<i>logical :: rm_misval</i>	switch to exclude missing values

Note

Hours values should be from 0 to 23 (NOT from 1 to 24!)

Author

Oldrich Rakovec, Rohini Kumar

Date

Oct 2015

16.38.2 Member Function/Subroutine Documentation

16.38.2.1 hour2day_average_dp()

```
subroutine mo_temporal_aggregation::hour2day_average::hour2day_average_dp (
    real(dp), dimension(:), intent(in) hourly_data,
    integer(i4), intent(in) yearS,
    integer(i4), intent(in) monthS,
    integer(i4), intent(in) dayS,
    integer(i4), intent(in) hourS,
    real(dp), dimension(:), intent(inout), allocatable day_avg,
    real(dp), intent(in), optional misval,
    logical, intent(in), optional rm_misval )
```

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

- [mo_temporal_aggregation.f90](#)

16.39 mo_orderpack::indmed Interface Reference

Public Member Functions

- subroutine [d_indmed](#) (XDONT, INDM)

- subroutine [r_indmed](#) (XDONT, INDM)
- subroutine [i_indmed](#) (XDONT, INDM)

16.39.1 Member Function/Subroutine Documentation

16.39.1.1 d_indmed()

```
subroutine mo_orderpack::indmed::d_indmed (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM )
```

16.39.1.2 i_indmed()

```
subroutine mo_orderpack::indmed::i_indmed (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM )
```

16.39.1.3 r_indmed()

```
subroutine mo_orderpack::indmed::r_indmed (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(out) INDM )
```

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

- [mo_orderpack.f90](#)

16.40 mo_orderpack::indnth Interface Reference

Public Member Functions

- integer(kind=i4) function [d_indnth](#) (XDONT, NORD)
- integer(kind=i4) function [r_indnth](#) (XDONT, NORD)
- integer(kind=i4) function [i_indnth](#) (XDONT, NORD)

16.40.1 Member Function/Subroutine Documentation

16.40.1.1 d_indnth()

```
integer(kind=i4) function mo_orderpack::indnth::d_indnth (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.40.1.2 i_indnth()

```
integer(kind=i4) function mo_orderpack::indnth::i_indnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.40.1.3 r_indnth()

```
integer(kind=i4) function mo_orderpack::indnth::r_indnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.41 mo_orderpack::inspar Interface Reference**Public Member Functions**

- subroutine [d_inspar](#) (XDONT, NORD)
- subroutine [r_inspar](#) (XDONT, NORD)
- subroutine [i_inspar](#) (XDONT, NORD)

16.41.1 Member Function/Subroutine Documentation**16.41.1.1 d_inspar()**

```
subroutine mo_orderpack::inspar::d_inspar (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.41.1.2 i_inspar()

```
subroutine mo_orderpack::inspar::i_inspar (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.41.1.3 r_inspar()

```
subroutine mo_orderpack::inspar::r_inspar (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.42 mo_orderpack::inssor Interface Reference

Public Member Functions

- subroutine [d_inssor](#) (XDONT)
- subroutine [r_inssor](#) (XDONT)
- subroutine [i_inssor](#) (XDONT)

16.42.1 Member Function/Subroutine Documentation

16.42.1.1 d_inssor()

```
subroutine mo_orderpack::inssor::d_inssor (
    real(kind=dp), dimension (:), intent(inout) XDONT )
```

16.42.1.2 i_inssor()

```
subroutine mo_orderpack::inssor::i_inssor (
    integer(kind=i4), dimension (:), intent(inout) XDONT )
```

16.42.1.3 r_inssor()

```
subroutine mo_orderpack::inssor::r_inssor (
    real(kind=sp), dimension (:), intent(inout) XDONT )
```

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

- [mo_orderpack.f90](#)

16.43 mo_utils::is_finite Interface Reference

.true. if not IEEE Inf, IEEE NaN, nor IEEE Inf nor IEEE NaN, respectively.

Public Member Functions

- elemental pure logical function [is_finite_sp](#) (a)
- elemental pure logical function [is_finite_dp](#) (a)

16.43.1 Detailed Description

.true. if not IEEE Inf, IEEE NaN, nor IEEE Inf nor IEEE NaN, respectively.

Checks for IEEE Inf and IEEE NaN, i.e. Infinity and Not-a-Number.

Wraps to functions of the intrinsic module ieee_arithmetic but gives alternatives for gfortran, which does not provide ieee_arithmetic.

Parameters

in	<i>real(sp/dp) :: x</i>	Number to check
----	-------------------------	-----------------

Returns

logical :: is_finite/is_nan/is_normal — $a/ = Inf, a == NaN, a/ = Inf$ and $a == NaN$, logically true or false

Authors

Matthias Cuntz

Date

Mar 2015

16.43.2 Member Function/Subroutine Documentation

16.43.2.1 is_finite_dp()

```
elemental pure logical function mo_utils::is_finite::is_finite_dp (
    real(dp), intent(in) a )
```

16.43.2.2 is_finite_sp()

```
elemental pure logical function mo_utils::is_finite::is_finite_sp (
    real(sp), intent(in) a )
```

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

- [mo_utils.f90](#)

16.44 mo_utils::is_nan Interface Reference

Public Member Functions

- elemental pure logical function [is_nan_sp](#) (a)
- elemental pure logical function [is_nan_dp](#) (a)

16.44.1 Member Function/Subroutine Documentation

16.44.1.1 is_nan_dp()

```
elemental pure logical function mo_utils::is_nan::is_nan_dp (
    real(dp), intent(in) a )
```

16.44.1.2 is_nan_sp()

```
elemental pure logical function mo_utils::is_nan::is_nan_sp (
    real(sp), intent(in) a )
```

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

- [mo_utils.f90](#)

16.45 mo_utils::is_normal Interface Reference

Public Member Functions

- elemental pure logical function [is_normal_sp](#) (a)
- elemental pure logical function [is_normal_dp](#) (a)

16.45.1 Member Function/Subroutine Documentation

16.45.1.1 is_normal_dp()

```
elemental pure logical function mo_utils::is_normal::is_normal_dp (
    real(dp), intent(in) a )
```

16.45.1.2 is_normal_sp()

```
elemental pure logical function mo_utils::is_normal::is_normal_sp (
    real(sp), intent(in) a )
```

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

- [mo_utils.f90](#)

16.46 mo_errormasures::kge Interface Reference

Kling-Gupta-Efficiency measure.

Public Member Functions

- real(dp) function [kge_dp_1d](#) (x, y, mask)
- real(dp) function [kge_dp_2d](#) (x, y, mask)
- real(dp) function [kge_dp_3d](#) (x, y, mask)
- real(sp) function [kge_sp_1d](#) (x, y, mask)
- real(sp) function [kge_sp_2d](#) (x, y, mask)
- real(sp) function [kge_sp_3d](#) (x, y, mask)

16.46.1 Detailed Description

Kling-Gupta-Efficiency measure.

The Kling-Gupta model efficiency coefficient KGE is

$$KGE = 1 - \sqrt{((1-r)^2 + (1-\alpha)^2 + (1-\beta)^2)}$$

where

r = Pearson product-moment correlation coefficient

α = ratio of simulated mean to observed mean

β = ratio of simulated standard deviation to observed standard deviation

This three measures are calculated between two arrays (1d, 2d, or 3d). Usually, one is an observation and the second is a modelled variable.

The higher the KGE the better the observation and simulation are matching. The upper limit of KGE is 1.

Therefore, if you apply a minimization algorithm to calibrate regarding KGE you have to use the objective function

$$obj_value = 1.0 - KGE$$

which has then the optimum at 0.0. (Like for the NSE where you always optimize 1-NSE.)

real(sp/dp), dimension(:) :: x, y 1D-array with input numbers
 real(sp/dp), dimension(:, :) :: x, y 2D-array with input numbers
 real(sp/dp), dimension(:, :, :) :: x, y 3D-array with input numbers
 logical :: mask(:) 1D-array of logical values with size(x/y).
 logical :: mask(:, :) 2D-array of logical values with size(x/y).
 logical :: mask(:, :, :) 3D-array of logical values with size(x/y).

Returns

kge — Kling-Gupta-Efficiency (value less equal 1.0)

Note

Input values must be floating points.

Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." Journal of Hydrology 377.1 (2009): 80-91.

Author

Rohini Kumar

Date

August 2014

16.46.2 Member Function/Subroutine Documentation

16.46.2.1 kge_dp_1d()

```
real(dp) function mo_errormeasures::kge::kge_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.46.2.2 kge_dp_2d()

```
real(dp) function mo_errormeasures::kge::kge_dp_2d (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.46.2.3 kge_dp_3d()

```
real(dp) function mo_errormeasures::kge::kge_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

16.46.2.4 kge_sp_1d()

```
real(sp) function mo_errormeasures::kge::kge_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.46.2.5 kge_sp_2d()

```
real(sp) function mo_errormeasures::kge::kge_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.46.2.6 kge_sp_3d()

```
real(sp) function mo_errormeasures::kge::kge_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.47 mo_errormeasures::kgenocorr Interface Reference

Kling-Gupta-Efficiency measure without correlation.

Public Member Functions

- real(dp) function [kgenocorr_dp_1d](#) (x, y, mask)
- real(dp) function [kgenocorr_dp_2d](#) (x, y, mask)

- real(dp) function [kgenocorr_dp_3d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_1d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_2d](#) (x, y, mask)
- real(sp) function [kgenocorr_sp_3d](#) (x, y, mask)

16.47.1 Detailed Description

Kling-Gupta-Efficiency measure without correlation.

The modified Kling-Gupta model efficiency coefficient *KGenocorr* is

$$KGenocorr = 1 - \sqrt{((1 - \alpha)^2 + (1 - \beta)^2)}$$

where

α = ratio of simulated mean to observed mean

β = ratio of simulated standard deviation to observed standard deviation

This two measures are calculated between two arrays (1d, 2d, or 3d). Usually, one is an observation and the second is a modelled variable.

The higher the KGenocorr the better the observation and simulation are matching. The upper limit of KGenocorr is 1.

Therefore, if you apply a minimization algorithm to calibrate regarding KGenocorr you have to use the objective function

$$obj_value = 1.0 - KGenocorr$$

which has then the optimum at 0.0. (Like for the NSE where you always optimize 1-NSE.)

real(sp/dp), dimension(:) :: x, y 1D-array with input numbers
 real(sp/dp), dimension(:, :) :: x, y 2D-array with input numbers
 real(sp/dp), dimension(:, :, :) :: x, y 3D-array with input numbers
 logical :: mask(:) 1D-array of logical values with size(x/y).
 logical :: mask(:, :) 2D-array of logical values with size(x/y).
 logical :: mask(:, :, :) 3D-array of logical values with size(x/y).

Returns

kgenocorr — Kling-Gupta-Efficiency without correlation (value less equal 1.0)

Note

Input values must be floating points.

Gupta, Hoshin V., et al. "Decomposition of the mean squared error and NSE performance criteria: Implications for improving hydrological modelling." *Journal of Hydrology* 377.1 (2009): 80-91.

Author

Rohini Kumar

Date

August 2014

16.47.2 Member Function/Subroutine Documentation

16.47.2.1 kgenocorr_dp_1d()

```
real(dp) function mo_errormeasures::kgenocorr::kgenocorr_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.47.2.2 kgenocorr_dp_2d()

```
real(dp) function mo_errormeasures::kgenocorr::kgenocorr_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )
```

16.47.2.3 kgenocorr_dp_3d()

```
real(dp) function mo_errormeasures::kgenocorr::kgenocorr_dp_3d (
    real(dp), dimension(:, :, :), intent(in) x,
    real(dp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask )
```

16.47.2.4 kgenocorr_sp_1d()

```
real(sp) function mo_errormeasures::kgenocorr::kgenocorr_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.47.2.5 kgenocorr_sp_2d()

```
real(sp) function mo_errormeasures::kgenocorr::kgenocorr_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )
```

16.47.2.6 kgenocorr_sp_3d()

```
real(sp) function mo_errormeasures::kgenocorr::kgenocorr_sp_3d (
    real(sp), dimension(:, :, :), intent(in) x,
    real(sp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.48 mo_moment::kurtosis Interface Reference**Public Member Functions**

- real(sp) function [kurtosis_sp](#) (dat, mask)
- real(dp) function [kurtosis_dp](#) (dat, mask)

16.48.1 Member Function/Subroutine Documentation

16.48.1.1 kurtosis_dp()

```
real(dp) function mo_moment::kurtosis::kurtosis_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

16.48.1.2 kurtosis_sp()

```
real(sp) function mo_moment::kurtosis::kurtosis_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.49 mo_utils::le Interface Reference

Public Member Functions

- elemental pure logical function [lesserequal_sp](#) (a, b)
- elemental pure logical function [lesserequal_dp](#) (a, b)

16.49.1 Member Function/Subroutine Documentation

16.49.1.1 lesserequal_dp()

```
elemental pure logical function mo_utils::le::lesserequal_dp (  
    real(dp), intent(in) a,  
    real(dp), intent(in) b )
```

16.49.1.2 lesserequal_sp()

```
elemental pure logical function mo_utils::le::lesserequal_sp (  
    real(sp), intent(in) a,  
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.50 mo_utils::lesserequal Interface Reference

Public Member Functions

- elemental pure logical function [lesserequal_sp](#) (a, b)
- elemental pure logical function [lesserequal_dp](#) (a, b)

16.50.1 Member Function/Subroutine Documentation

16.50.1.1 lesserequal_dp()

```
elemental pure logical function mo_utils::lesserequal::lesserequal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.50.1.2 lesserequal_sp()

```
elemental pure logical function mo_utils::lesserequal::lesserequal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.51 mo_linfit::linfit Interface Reference

Fits a straight line to input data by minimizing χ^2 .

Public Member Functions

- real(sp) function, dimension(:), allocatable [linfit_sp](#) (x, y, a, b, siga, sigb, chi2, model2)
- real(dp) function, dimension(:), allocatable [linfit_dp](#) (x, y, a, b, siga, sigb, chi2, model2)

16.51.1 Detailed Description

Fits a straight line to input data by minimizing χ^2 .

Given a set of data points $x(1:n_{\text{data}})$, $y(1:n_{\text{data}})$, fit them to a straight line $y = a + bx$ by minimizing χ^2 .

Model I minimizes y vs. x while Model II takes the geometric mean of y vs. x and x vs. y . Returned is the fitted line at x . Optional returns are a , b and their respective probable uncertainties siga and sigb , and the chi-square χ^2 .

Parameters

in	<i>real(sp/dp) :: x(:)</i>	1D-array with input x
in	<i>real(sp/dp) :: y(:)</i>	1D-array with input y
in	<i>logical, optional :: model2</i>	If present, use geometric mean regression instead of ordinary least square
out	<i>real(sp/dp), dimension(M) :: a</i>	intercept

Parameters

out	<i>real(sp/dp), dimension(M) :: b</i>	slope
out	<i>real(sp/dp), dimension(M) :: siga</i>	error on intercept
out	<i>real(sp/dp), dimension(M) :: sigb</i>	error on slope
out	<i>real(sp/dp) :: chisq</i>	Minimum χ^2

Returns

real(sp/dp), dimension(:), allocatable :: out — fitted values at *x(:)*.

16.51.2 Member Function/Subroutine Documentation

16.51.2.1 linfit_dp()

```
real(dp) function, dimension(:), allocatable mo_linfit::linfit::linfit_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    real(dp), intent(out), optional a,
    real(dp), intent(out), optional b,
    real(dp), intent(out), optional siga,
    real(dp), intent(out), optional sigb,
    real(dp), intent(out), optional chi2,
    logical, intent(in), optional model2 )
```

16.51.2.2 linfit_sp()

```
real(sp) function, dimension(:), allocatable mo_linfit::linfit::linfit_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    real(sp), intent(out), optional a,
    real(sp), intent(out), optional b,
    real(sp), intent(out), optional siga,
    real(sp), intent(out), optional sigb,
    real(sp), intent(out), optional chi2,
    logical, intent(in), optional model2 )
```

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

- [mo_linfit.f90](#)

16.52 mo_errormeasures::lnnse Interface Reference

Public Member Functions

- *real(sp)* function [lnnse_sp_1d](#) (*x*, *y*, *mask*)
- *real(dp)* function [lnnse_dp_1d](#) (*x*, *y*, *mask*)
- *real(dp)* function [lnnse_dp_2d](#) (*x*, *y*, *mask*)
- *real(sp)* function [lnnse_sp_2d](#) (*x*, *y*, *mask*)
- *real(sp)* function [lnnse_sp_3d](#) (*x*, *y*, *mask*)
- *real(dp)* function [lnnse_dp_3d](#) (*x*, *y*, *mask*)

16.52.1 Member Function/Subroutine Documentation

16.52.1.1 Innse_dp_1d()

```
real(dp) function mo_errormeasures::lnnse::lnnse_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(inout), optional mask )
```

16.52.1.2 Innse_dp_2d()

```
real(dp) function mo_errormeasures::lnnse::lnnse_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(inout), optional mask )
```

16.52.1.3 Innse_dp_3d()

```
real(dp) function mo_errormeasures::lnnse::lnnse_dp_3d (
    real(dp), dimension(:, :, :), intent(in) x,
    real(dp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(inout), optional mask )
```

16.52.1.4 Innse_sp_1d()

```
real(sp) function mo_errormeasures::lnnse::lnnse_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(inout), optional mask )
```

16.52.1.5 Innse_sp_2d()

```
real(sp) function mo_errormeasures::lnnse::lnnse_sp_2d (
    real(sp), dimension(:, :), intent(in) x,
    real(sp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(inout), optional mask )
```

16.52.1.6 Innse_sp_3d()

```
real(sp) function mo_errormeasures::lnnse::lnnse_sp_3d (
    real(sp), dimension(:, :, :), intent(in) x,
    real(sp), dimension(:, :, :), intent(in) y,
    logical, dimension(:, :, :), intent(inout), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.53 mo_utils::locate Interface Reference

Find closest values in a monotonic series, returns the indexes.

Public Member Functions

- integer(i4) function [locate_0d_dp](#) (x, y)
- integer(i4) function [locate_0d_sp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [locate_1d_dp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [locate_1d_sp](#) (x, y)

16.53.1 Detailed Description

Find closest values in a monotonic series, returns the indexes.

Given an array $x(1:n)$, and given a value y , returns a value j such that y is between $x(j)$ and $x(j+1)$.

x must be monotonically increasing.

$j=0$ or $j=N$ is returned to indicate that x is out of range.

Parameters

in	<i>real(dp/sp) :: x(:)</i>	Sorted array
in	<i>real(dp/sp) :: y[(:)]</i>	Value(s) of which the closest match in $x(:)$ is wanted

Returns

integer(i4) :: index[(:)] — index(es) of x so that y is between $x(\text{index})$ and $x(\text{index}+1)$

Note

x must be monotonically increasing.

Author

Matthias Cuntz

Date

May 2014

16.53.2 Member Function/Subroutine Documentation

16.53.2.1 locate_0d_dp()

```
integer(i4) function mo_utils::locate::locate_0d_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), intent(in) y )
```

16.53.2.2 locate_0d_sp()

```
integer(i4) function mo_utils::locate::locate_0d_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), intent(in) y )
```

16.53.2.3 locate_1d_dp()

```
integer(i4) function, dimension(:), allocatable mo_utils::locate::locate_1d_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y )
```

16.53.2.4 locate_1d_sp()

```
integer(i4) function, dimension(:), allocatable mo_utils::locate::locate_1d_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y )
```

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

- [mo_utils.f90](#)

16.54 mo_errormeasures::mae Interface Reference**Public Member Functions**

- real(sp) function [mae_sp_1d](#) (x, y, mask)
- real(dp) function [mae_dp_1d](#) (x, y, mask)
- real(sp) function [mae_sp_2d](#) (x, y, mask)
- real(dp) function [mae_dp_2d](#) (x, y, mask)
- real(sp) function [mae_sp_3d](#) (x, y, mask)
- real(dp) function [mae_dp_3d](#) (x, y, mask)

16.54.1 Member Function/Subroutine Documentation**16.54.1.1 mae_dp_1d()**

```
real(dp) function mo_errormeasures::mae::mae_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.54.1.2 mae_dp_2d()

```
real(dp) function mo_errormeasures::mae::mae_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
```

```

real(dp), dimension(:,:), intent(in) y,
logical, dimension(:,:), intent(in), optional mask )

```

16.54.1.3 mae_dp_3d()

```

real(dp) function mo_errormeasures::mae::mae_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )

```

16.54.1.4 mae_sp_1d()

```

real(sp) function mo_errormeasures::mae::mae_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )

```

16.54.1.5 mae_sp_2d()

```

real(sp) function mo_errormeasures::mae::mae_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )

```

16.54.1.6 mae_sp_3d()

```

real(sp) function mo_errormeasures::mae::mae_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )

```

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

- [mo_errormeasures.f90](#)

16.55 mo_mcmc::mcmc Interface Reference

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are either known or modeled).

Public Member Functions

- subroutine [mcmc_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, restart, restart_file, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)

16.55.1 Detailed Description

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are either known or modeled).

Sample posterior parameter distribution with Metropolis Algorithm.

This sampling is performed in two steps, i.e. the burn-in phase for adjusting model dependent parameters for the second step which is the proper sampling using the Metropolis Hastings Algorithm.

This sampler does not change the best parameter set, i.e. it cannot be used as an optimiser.

However, the serial and the parallel version give therefore the bitwise same results. **1. BURN IN PHASE: FIND THE OPTIMAL STEP SIZE**

Purpose:

Find optimal stepsize for each parameter such that the acceptance ratio converges to a value around 0.3.

Important variables:

Variable	Description
burnin_iter	length of markov chain performed to calculate acceptance ratio
acceptance ratio	ratio between accepted jumps and all trials (LEN)
acceptance multiplier	stepsize of a parameter is multiplied with this value when jump is

accepted (initial : 1.01) rejection multiplier | stepsize of a parameter is multiplied with this value when jump is rejected (initial : 0.99 and will never be changed) stepsize | a new parameter value is chosen based on a uniform distribution $p_{new_i} = p_{old_i} + \text{Unif}(-\text{stepsize_i}, \text{stepsize_i})$ (initial : stepsize_i = 1.0 for all i)

Algorithm:

1. start a new markov chain of length burnin_iter with initial parameter set is the OPTIMAL one

- select a set of parameters to change:
 - accurate → choose one parameter,
 - comput. efficient → choose all parameters,
 - moderate accurate & efficient → choose half of the parameters
- change parameter(s) based on their stepsize
- decide whether changed parameter set is accepted or rejected:
 - $\text{oddsRatio} = \frac{\text{likelihood}(p_{new})}{\text{likelihood}(p_{old})}$
 - random number $r = \text{Uniform}[0,1]$
 - odds Ratio > 0 → positive accept
 - odds Ratio > r → negative accept
 - odds Ratio < r → reject
- adapt stepsize of parameters changed:

- accepted step: $\text{stepsize}_i = \text{stepsize}_i * \text{acceptance multiplier}$
 - rejected step: $\text{stepsize}_i = \text{stepsize}_i * \text{rejection multiplier}$
 - if step is accepted: for all changed parameter(s) change stepsize
2. calculate acceptance ratio of the Markov Chain
 3. adjust acceptance multiplier acc_mult and store good ratios in history list
 - acceptance ratio $< 0.23 \rightarrow \text{acc_mult} = \text{acc_mult} * 0.99$
delete history list
 - acceptance ratio $> 0.44 \rightarrow \text{acc_mult} = \text{acc_mult} * 1.01$
delete history list
 - $0.23 < \text{acceptance ratio} < 0.44$
add acceptance ratio to history list
 4. check if already 10 values are stored in history list and if they have converged to a value above 0.3
(mean above 0.3 and variance less $\sqrt{1/12 * 0.05^2} = \text{Variance of uniform } [\text{acc_ratio} \pm 2.5\%]$)
 - if check is positive abort and save stepsizes
else goto (1)

2. MONTE CARLO MARKOV CHAIN: SAMPLE POSTERIOR DISTRIBUTION OF PARAMETER

Purpose:

use the previous adapted stepsizes and perform ONE monte carlo markov chain
the accepted parameter sets show the posterior distribution of parameters

Important variables:

Variable	Description
iter_mcmc	length of the markov chain ($>>$ iter_burnin)
stepsize	a new parameter value is chosen based on a uniform distribution

$\text{pnew}_i = \text{pold}_i + \text{Unif}(-\text{stepsize}_i, \text{stepsize}_i)$ use stepsizes of the burn-in (1)

Algorithm:

1. select a set of parameters to change
 - accurate \rightarrow choose one parameter,
 - comput. efficient \rightarrow choose all parameters,

- moderate accurate & efficient → choose half of the parameters
2. change parameter(s) based on their stepsize
 3. decide whether changed parameter set is accepted or rejected:
 - $\text{oddsRatio} = \frac{\text{likelihood}(p_{\text{new}})}{\text{likelihood}(p_{\text{old}})}$
 - random number $r = \text{Uniform}[0,1]$
 - odds Ratio > 0 → positive accept
 odds Ratio $> r$ → negative accept
 odds Ratio $< r$ → reject
 4. if step is accepted: save parameter set
 5. goto (1)

Parameters

in	<i>real(dp) :: likelihood(x)</i>	Interface Function which calculates likelihood of given parameter set x
in	<i>real(dp) :: para(:)</i>	Initial parameter set (should be GOOD approximation of best parameter set)
in	<i>real(dp) :: rangePar(size(para),2)</i>	Min/max range of parameters
out	<i>real(dp), allocatable :: mcmc_paras(:, :)</i>	Parameter sets sampled in proper MCMC part of algorithm
out	<i>real(dp), allocatable :: burnin_paras(:, :)</i>	Parameter sets sampled during burn-in part of algorithm
in	<i>integer(i8), optional :: seed_in</i>	User seed to initialise the random number generator (default: none → initialized with timeseed)
in	<i>logical, optional :: printflag_in</i>	Print of output on command line (default: .False.)
in	<i>logical, optional :: maskpara_in(size(para))</i>	Parameter will be sampled (.True.) or not (.False.) (default: .True.)
in	<i>character(len=*), optional :: tmp_file</i>	filename for temporal data saving: every iter_mcmc_in iterations parameter sets are appended to this file the number of the chain will be prepended to filename output format: netcdf (default: no file writing)
in	<i>logical, optional :: loglike_in</i>	true if loglikelihood function is given instead of likelihood function (default: .false.)
in	<i>integer(i4), optional :: ParaSelectMode_in</i>	How many parameters will be changed at once? <ul style="list-style-type: none"> • half of the parameter → 1_i4 • only one parameter → 2_i4 • all parameter → 3_i4 (default: 2_i4)
		Generated on December 1, 2017

Parameters

in	<i>integer(i4), optional :: iter_burnin_in</i>	Length of Markov chains of initial burn-in part (default: Max(250, 200*count(maskpara)))
in	<i>integer(i4), optional :: iter_mcmc_in</i>	Length of Markov chains of proper MCMC part (default: 1000 * count(maskpara))
in	<i>integer(i4), optional :: chains_in</i>	number of parallel mcmc chains (default: 5_i4)
in	<i>real(dp), DIMENSION(size(para,1)), optional :: stepsize_in</i>	stepsize for each parameter if given burn-in is discarded (default: none -> adjusted in burn-in)

Note

Likelihood has to be defined as a function interface
The maximal number of parameters is 1000.

16.55.2 Member Function/Subroutine Documentation

16.55.2.1 mcmc_dp()

```

subroutine mo_mcmc::mcmc::mcmc_dp (
    likelihood,
    real(dp), dimension(:), intent(in) para,
    real(dp), dimension(:, :), intent(in) rangePar,
    real(dp), dimension(:, :), intent(out), allocatable mcmc_paras,
    real(dp), dimension(:, :), intent(out), allocatable burnin_paras,
    integer(i8), intent(in), optional seed_in,
    logical, intent(in), optional printflag_in,
    logical, dimension(size(para,1)), intent(in), optional maskpara_in,
    logical, intent(in), optional restart,
    character(len=*), intent(in), optional restart_file,
    character(len=*), intent(in), optional tmp_file,
    logical, intent(in), optional loglike_in,
    integer(i4), intent(in), optional ParaSelectMode_in,
    integer(i4), intent(in), optional iter_burnin_in,
    integer(i4), intent(in), optional iter_mcmc_in,
    integer(i4), intent(in), optional chains_in,
    real(dp), dimension(size(para,1)), intent(in), optional stepsize_in )

```

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

- [mo_mcmc.f90](#)

16.56 mo_mcmc::mcmc_stddev Interface Reference

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are neither known nor modeled).

Public Member Functions

- subroutine [mcmc_stddev_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)

16.56.1 Detailed Description

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are neither known nor modeled).

Sample posterior parameter distribution with Metropolis Algorithm.

This sampling is performed in two steps, i.e. the burn-in phase for adjusting model dependent parameters for the second step which is the proper sampling using the Metropolis Hastings Algorithm.

1. BURN IN PHASE: FIND THE OPTIMAL STEP SIZE

Purpose:

Find optimal stepsize for each parameter such that the acceptance ratio converges to a value around 0.3.

Important variables:

Variable	Description
burnin_iter	length of markov chain performed to calculate acceptance ratio
acceptance ratio	ratio between accepted jumps and all trials (LEN)
acceptance multiplier	stepsize of a parameter is multiplied with this value when jump is

accepted (initial : 1.01) rejection multiplier | stepsize of a parameter is multiplied with this value when jump is rejected (initial : 0.99 and will never be changed) stepsize | a new parameter value is chosen based on a uniform distribution $p_{new_i} = p_{old_i} + \text{Unif}(-\text{stepsize_i}, \text{stepsize_i})$ (initial : stepsize_i = 1.0 for all i)

Algorithm:

1. start a new markov chain of length burnin_iter with initial parameter set is the OPTIMAL one

- select a set of parameters to change:
 - accurate → choose one parameter,
 - comput. efficient → choose all parameters,
 - moderate accurate & efficient → choose half of the parameters
- change parameter(s) based on their stepsize
- decide whether changed parameter set is accepted or rejected:
 - $\text{oddsRatio} = \frac{\text{likelihood}(p_{new})}{\text{likelihood}(p_{old})}$
 - random number $r = \text{Uniform}[0,1]$
 - odds Ratio > 0 → positive accept
odds Ratio > r → negative accept

odds Ratio < r → reject

- adapt stepsize of parameters changed:
 - accepted step: $\text{stepsize}_i = \text{stepsize}_i * \text{acceptance multiplier}$
 - rejected step: $\text{stepsize}_i = \text{stepsize}_i * \text{rejection multiplier}$
- if step is accepted: for all changed parameter(s) change stepsize

2. calculate acceptance ratio of the Markov Chain

3. adjust acceptance multiplier acc_mult and store good ratios in history list

- acceptance ratio < 0.23 → $\text{acc_mult} = \text{acc_mult} * 0.99$
delete history list
- acceptance ratio > 0.44 → $\text{acc_mult} = \text{acc_mult} * 1.01$
delete history list
- $0.23 < \text{acceptance ratio} < 0.44$
add acceptance ratio to history list

4. check if already 10 values are stored in history list and if they have converged to a value above 0.3
(mean above 0.3 and variance less $\sqrt{1/12 * 0.05^2} = \text{Variance of uniform [acc_ratio +/- 2.5\%]}$)

- if check is positive abort and save stepsizes
else goto (1)

2. MONTE CARLO MARKOV CHAIN: SAMPLE POSTERIOR DISTRIBUTION OF PARAMETER

Purpose:

use the previous adapted stepsizes and perform ONE monte carlo markov chain
the accepted parameter sets show the posterior distribution of parameters

Important variables:

Variable	Description
iter_mcmc	length of the markov chain (>> iter_burnin)
stepsize	a new parameter value is chosen based on a uniform distribution

$\text{pnew}_i = \text{pold}_i + \text{Unif}(-\text{stepsize}_i, \text{stepsize}_i)$ use stepsizes of the burn-in (1)

Algorithm:

1. select a set of parameters to change

- accurate → choose one parameter,
 - comput. efficient → choose all parameters,
 - moderate accurate & efficient → choose half of the parameters
2. change parameter(s) based on their stepsize
 3. decide whether changed parameter set is accepted or rejected:
 - $\text{oddsRatio} = \frac{\text{likelihood}(p_{\text{new}})}{\text{likelihood}(p_{\text{old}})}$
 - random number $r = \text{Uniform}[0,1]$
 - odds Ratio > 0 → positive accept
 odds Ratio $> r$ → negative accept
 odds Ratio $< r$ → reject
 4. if step is accepted: save parameter set
 5. goto (1)

Parameters

in	<i>real(dp) :: likelihood(x,sigma,stddev_new,likeli_new)</i>	Interface Function which calculates likelihood of given parameter set x and given standard deviation sigma and returns optionally the standard deviation stddev_new of the errors using x and likelihood likeli_new using stddev_new
in	<i>real(dp) :: para(:)</i>	Initial parameter set (should be GOOD approximation of best parameter set)
in	<i>real(dp) :: rangePar(size(para),2)</i>	Min/max range of parameters
out	<i>real(dp), allocatable :: mcmc_paras(:, :)</i>	Parameter sets sampled in proper MCMC part of algorithm
out	<i>real(dp), allocatable :: burnin_paras(:, :)</i>	Parameter sets sampled during burn-in part of algorithm
in	<i>integer(i8), optional :: seed_in</i>	User seed to initialise the random number generator (default: none → initialized with timeseed)
in	<i>logical, optional :: printflag_in</i>	Print of output on command line (default: .False.)
in	<i>logical, optional :: maskpara_in(size(para))</i>	Parameter will be sampled (.True.) or not (.False.) (default: .True.)
in	<i>character(len=*), optional :: tmp_file</i>	filename for temporal data saving: every iter_mcmc_in iterations parameter sets are appended to this file the number of the chain will be prepended to filename output format: netcdf (default: no file writing)

Parameters

in	<i>logical, optional :: loglike_in</i>	true if loglikelihood function is given instead of likelihood function (default: .false.)
in	<i>integer(i4), optional :: ParaSelectMode_in</i>	How many parameters will be changed at once? <ul style="list-style-type: none">• half of the parameter → 1_i4• only one parameter → 2_i4• all parameter → 3_i4 (default: 2_i4)
in	<i>integer(i4), optional :: iter_burnin_in</i>	Length of Markov chains of initial burn-in part (default: Max(250, 200*count(maskpara)))
in	<i>integer(i4), optional :: iter_mcmc_in</i>	Length of Markov chains of proper MCMC part (default: 1000 * count(maskpara))
in	<i>integer(i4), optional :: chains_in</i>	number of parallel mcmc chains (default: 5_i4)
in	<i>real(dp), DIMENSION(size(para,1)), optional :: stepsize_in</i>	stepsize for each parameter if given burn-in is discarded (default: none → adjusted in burn-in)

16.56.2 Member Function/Subroutine Documentation

16.56.2.1 mcmc_stddev_dp()

```

subroutine mo_mcmc::mcmc_stddev::mcmc_stddev_dp (
    likelihood,
    real(dp), dimension(:), intent(in) para,
    real(dp), dimension(:, :), intent(in) rangePar,
    real(dp), dimension(:, :), intent(out), allocatable mcmc_paras,
    real(dp), dimension(:, :), intent(out), allocatable burnin_paras,
    integer(i8), intent(in), optional seed_in,
    logical, intent(in), optional printflag_in,
    logical, dimension(size(para,1)), intent(in), optional maskpara_in,
    character(len=*), intent(in), optional tmp_file,
    logical, intent(in), optional loglike_in,
    integer(i4), intent(in), optional ParaSelectMode_in,
    integer(i4), intent(in), optional iter_burnin_in,
    integer(i4), intent(in), optional iter_mcmc_in,
    integer(i4), intent(in), optional chains_in,
    real(dp), dimension(size(para,1)), intent(in), optional stepsize_in )

```

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

- [mo_mcmc.f90](#)

16.57 mo_moment::mean Interface Reference

Public Member Functions

- `real(sp)` function [mean_sp](#) (dat, mask)

- real(dp) function [mean_dp](#) (dat, mask)

16.57.1 Member Function/Subroutine Documentation

16.57.1.1 mean_dp()

```
real(dp) function mo_moment::mean::mean_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

16.57.1.2 mean_sp()

```
real(sp) function mo_moment::mean::mean_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.58 mo_template::mean Interface Reference

The average.

Public Member Functions

- real(sp) function [mean_sp](#) (dat, mask)
- real(dp) function [mean_dp](#) (dat, mask)

16.58.1 Detailed Description

The average.

Calculates the average value of a vector, i.e. the first moment of a series of numbers:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

If an optional mask is given, the mean is only over those locations that correspond to true values in the mask. x can be single or double precision. The result will have the same numerical precision.

Parameters

in	<i>real(sp/dp) :: dat(:)</i>	x_i 1D-array with input numbers
in	<i>logical, optional :: mask(:)</i>	1D-array with input mask If present, only those locations in dat corresponding to the true values in mask are used.

Returns

real(sp/dp) :: mean — \bar{x} average of all elements in vec

Note

Input values must be floating points.

Author

Matthias Cuntz

Date

Nov 2011

16.58.2 Member Function/Subroutine Documentation**16.58.2.1 mean_dp()**

```
real(dp) function mo_template::mean::mean_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

16.58.2.2 mean_sp()

```
real(sp) function mo_template::mean::mean_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_template.f90](#)

16.59 mo_percentile::median Interface Reference**Public Member Functions**

- real(sp) function [median_sp](#) (arrin, mask)
- real(dp) function [median_dp](#) (arrin, mask)

16.59.1 Member Function/Subroutine Documentation**16.59.1.1 median_dp()**

```
real(dp) function mo_percentile::median::median_dp (
    real(dp), dimension(:), intent(in) arrin,
    logical, dimension(:), intent(in), optional mask )
```

16.59.1.2 median_sp()

```
real(sp) function mo_percentile::median::median_sp (
    real(sp), dimension(:), intent(in) arrin,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_percentile.f90](#)

16.60 mo_moment::mixed_central_moment Interface Reference

Public Member Functions

- real(sp) function [mixed_central_moment_sp](#) (x, y, r, s, mask)
- real(dp) function [mixed_central_moment_dp](#) (x, y, r, s, mask)

16.60.1 Member Function/Subroutine Documentation

16.60.1.1 mixed_central_moment_dp()

```
real(dp) function mo_moment::mixed_central_moment::mixed_central_moment_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) r,
    integer(i4), intent(in) s,
    logical, dimension(:), intent(in), optional mask )
```

16.60.1.2 mixed_central_moment_sp()

```
real(sp) function mo_moment::mixed_central_moment::mixed_central_moment_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    integer(i4), intent(in) r,
    integer(i4), intent(in) s,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.61 mo_moment::mixed_central_moment_var Interface Reference

Public Member Functions

- real(sp) function [mixed_central_moment_var_sp](#) (x, y, r, s, mask)
- real(dp) function [mixed_central_moment_var_dp](#) (x, y, r, s, mask)

16.61.1 Member Function/Subroutine Documentation

16.61.1.1 mixed_central_moment_var_dp()

```
real(dp) function mo_moment::mixed_central_moment_var::mixed_central_moment_var_dp (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    integer(i4), intent(in) r,
    integer(i4), intent(in) s,
    logical, dimension(:), intent(in), optional mask )
```

16.61.1.2 mixed_central_moment_var_sp()

```
real(sp) function mo_moment::mixed_central_moment_var::mixed_central_moment_var_sp (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    integer(i4), intent(in) r,
    integer(i4), intent(in) s,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.62 mo_moment::moment Interface Reference

Public Member Functions

- subroutine [moment_sp](#) (dat, [average](#), [variance](#), [skewness](#), [kurtosis](#), [mean](#), [stddev](#), [absdev](#), mask)
- subroutine [moment_dp](#) (dat, [average](#), [variance](#), [skewness](#), [kurtosis](#), [mean](#), [stddev](#), [absdev](#), mask)

16.62.1 Member Function/Subroutine Documentation

16.62.1.1 moment_dp()

```
subroutine mo_moment::moment::moment_dp (
    real(dp), dimension(:), intent(in) dat,
    real(dp), intent(out), optional average,
    real(dp), intent(out), optional variance,
    real(dp), intent(out), optional skewness,
    real(dp), intent(out), optional kurtosis,
    real(dp), intent(out), optional mean,
    real(dp), intent(out), optional stddev,
    real(dp), intent(out), optional absdev,
    logical, dimension(:), intent(in), optional mask )
```

16.62.1.2 moment_sp()

```
subroutine mo_moment::moment::moment_sp (
    real(sp), dimension(:), intent(in) dat,
    real(sp), intent(out), optional average,
    real(sp), intent(out), optional variance,
    real(sp), intent(out), optional skewness,
    real(sp), intent(out), optional kurtosis,
    real(sp), intent(out), optional mean,
    real(sp), intent(out), optional stddev,
    real(sp), intent(out), optional absdev,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.63 mo_orderpack::mrgref Interface Reference

Public Member Functions

- subroutine [d_mrgref](#) (XVALT, IRNGT)
- subroutine [r_mrgref](#) (XVALT, IRNGT)
- subroutine [i_mrgref](#) (XVALT, IRNGT)

16.63.1 Member Function/Subroutine Documentation

16.63.1.1 d_mrgref()

```
subroutine mo_orderpack::mrgref::d_mrgref (
    real(kind=dp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

16.63.1.2 i_mrgref()

```
subroutine mo_orderpack::mrgref::i_mrgref (
    integer(kind=i4), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

16.63.1.3 r_mrgref()

```
subroutine mo_orderpack::mrgref::r_mrgref (
    real(kind=sp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

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

- [mo_orderpack.f90](#)

16.64 mo_orderpack::mrgrnk Interface Reference

Public Member Functions

- subroutine [d_mrgrnk](#) (XDONT, IRNGT)
- subroutine [r_mrgrnk](#) (XDONT, IRNGT)
- subroutine [i_mrgrnk](#) (XDONT, IRNGT)

16.64.1 Member Function/Subroutine Documentation

16.64.1.1 d_mrgrnk()

```
subroutine mo_orderpack::mrgrnk::d_mrgrnk (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

16.64.1.2 i_mrgrnk()

```
subroutine mo_orderpack::mrgrnk::i_mrgrnk (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

16.64.1.3 r_mrgrnk()

```
subroutine mo_orderpack::mrgrnk::r_mrgrnk (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT )
```

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

- [mo_orderpack.f90](#)

16.65 mo_errormeasures::mse Interface Reference

Public Member Functions

- real(sp) function [mse_sp_1d](#) (x, y, mask)
- real(dp) function [mse_dp_1d](#) (x, y, mask)
- real(sp) function [mse_sp_2d](#) (x, y, mask)
- real(dp) function [mse_dp_2d](#) (x, y, mask)
- real(sp) function [mse_sp_3d](#) (x, y, mask)
- real(dp) function [mse_dp_3d](#) (x, y, mask)

16.65.1 Member Function/Subroutine Documentation

16.65.1.1 mse_dp_1d()

```
real(dp) function mo_errormeasures::mse::mse_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.65.1.2 mse_dp_2d()

```
real(dp) function mo_errormeasures::mse::mse_dp_2d (  
    real(dp), dimension(:, :), intent(in) x,  
    real(dp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.65.1.3 mse_dp_3d()

```
real(dp) function mo_errormeasures::mse::mse_dp_3d (  
    real(dp), dimension(:, :, :), intent(in) x,  
    real(dp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

16.65.1.4 mse_sp_1d()

```
real(sp) function mo_errormeasures::mse::mse_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.65.1.5 mse_sp_2d()

```
real(sp) function mo_errormeasures::mse::mse_sp_2d (  
    real(sp), dimension(:, :), intent(in) x,  
    real(sp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.65.1.6 mse_sp_3d()

```
real(sp) function mo_errormeasures::mse::mse_sp_3d (  
    real(sp), dimension(:, :, :), intent(in) x,  
    real(sp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.66 mo_orderpack::mulcnt Interface Reference

Public Member Functions

- subroutine [d_mulcnt](#) (XDONT, IMULT)
- subroutine [r_mulcnt](#) (XDONT, IMULT)
- subroutine [i_mulcnt](#) (XDONT, IMULT)

16.66.1 Member Function/Subroutine Documentation

16.66.1.1 d_mulcnt()

```
subroutine mo_orderpack::mulcnt::d_mulcnt (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IMULT )
```

16.66.1.2 i_mulcnt()

```
subroutine mo_orderpack::mulcnt::i_mulcnt (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IMULT )
```

16.66.1.3 r_mulcnt()

```
subroutine mo_orderpack::mulcnt::r_mulcnt (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IMULT )
```

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

- [mo_orderpack.f90](#)

16.67 mo_percentile::n_element Interface Reference

Public Member Functions

- real(dp) function [n_element_dp](#) (idat, n, mask, before, after, previous, next)
- real(sp) function [n_element_sp](#) (idat, n, mask, before, after, previous, next)

16.67.1 Member Function/Subroutine Documentation

16.67.1.1 `n_element_dp()`

```
real(dp) function mo_percentile::n_element::n_element_dp (
    real(dp), dimension(:), intent(in) idat,
    integer(i4), intent(in) n,
    logical, dimension(:), intent(in), optional mask,
    real(dp), intent(out), optional before,
    real(dp), intent(out), optional after,
    real(dp), intent(out), optional previous,
    real(dp), intent(out), optional next )
```

16.67.1.2 `n_element_sp()`

```
real(sp) function mo_percentile::n_element::n_element_sp (
    real(sp), dimension(:), intent(in) idat,
    integer(i4), intent(in) n,
    logical, dimension(:), intent(in), optional mask,
    real(sp), intent(out), optional before,
    real(sp), intent(out), optional after,
    real(sp), intent(out), optional previous,
    real(sp), intent(out), optional next )
```

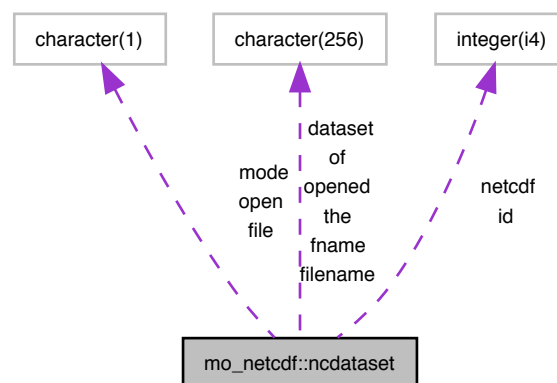
The documentation for this interface was generated from the following file:

- [mo_percentile.f90](#)

16.68 `mo_netcdf::ncdataset` Interface Reference

Provides basic file modification functionality.

Collaboration diagram for `mo_netcdf::ncdataset`:



Public Member Functions

- procedure, public [initncdataset](#)
- procedure, public [getnovariables](#)
- procedure, public [getvariableids](#)
- procedure, public [getvariables](#)
- procedure, public [hasvariable](#)
 - Check if variable exists.*
- procedure, public [hasdimension](#)
 - Check if dimension exists.*
- procedure, public [getunlimiteddimension](#)
 - Return the unlimited dimension of the dataset.*
- procedure, public [isunlimited](#) => isDatasetUnlimited
 - Check if the dataset is unlimited.*
- procedure, public [close](#)
 - Close the dataset.*
- procedure, public [setdimension](#)
 - Create a new dimension.*
- generic, public [setattribute](#) => setGlobalAttributeChar, setGlobalAttributeI8, setGlobalAttributeI16, setGlobalAttributeI32, setGlobalAttributeF32, setGlobalAttributeF64
 - Create a new global Attribute.*
- generic, public [getattribute](#) => getGlobalAttributeChar, getGlobalAttributeI8, getGlobalAttributeI16, getGlobalAttributeI32, getGlobalAttributeF32, getGlobalAttributeF64
 - Retrieve global attribute value.*
- generic, public [getdimension](#) => getDimensionById, getDimensionByName
 - Retrieve NcDimension.*
- generic, public [setvariable](#) => setVariableWithNames, setVariableWithTypes, setVariableWithIds
 - Create a NetCDF variable.*
- generic, public [getvariable](#) => getVariableByName
 - Retrieve NcVariable.*

Public Attributes

- character(256) [fname](#)
- character(256) [filename](#)
- character(256) [of](#)
- character(256) [the](#)
- character(256) [opened](#)
- character(256) [dataset](#)
- character(1) [mode](#)
- character(1) [file](#)
- character(1) [open](#)
- integer(i4) [id](#)
- integer(i4) [netcdf](#)

Private Member Functions

- procedure, private [setglobalattributechar](#)
- procedure, private [setglobalattributei8](#)
- procedure, private [setglobalattributei16](#)
- procedure, private [setglobalattributei32](#)
- procedure, private [setglobalattributef32](#)

- procedure, private [setglobalattributef64](#)
- procedure, private [getglobalattributefchar](#)
- procedure, private [getglobalattributei8](#)
- procedure, private [getglobalattributei16](#)
- procedure, private [getglobalattributei32](#)
- procedure, private [getglobalattributef32](#)
- procedure, private [getglobalattributef64](#)
- procedure, private [getdimensionbyname](#)
- procedure, private [getdimensionbyid](#)
- procedure, private [setvariablewithtypes](#)
- procedure, private [setvariablewithnames](#)
- procedure, private [setvariablewithids](#)
- procedure, private [getvariablebyname](#)

16.68.1 Detailed Description

Provides basic file modification functionality.

Bound to this derived type is the basic file level create/retrieve functionality, i.e. functions/subroutines to create/retrieve dimensions, variables and global attributes. All files created by this derived type and its procedures are are NF90_NETCDF4 only. The supported modes are: r: read w: write/create

Parameters

in	<i>character(*) :: fname</i>	
in	<i>character(1) :: mode</i>	

Returns

"type(NcDataset)"

16.68.2 Member Function/Subroutine Documentation

16.68.2.1 close()

```
procedure, public mo_netcdf::ncdataset::close ( )
```

Close the dataset.

Close the NetCDF dataset. The program will terminate abruptly if the file cannot be closed correctly.

Author

David Schaefer

Date

June 2015

16.68.2.2 getattribute()

```
generic, public mo_netcdf::ncdataset::getattribute ( )
```

Retrieve global attribute value.

Retrieve the value for a global attribute specified by its name. The program will terminate abruptly if the attribute does not exist.

Parameters

in	<i>character(*) :: name</i>	
out	<i>character(*)/integer(i4)/real(sp)/real(dp) :: value</i>	

Author

David Schaefer

Date

June 2015

16.68.2.3 getdimension()

```
generic, public mo_netcdf::ncdataset::getdimension ( )
```

Retrieve NcDimension.

Retrieve the NcDimension derived type for the dimension specified by its name or id. The program will terminate abruptly if no such dimension exists.

Parameters

in	<i>character(*)/integer(i4) :: name/id</i>	
----	--	--

Returns

NcDimension

Author

David Schaefer

Date

June 2015

16.68.2.4 getdimensionbyid()

```
procedure, private mo_netcdf::ncdataset::getdimensionbyid ( ) [private]
```

16.68.2.5 getdimensionbyname()

```
procedure, private mo_netcdf::ncdataset::getdimensionbyname ( ) [private]
```

16.68.2.6 getglobalattributechar()

```
procedure, private mo_netcdf::ncdataset::getglobalattributechar ( ) [private]
```

16.68.2.7 getglobalattributef32()

```
procedure, private mo_netcdf::ncdataset::getglobalattributef32 ( ) [private]
```

16.68.2.8 getglobalattributef64()

```
procedure, private mo_netcdf::ncdataset::getglobalattributef64 ( ) [private]
```

16.68.2.9 getglobalattributei16()

```
procedure, private mo_netcdf::ncdataset::getglobalattributei16 ( ) [private]
```

16.68.2.10 getglobalattributei32()

```
procedure, private mo_netcdf::ncdataset::getglobalattributei32 ( ) [private]
```

16.68.2.11 getglobalattributei8()

```
procedure, private mo_netcdf::ncdataset::getglobalattributei8 ( ) [private]
```

16.68.2.12 getnvariables()

```
procedure, public mo_netcdf::ncdataset::getnvariables ( )
```

16.68.2.13 getunlimiteddimension()

```
procedure, public mo_netcdf::ncdataset::getunlimiteddimension ( )
```

Return the unlimited dimension of the dataset.

Returns the NcDimension derived type of the unlimited dimension. The program will terminate abruptly if no such dimension exists.

Parameters

in	<i>character(*) :: name</i>	
----	-----------------------------	--

Returns

"logical"

Author

David Schaefer

Date

June 2015

16.68.2.14 getvariable()

```
generic, public mo_netcdf::ncdataset::getvariable ( )
```

Retrieve NcVariable.

Retrieve the NcVariable derived type for the variable specified by its name. The program will terminate abruptly if no such dimension exists.

Parameters

in	<i>character(*) :: name</i>	
----	-----------------------------	--

Returns

NcVariable

Author

David Schaefer

Date

June 2015

16.68.2.15 getvariablebyname()

```
procedure, private mo_netcdf::ncdataset::getvariablebyname ( ) [private]
```

16.68.2.16 getvariableids()

```
procedure, public mo_netcdf::ncdataset::getvariableids ( )
```

16.68.2.17 getvariables()

```
procedure, public mo_netcdf::ncdataset::getvariables ( )
```

16.68.2.18 hasdimension()

```
procedure, public mo_netcdf::ncdataset::hasdimension ( )
```

Check if dimension exists.

Returns true if a dimension with the given name exists, false otherwise.

Parameters

in	<i>character(*) :: name</i>	
----	-----------------------------	--

Returns

"logical"

Author

David Schaefer

Date

June 2015

16.68.2.19 hasvariable()

```
procedure, public mo_netcdf::ncdataset::hasvariable ( )
```

Check if variable exists.

Returns true if a variable with the given name exists, false otherwise.

Parameters

in	<i>character(*) :: name</i>	
----	-----------------------------	--

Returns

"logical"

Author

David Schaefer

Date

June 2015

16.68.2.20 initncdataset()

```
procedure, public mo_netcdf::ncdataset::initncdataset ( )
```

16.68.2.21 isunlimited()

```
procedure, public mo_netcdf::ncdataset::isunlimited ( )
```

Check if the dataset is unlimited.

Returns true if the dataset contains an unlimited dimension, false otherwise.

Returns

"logical"

Author

David Schaefer

Date

June 2015

16.68.2.22 setattribute()

```
generic, public mo_netcdf::ncdataset::setattribute ( )
```

Create a new global Attribute.

Create a new global attribute from given name and value. The program will terminate abruptly if the attribute cannot be created.

Parameters

in	<i>character(*) :: name</i>	
in	<i>character(*)/integer(i4)/real(sp)/real(dp) :: value</i>	

Author

David Schaefer

Date

June 2015

16.68.2.23 setdimension()

```
procedure, public mo_netcdf::ncdataset::setdimension ( )
```

Create a new dimension.

Create a new dimension of given length. A length < 0 indicates an unlimited dimension. The program will terminate abruptly if the dimension cannot be created.

Parameters

in	<i>character(*) :: name</i>	
in	<i>integer(i4) :: length</i>	

Returns

NcDimension

Author

David Schaefer

Date

June 2015

16.68.2.24 setglobalattributecar()

```
procedure, private mo_netcdf::ncdataset::setglobalattributecar ( ) [private]
```

16.68.2.25 setglobalattributef32()

```
procedure, private mo_netcdf::ncdataset::setglobalattributef32 ( ) [private]
```

16.68.2.26 setglobalattributef64()

```
procedure, private mo_netcdf::ncdataset::setglobalattributef64 ( ) [private]
```

16.68.2.27 setglobalattributei16()

```
procedure, private mo_netcdf::ncdataset::setglobalattributei16 ( ) [private]
```

16.68.2.28 setglobalattributei32()

```
procedure, private mo_netcdf::ncdataset::setglobalattributei32 ( ) [private]
```

16.68.2.29 setglobalattributei8()

```
procedure, private mo_netcdf::ncdataset::setglobalattributei8 ( ) [private]
```

16.68.2.30 setvariable()

```
generic, public mo_netcdf::ncdataset::setvariable ( )
```

Create a NetCDF variable.

Create a NetCDF Variable with given name, data type and dimensions. All optional arguments to the `nf90_def_var` function are supported. The program will terminate abruptly if the variable cannot be created. Supported data types and their string encodings: `NF90_BYTE` -> "i8" `NF90_SHORT` -> "i16" `NF90_INT` -> "i32" `NF90_FLOAT` -> "f32" `NF90_DOUBLE` -> "f64"

Parameters

in	<i>character(*) :: name</i>	
in	<i>character(3) :: dtype</i>	
in	<i>integer(i4)/character(*)/type(NcDataset) :: dimensions(:)</i>	
in	<i>logical :: contiguous</i>	
in	<i>integer(i4) :: chunksize(:)</i>	
in	<i>integer(i4) :: deflate_level</i>	
in	<i>logical :: shuffle</i>	
in	<i>logical :: fletcher32</i>	
in	<i>integer(i4) :: endianess</i>	
in	<i>integer(i4) :: cache_size</i>	
in	<i>integer(i4) :: cache_nlems</i>	
in	<i>integer(i4) :: cache_preemption</i>	

Returns

NcVariable

Author

David Schaefer

Date

June 2015

16.68.2.31 setvariablewithids()

```
procedure, private mo_netcdf::ncdataset::setvariablewithids ( ) [private]
```

16.68.2.32 setvariablewithnames()

```
procedure, private mo_netcdf::ncdataset::setvariablewithnames ( ) [private]
```

16.68.2.33 setvariablewithtypes()

```
procedure, private mo_netcdf::ncdataset::setvariablewithtypes ( ) [private]
```

16.68.3 Member Data Documentation

16.68.3.1 dataset

`character(256) mo_netcdf::ncdataset::dataset`

16.68.3.2 file

`character(1) mo_netcdf::ncdataset::file`

16.68.3.3 filename

`character(256) mo_netcdf::ncdataset::filename`

16.68.3.4 fname

`character(256) mo_netcdf::ncdataset::fname`

16.68.3.5 id

`integer(i4) mo_netcdf::ncdataset::id`

16.68.3.6 mode

`character(1) mo_netcdf::ncdataset::mode`

16.68.3.7 netcdf

`integer(i4) mo_netcdf::ncdataset::netcdf`

16.68.3.8 of

`character(256) mo_netcdf::ncdataset::of`

16.68.3.9 open

```
character(1) mo_netcdf::ncdataset::open
```

16.68.3.10 opened

```
character(256) mo_netcdf::ncdataset::opened
```

16.68.3.11 the

```
character(256) mo_netcdf::ncdataset::the
```

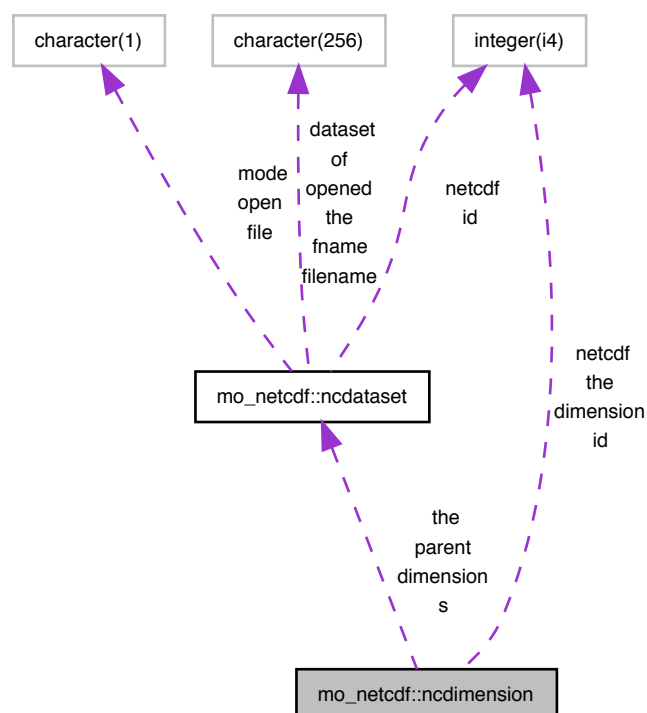
The documentation for this interface was generated from the following file:

- [mo_netcdf.f90](#)

16.69 mo_netcdf::ncdimension Type Reference

Provides the dimension access functionality.

Collaboration diagram for mo_netcdf::ncdimension:



Public Member Functions

- procedure, public `initncvariable`
- procedure, public `getname` => `getVariableName`
- procedure, public `getnodimensions`
 - Retrieve the number of dimensions.*
- procedure, public `getdimensions` => `getVariableDimensions`
 - Retrieve the variable dimensions.*
- procedure, public `getshape` => `getVariableShape`
 - Retrieve the shape of the variable.*
- procedure, public `getdtype` => `getVariableDtype`
 - Retrieve the variable data type.*
- procedure, public `hasattribute`
 - Check if attribute exists.*
- procedure, public `isunlimited` => `isUnlimitedVariable`
 - Check if the variable is unlimited.*
- generic, public `setdata` => `setDataScalarI8`, `setData1dI8`, `setData2dI8`, `setData3dI8`, `setData4dI8`, `setData5dI8`, `setDataScalarI16`, `setData1dI16`, `setData2dI16`, `setData3dI16`, `setData4dI16`, `setData5dI16`, `setDataScalarI32`, `setData1dI32`, `setData2dI32`, `setData3dI32`, `setData4dI32`, `setData5dI32`, `setDataScalarF32`, `setData1dF32`, `setData2dF32`, `setData3dF32`, `setData4dF32`, `setData5dF32`, `setDataScalarF64`, `setData1dF64`, `setData2dF64`, `setData3dF64`, `setData4dF64`, `setData5dF64`
 - Write data to variable.*
- generic, public `getdata` => `getDataScalarI8`, `getData1dI8`, `getData2dI8`, `getData3dI8`, `getData4dI8`, `getData5dI8`, `getDataScalarI16`, `getData1dI16`, `getData2dI16`, `getData3dI16`, `getData4dI16`, `getData5dI16`, `getDataScalarI32`, `getData1dI32`, `getData2dI32`, `getData3dI32`, `getData4dI32`, `getData5dI32`, `getDataScalarF32`, `getData1dF32`, `getData2dF32`, `getData3dF32`, `getData4dF32`, `getData5dF32`, `getDataScalarF64`, `getData1dF64`, `getData2dF64`, `getData3dF64`, `getData4dF64`, `getData5dF64`
 - Retrieve data.*
- generic, public `setfillvalue` => `setVariableFillValueI8`, `setVariableFillValueI16`, `setVariableFillValueI32`, `setVariableFillValueF32`, `setVariableFillValueF64`
 - Set the variable fill value.*
- generic, public `getfillvalue` => `getVariableFillValueI8`, `getVariableFillValueI16`, `getVariableFillValueI32`, `getVariableFillValueF32`, `getVariableFillValueF64`
 - Retrieve the variable fill value.*
- generic, public `setattribute` => `setVariableAttributeChar`, `setVariableAttributeI8`, `setVariableAttributeI16`, `setVariableAttributeI32`, `setVariableAttributeF32`, `setVariableAttributeF64`
 - Create a new variable attribute.*
- generic, public `getattribute` => `getVariableAttributeChar`, `getVariableAttributeI8`, `getVariableAttributeI16`, `getVariableAttributeI32`, `getVariableAttributeF32`, `getVariableAttributeF64`
 - Retrieve variable attribute value.*

Public Attributes

- integer(i4) `id`
- integer(i4) `the`
- integer(i4) `netcdf`
- integer(i4) `dimension`
- type(ncdataset) `parent`
- type(ncdataset) `the`
- type(ncdataset) `dimension`
- type(ncdataset) `s`

Private Member Functions

- procedure, private [setvariableattributechar](#)
- procedure, private [setvariableattributei8](#)
- procedure, private [setvariableattributei16](#)
- procedure, private [setvariableattributei32](#)
- procedure, private [setvariableattributef32](#)
- procedure, private [setvariableattributef64](#)
- procedure, private [getvariableattributechar](#)
- procedure, private [getvariableattributei8](#)
- procedure, private [getvariableattributei16](#)
- procedure, private [getvariableattributei32](#)
- procedure, private [getvariableattributef32](#)
- procedure, private [getvariableattributef64](#)
- procedure, private [setdatascalar_i8](#)
- procedure, private [setdata1di8](#)
- procedure, private [setdata2di8](#)
- procedure, private [setdata3di8](#)
- procedure, private [setdata4di8](#)
- procedure, private [setdata5di8](#)
- procedure, private [setdatascalar_i16](#)
- procedure, private [setdata1di16](#)
- procedure, private [setdata2di16](#)
- procedure, private [setdata3di16](#)
- procedure, private [setdata4di16](#)
- procedure, private [setdata5di16](#)
- procedure, private [setdatascalar_i32](#)
- procedure, private [setdata1di32](#)
- procedure, private [setdata2di32](#)
- procedure, private [setdata3di32](#)
- procedure, private [setdata4di32](#)
- procedure, private [setdata5di32](#)
- procedure, private [setdatascalar_f32](#)
- procedure, private [setdata1df32](#)
- procedure, private [setdata2df32](#)
- procedure, private [setdata3df32](#)
- procedure, private [setdata4df32](#)
- procedure, private [setdata5df32](#)
- procedure, private [setdatascalar_f64](#)
- procedure, private [setdata1df64](#)
- procedure, private [setdata2df64](#)
- procedure, private [setdata3df64](#)
- procedure, private [setdata4df64](#)
- procedure, private [setdata5df64](#)
- procedure, private [getdatascalar_i8](#)
- procedure, private [getdata1di8](#)
- procedure, private [getdata2di8](#)
- procedure, private [getdata3di8](#)
- procedure, private [getdata4di8](#)
- procedure, private [getdata5di8](#)
- procedure, private [getdatascalar_i16](#)
- procedure, private [getdata1di16](#)
- procedure, private [getdata2di16](#)
- procedure, private [getdata3di16](#)
- procedure, private [getdata4di16](#)

- procedure, private [getdata5di16](#)
- procedure, private [getdatascalari32](#)
- procedure, private [getdata1di32](#)
- procedure, private [getdata2di32](#)
- procedure, private [getdata3di32](#)
- procedure, private [getdata4di32](#)
- procedure, private [getdata5di32](#)
- procedure, private [getdatascalarf32](#)
- procedure, private [getdata1df32](#)
- procedure, private [getdata2df32](#)
- procedure, private [getdata3df32](#)
- procedure, private [getdata4df32](#)
- procedure, private [getdata5df32](#)
- procedure, private [getdatascalarf64](#)
- procedure, private [getdata1df64](#)
- procedure, private [getdata2df64](#)
- procedure, private [getdata3df64](#)
- procedure, private [getdata4df64](#)
- procedure, private [getdata5df64](#)
- procedure, private [setvariablefillvaluei8](#)
- procedure, private [setvariablefillvaluei16](#)
- procedure, private [setvariablefillvaluei32](#)
- procedure, private [setvariablefillvaluef32](#)
- procedure, private [setvariablefillvaluef64](#)
- procedure, private [getvariablefillvaluei8](#)
- procedure, private [getvariablefillvaluei16](#)
- procedure, private [getvariablefillvaluei32](#)
- procedure, private [getvariablefillvaluef32](#)
- procedure, private [getvariablefillvaluef64](#)

16.69.1 Detailed Description

Provides the dimension access functionality.

Bound to this derived type is some necessary inquire functionality. This type is not to be instantiated directly! Use the `getDimension/setDimension` functions of a `NcDataset` instance as a "constructor".

16.69.2 Member Function/Subroutine Documentation

16.69.2.1 `getattribute()`

```
generic, public mo_netcdf::ncdimension::getattribute ( )
```

Retrieve variable attribute value.

Retrieve the value for a variable attribute specified by its name. The program will terminate abruptly if the attribute does not exist.

Parameters

in	<i>character(*) :: name</i>	
out	<i>character(*)/integer(i4)/real(sp)/real(dp) :: value</i>	

Author

David Schaefer

Date

June 2015

16.69.2.2 getdata()

```
generic, public mo_netcdf::ncdimension::getdata ( )
```

Retrieve data.

Read the data from an optionally given position. All optional arguments to the `nf90_get_var` function are supported. A read error will result in abrupt program termination.

Parameters

in	<i>integer(i4) :: start(:), cnt(:), stride(:), map(:)</i>	
out	<i>"integer(i4)/real(sp)/real(dp),allocatable</i>	<i>:: & values/(:)/(:,:)/(::,)/(::,::)/(::,::,::)/(::,::,::,::)"</i>

Author

David Schaefer

Date

June 2015

16.69.2.3 getdata1df32()

```
procedure, private mo_netcdf::ncdimension::getdata1df32 ( ) [private]
```

16.69.2.4 getdata1df64()

```
procedure, private mo_netcdf::ncdimension::getdata1df64 ( ) [private]
```

16.69.2.5 getdata1di16()

```
procedure, private mo_netcdf::ncdimension::getdata1di16 ( ) [private]
```

16.69.2.6 getdata1di32()

```
procedure, private mo_netcdf::ncdimension::getdata1di32 ( ) [private]
```

16.69.2.7 getdata1di8()

```
procedure, private mo_netcdf::ncdimension::getdata1di8 ( ) [private]
```

16.69.2.8 getdata2df32()

```
procedure, private mo_netcdf::ncdimension::getdata2df32 ( ) [private]
```

16.69.2.9 getdata2df64()

```
procedure, private mo_netcdf::ncdimension::getdata2df64 ( ) [private]
```

16.69.2.10 getdata2di16()

```
procedure, private mo_netcdf::ncdimension::getdata2di16 ( ) [private]
```

16.69.2.11 getdata2di32()

```
procedure, private mo_netcdf::ncdimension::getdata2di32 ( ) [private]
```

16.69.2.12 getdata2di8()

```
procedure, private mo_netcdf::ncdimension::getdata2di8 ( ) [private]
```

16.69.2.13 getdata3df32()

```
procedure, private mo_netcdf::ncdimension::getdata3df32 ( ) [private]
```

16.69.2.14 getdata3df64()

```
procedure, private mo_netcdf::ncdimension::getdata3df64 ( ) [private]
```

16.69.2.15 getdata3di16()

```
procedure, private mo_netcdf::ncdimension::getdata3di16 ( ) [private]
```

16.69.2.16 getdata3di32()

```
procedure, private mo_netcdf::ncdimension::getdata3di32 ( ) [private]
```

16.69.2.17 getdata3di8()

```
procedure, private mo_netcdf::ncdimension::getdata3di8 ( ) [private]
```

16.69.2.18 getdata4df32()

```
procedure, private mo_netcdf::ncdimension::getdata4df32 ( ) [private]
```

16.69.2.19 getdata4df64()

```
procedure, private mo_netcdf::ncdimension::getdata4df64 ( ) [private]
```

16.69.2.20 getdata4di16()

```
procedure, private mo_netcdf::ncdimension::getdata4di16 ( ) [private]
```

16.69.2.21 getdata4di32()

```
procedure, private mo_netcdf::ncdimension::getdata4di32 ( ) [private]
```

16.69.2.22 getdata4di8()

```
procedure, private mo_netcdf::ncdimension::getdata4di8 ( ) [private]
```

16.69.2.23 getdata5df32()

```
procedure, private mo_netcdf::ncdimension::getdata5df32 ( ) [private]
```

16.69.2.24 getdata5df64()

```
procedure, private mo_netcdf::ncdimension::getdata5df64 ( ) [private]
```

16.69.2.25 getdata5di16()

```
procedure, private mo_netcdf::ncdimension::getdata5di16 ( ) [private]
```

16.69.2.26 getdata5di32()

```
procedure, private mo_netcdf::ncdimension::getdata5di32 ( ) [private]
```

16.69.2.27 getdata5di8()

```
procedure, private mo_netcdf::ncdimension::getdata5di8 ( ) [private]
```

16.69.2.28 getdatascalarf32()

```
procedure, private mo_netcdf::ncdimension::getdatascalarf32 ( ) [private]
```

16.69.2.29 getdatascalarf64()

```
procedure, private mo_netcdf::ncdimension::getdatascalarf64 ( ) [private]
```

16.69.2.30 getdatascalaril16()

```
procedure, private mo_netcdf::ncdimension::getdatascalaril16 ( ) [private]
```

16.69.2.31 getdatascalaril32()

```
procedure, private mo_netcdf::ncdimension::getdatascalaril32 ( ) [private]
```

16.69.2.32 getdatascalaril8()

```
procedure, private mo_netcdf::ncdimension::getdatascalaril8 ( ) [private]
```

16.69.2.33 getdimensions()

```
procedure, public mo_netcdf::ncdimension::getdimensions ( )
```

Retrieve the variable dimensions.

Return the ids of the dimensions associated with variable.

16.69.2.34 getdtype()

```
procedure, public mo_netcdf::ncdimension::getdtype ( )
```

Retrieve the variable data type.

Return the encoded data type of the variable. Data type encodeings "f32" -> NF90_FLOAT "f64" -> NF90_DOUBLE "i8" -> NF90_BYTE "i16" -> NF90_SHORT "i32" -> NF90_INT "i64" -> NF90_INT64

16.69.2.35 getfillvalue()

```
generic, public mo_netcdf::ncdimension::getfillvalue ( )
```

Retrieve the variable fill value.

Retrieve the variable fill value or a default value if fill value was not explicitly set. A read error results in abrupt program termination.

Parameters

out	<i>integer(i4)/real(sp)/real(dp) :: fvalue</i>	
-----	--	--

Author

David Schaefer

Date

June 2015

16.69.2.36 getname()

```
procedure, public mo_netcdf::ncdimension::getname ( )
```

16.69.2.37 getnodimensions()

```
procedure, public mo_netcdf::ncdimension::getnodimensions ( )
```

Retrieve the number of dimensions.

Return the number of dimensions associated with variable

16.69.2.38 getshape()

```
procedure, public mo_netcdf::ncdimension::getshape ( )
```

Retrieve the shape of the variable.

Return the shape of the variable.

16.69.2.39 getvariableattributechar()

```
procedure, private mo_netcdf::ncdimension::getvariableattributechar ( ) [private]
```

16.69.2.40 getvariableattribuf32()

```
procedure, private mo_netcdf::ncdimension::getvariableattribuf32 ( ) [private]
```

16.69.2.41 getvariableattribuf64()

```
procedure, private mo_netcdf::ncdimension::getvariableattribuf64 ( ) [private]
```

16.69.2.42 getvariableattributei16()

```
procedure, private mo_netcdf::ncdimension::getvariableattributei16 ( ) [private]
```

16.69.2.43 getvariableattributei32()

```
procedure, private mo_netcdf::ncdimension::getvariableattributei32 ( ) [private]
```

16.69.2.44 getvariableattributei8()

```
procedure, private mo_netcdf::ncdimension::getvariableattributei8 ( ) [private]
```

16.69.2.45 getvariablefillvaluef32()

```
procedure, private mo_netcdf::ncdimension::getvariablefillvaluef32 ( ) [private]
```

16.69.2.46 getvariablefillvaluef64()

```
procedure, private mo_netcdf::ncdimension::getvariablefillvaluef64 ( ) [private]
```

16.69.2.47 getvariablefillvaluei16()

```
procedure, private mo_netcdf::ncdimension::getvariablefillvaluei16 ( ) [private]
```

16.69.2.48 getvariablefillvaluei32()

```
procedure, private mo_netcdf::ncdimension::getvariablefillvaluei32 ( ) [private]
```

16.69.2.49 getvariablefillvaluei8()

```
procedure, private mo_netcdf::ncdimension::getvariablefillvaluei8 ( ) [private]
```

16.69.2.50 hasattribute()

```
procedure, public mo_netcdf::ncdimension::hasattribute ( )
```

Check if attribute exists.

Returns true if an attribute with the given name exists, false otherwise.

16.69.2.51 initncvariable()

```
procedure, public mo_netcdf::ncdimension::initncvariable ( )
```

16.69.2.52 isunlimited()

```
procedure, public mo_netcdf::ncdimension::isunlimited ( )
```

Check if the variable is unlimited.

Returns true if the variable has an unlimited dimension, false otherwise.

Returns

"logical"

Author

David Schaefer

Date

June 2015

16.69.2.53 setattribute()

```
generic, public mo_netcdf::ncdimension::setattribute ( )
```

Create a new variable attribute.

Create a new variable attribute from given name and value. A write error results in abrupt program termination.

Parameters

in	<i>character(*) :: name</i>	
in	<i>character(*)/integer(i4)/real(sp)/real(dp) :: value</i>	

Author

David Schaefer

Date

June 2015

16.69.2.54 setdata()

```
generic, public mo_netcdf::ncdimension::setdata ( )
```

Write data to variable.

Write the given data into the variable at an optionally given position. All optional arguments to the `nf90_put_var` function are supported. A write error will result in abrupt program termination.

Parameters

<code>in</code>	<code>"integer(i4)/real(sp)/real(dp)</code>	<code>:: & values/(:)/(:(:))/(:(:,:))/(:(:,:,:))/(:(:,:,:,:))" \param[in] "integer(i4) :: start(:), cnt(:), stride(:), map(:)"</code>
-----------------	---	---

Author

David Schaefer

Date

June 2015

16.69.2.55 setdata1df32()

```
procedure, private mo_netcdf::ncdimension::setdata1df32 ( ) [private]
```

16.69.2.56 setdata1df64()

```
procedure, private mo_netcdf::ncdimension::setdata1df64 ( ) [private]
```

16.69.2.57 setdata1di16()

```
procedure, private mo_netcdf::ncdimension::setdata1di16 ( ) [private]
```

16.69.2.58 setdata1di32()

```
procedure, private mo_netcdf::ncdimension::setdata1di32 ( ) [private]
```

16.69.2.59 setdata1di8()

```
procedure, private mo_netcdf::ncdimension::setdata1di8 ( ) [private]
```

16.69.2.60 setdata2df32()

```
procedure, private mo_netcdf::ncdimension::setdata2df32 ( ) [private]
```

16.69.2.61 setdata2df64()

```
procedure, private mo_netcdf::ncdimension::setdata2df64 ( ) [private]
```

16.69.2.62 setdata2di16()

```
procedure, private mo_netcdf::ncdimension::setdata2di16 ( ) [private]
```

16.69.2.63 setdata2di32()

```
procedure, private mo_netcdf::ncdimension::setdata2di32 ( ) [private]
```

16.69.2.64 setdata2di8()

```
procedure, private mo_netcdf::ncdimension::setdata2di8 ( ) [private]
```

16.69.2.65 setdata3df32()

```
procedure, private mo_netcdf::ncdimension::setdata3df32 ( ) [private]
```

16.69.2.66 setdata3df64()

```
procedure, private mo_netcdf::ncdimension::setdata3df64 ( ) [private]
```

16.69.2.67 setdata3di16()

```
procedure, private mo_netcdf::ncdimension::setdata3di16 ( ) [private]
```

16.69.2.68 setdata3di32()

```
procedure, private mo_netcdf::ncdimension::setdata3di32 ( ) [private]
```

16.69.2.69 setdata3di8()

```
procedure, private mo_netcdf::ncdimension::setdata3di8 ( ) [private]
```

16.69.2.70 setdata4df32()

```
procedure, private mo_netcdf::ncdimension::setdata4df32 ( ) [private]
```

16.69.2.71 setdata4df64()

```
procedure, private mo_netcdf::ncdimension::setdata4df64 ( ) [private]
```

16.69.2.72 setdata4di16()

```
procedure, private mo_netcdf::ncdimension::setdata4di16 ( ) [private]
```

16.69.2.73 setdata4di32()

```
procedure, private mo_netcdf::ncdimension::setdata4di32 ( ) [private]
```

16.69.2.74 setdata4di8()

```
procedure, private mo_netcdf::ncdimension::setdata4di8 ( ) [private]
```

16.69.2.75 setdata5df32()

```
procedure, private mo_netcdf::ncdimension::setdata5df32 ( ) [private]
```

16.69.2.76 setdata5df64()

```
procedure, private mo_netcdf::ncdimension::setdata5df64 ( ) [private]
```

16.69.2.77 setdata5di16()

```
procedure, private mo_netcdf::ncdimension::setdata5di16 ( ) [private]
```

16.69.2.78 setdata5di32()

```
procedure, private mo_netcdf::ncdimension::setdata5di32 ( ) [private]
```

16.69.2.79 setdata5di8()

```
procedure, private mo_netcdf::ncdimension::setdata5di8 ( ) [private]
```

16.69.2.80 setdatascalarf32()

```
procedure, private mo_netcdf::ncdimension::setdatascalarf32 ( ) [private]
```

16.69.2.81 setdatascalarf64()

```
procedure, private mo_netcdf::ncdimension::setdatascalarf64 ( ) [private]
```

16.69.2.82 setdatascalaril16()

```
procedure, private mo_netcdf::ncdimension::setdatascalaril16 ( ) [private]
```

16.69.2.83 setdatascalaril32()

```
procedure, private mo_netcdf::ncdimension::setdatascalaril32 ( ) [private]
```

16.69.2.84 setdatascalaril8()

```
procedure, private mo_netcdf::ncdimension::setdatascalaril8 ( ) [private]
```

16.69.2.85 setfillvalue()

```
generic, public mo_netcdf::ncdimension::setfillvalue ( )
```

Set the variable fill value.

Define the variable fill value. A write error results in abrupt program termination.

Note

This procedure must be called AFTER the variable was created but BEFORE data is first written.

Parameters

in	<i>integer(i4)/real(sp)/real(dp) :: fvalue</i>	
----	--	--

Author

David Schaefer

Date

June 2015

16.69.2.86 setvariableattributechar()

```
procedure, private mo_netcdf::ncdimension::setvariableattributechar ( ) [private]
```

16.69.2.87 setvariableattributef32()

```
procedure, private mo_netcdf::ncdimension::setvariableattributef32 ( ) [private]
```

16.69.2.88 setvariableattributef64()

```
procedure, private mo_netcdf::ncdimension::setvariableattributef64 ( ) [private]
```

16.69.2.89 setvariableattributei16()

```
procedure, private mo_netcdf::ncdimension::setvariableattributei16 ( ) [private]
```

16.69.2.90 setvariableattributei32()

```
procedure, private mo_netcdf::ncdimension::setvariableattributei32 ( ) [private]
```

16.69.2.91 setvariableattributei8()

```
procedure, private mo_netcdf::ncdimension::setvariableattributei8 ( ) [private]
```

16.69.2.92 setvariablefillvaluef32()

```
procedure, private mo_netcdf::ncdimension::setvariablefillvaluef32 ( ) [private]
```

16.69.2.93 setvariablefillvaluef64()

```
procedure, private mo_netcdf::ncdimension::setvariablefillvaluef64 ( ) [private]
```

16.69.2.94 setvariablefillvaluei16()

```
procedure, private mo_netcdf::ncdimension::setvariablefillvaluei16 ( ) [private]
```

16.69.2.95 setvariablefillvaluei32()

```
procedure, private mo_netcdf::ncdimension::setvariablefillvaluei32 ( ) [private]
```

16.69.2.96 setvariablefillvaluei8()

```
procedure, private mo_netcdf::ncdimension::setvariablefillvaluei8 ( ) [private]
```

16.69.3 Member Data Documentation**16.69.3.1 dimension** [1/2]

```
integer(i4) mo_netcdf::ncdimension::dimension
```

16.69.3.2 dimension [2/2]

```
type(ncdataset) mo_netcdf::ncdimension::dimension
```

16.69.3.3 id

```
integer(i4) mo_netcdf::ncdimension::id
```

16.69.3.4 netcdf

```
integer(i4) mo_netcdf::ncdimension::netcdf
```

16.69.3.5 parent

```
type(ncdataset) mo_netcdf::ncdimension::parent
```

16.69.3.6 s

```
type(ncdataset) mo_netcdf::ncdimension::s
```

16.69.3.7 the [1/2]

```
integer(i4) mo_netcdf::ncdimension::the
```

16.69.3.8 the [2/2]

```
type(ncdataset) mo_netcdf::ncdimension::the
```

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

- [mo_netcdf.f90](#)

16.70 mo_netcdf::ncvariable Interface Reference

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

- [mo_netcdf.f90](#)

16.71 mo_utils::ne Interface Reference**Public Member Functions**

- elemental pure logical function [notequal_sp](#) (a, b)
- elemental pure logical function [notequal_dp](#) (a, b)

16.71.1 Member Function/Subroutine Documentation**16.71.1.1 notequal_dp()**

```
elemental pure logical function mo_utils::ne::notequal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.71.1.2 notequal_sp()

```
elemental pure logical function mo_utils::ne::notequal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.72 mo_orderpack::nearless Interface Reference

Public Member Functions

- real(kind=dp) function [d_nearless](#) (XVAL)
- real(kind=sp) function [r_nearless](#) (XVAL)
- integer(kind=i4) function [i_nearless](#) (XVAL)

16.72.1 Member Function/Subroutine Documentation

16.72.1.1 d_nearless()

```
real(kind=dp) function mo_orderpack::nearless::d_nearless (
    real(kind=dp), intent(in) XVAL )
```

16.72.1.2 i_nearless()

```
integer(kind=i4) function mo_orderpack::nearless::i_nearless (
    integer(kind=i4), intent(in) XVAL )
```

16.72.1.3 r_nearless()

```
real(kind=sp) function mo_orderpack::nearless::r_nearless (
    real(kind=sp), intent(in) XVAL )
```

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

- [mo_orderpack.f90](#)

16.73 mo_spatialsimilarity::nndv Interface Reference

Calculates the number of neighboring dominating values, a measure for spatial dissimilarity.

Public Member Functions

- real(sp) function [nndv_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [nndv_dp](#) (mat1, mat2, mask, valid)

16.73.1 Detailed Description

Calculates the number of neighboring dominating values, a measure for spatial dissimilarity.

$$NNDV = 1 - \text{sum}(\text{abs}(\text{dominating_neighbors}(\text{mat1}) - \text{dominating_neighbors}(\text{mat2}))) / \text{count}(\text{mask})$$

dominating_neighbors(mat1) = comparison if pixel is larger than its neighbouring values

An array element value is compared with its 8 neighbouring cells to check if these cells are larger than the array element value. The result is a 3x3 matrix in which larger cells are indicated by a true value. For comparison this is done with both input arrays. The resulting array is the sum of subtraction of the 3x3 matrices for each of the both arrays. The resulting matrix is afterwards normalized to its available neighbors. Furthermore an average over the entire field is calculated. The valid interval of the values for NNDV is [0..1]. In which 1 indicates full agreement and 0 full mismatching.

EXAMPLE:~

```
mat1 = | 12 17  1 | , mat2 = |  7  9 12 |
      |  4 10 11 |           | 12 11 11 |
      | 15  2 20 |           |  5 13  7 |

booleans determined for every grid cell following fortran array scrolling
i.e. (/coll_row1, coll_row2, coll_row3, col2_row1, .. ,col3_row3/), (/3,3/)
```

```
comp1 = | FFF FFF FTF, FFF FFF FFF, FTT FFT FFF |
        | FFF TFT TTF, TFT TFF FTT, TFF FFT FFF |
        | FFF FFF FFF, TTF TFF TTF, FFF FFF FFF |
```

```
comp2 = | FFF FFT FTT, FFT FFT FTT, FFF FFF FFF |
        | FFF FFF FFT, FTF FFT TFF, FFT TFF FFF |
        | FFF TFF TTF, FFF FFF FFF, TTF TFF FFF |
```

```
NNDVMatrix =
abs( count(comp1) - count(comp2) ) = | 1-3, 0-4, 3-0 | = | 2, 4, 3 |
                                     | 4-1, 5-3, 2-2 |   | 3, 2, 0 |
                                     | 0-3, 5-0, 0-3 |   | 3, 5, 3 |
```

```
                                DISSIMILAR / VALID NEIGH CELLS
NNDVMatrix / VALID NEIGH CELLS = | 2, 4, 3 | / | 3, 5, 3 |
                                | 3, 2, 0 |   | 5, 8, 5 |
                                | 3, 5, 3 |   | 3, 5, 3 |

                                = | 0.66, 0.80, 1.00 |
                                | 0.60, 0.25, 0.00 |
                                | 1.00, 1.00, 1.00 |
```

```
NNDV = 1 - sum(NNDVMatrix) / count(mask) = 1 - (6.31666666 / 9) = 0.2981
```

If an optional mask is given, the calculations are over those locations that correspond to true values in the mask. mat1 and mat2 can be single or double precision. The result will have the same numerical precision.

Parameters

in	<i>real(sp/dp), dimension(:, :) :: mat1</i>	2D-array with input numbers
in	<i>real(sp/dp), dimension(:, :) :: mat2</i>	2D-array with input numbers
in	<i>logical, dimension(:, :), optional :: mask</i>	2D-array of logical values with size(mat1/mat2). If present, only those locations in mat1/mat2 having true values in mask are evaluated.
out	<i>logical :: valid</i>	indicates if the function could determine a valid value result can be invalid if entire mask is .false. for ex. in this case PatternDissim is set to 0 (worst case)

Returns

real(sp/dp) :: NNDV — Number of neighboring dominating values

Note

routine based on algorithm by Luis Samaniego 2009

Author

Matthias Zink

Date

Nov 2012

16.73.2 Member Function/Subroutine Documentation**16.73.2.1 nndv_dp()**

```
real(dp) function mo_spatialsimilarity::nndv::nndv_dp (
    real(dp), dimension(:,:), intent(in) mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

16.73.2.2 nndv_sp()

```
real(sp) function mo_spatialsimilarity::nndv::nndv_sp (
    real(sp), dimension(:,:), intent(in) mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

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

- [mo_spatialsimilarity.f90](#)

16.74 mo_utils::notequal Interface Reference**Public Member Functions**

- elemental pure logical function [notequal_sp](#) (a, b)
- elemental pure logical function [notequal_dp](#) (a, b)

16.74.1 Member Function/Subroutine Documentation

16.74.1.1 notequal_dp()

```
elemental pure logical function mo_utils::notequal::notequal_dp (
    real(dp), intent(in) a,
    real(dp), intent(in) b )
```

16.74.1.2 notequal_sp()

```
elemental pure logical function mo_utils::notequal::notequal_sp (
    real(sp), intent(in) a,
    real(sp), intent(in) b )
```

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

- [mo_utils.f90](#)

16.75 mo_errormeasures::nse Interface Reference

Public Member Functions

- real(sp) function [nse_sp_1d](#) (x, y, mask)
- real(dp) function [nse_dp_1d](#) (x, y, mask)
- real(dp) function [nse_dp_2d](#) (x, y, mask)
- real(sp) function [nse_sp_2d](#) (x, y, mask)
- real(sp) function [nse_sp_3d](#) (x, y, mask)
- real(dp) function [nse_dp_3d](#) (x, y, mask)

16.75.1 Member Function/Subroutine Documentation

16.75.1.1 nse_dp_1d()

```
real(dp) function mo_errormeasures::nse::nse_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.75.1.2 nse_dp_2d()

```
real(dp) function mo_errormeasures::nse::nse_dp_2d (
    real(dp), dimension(:, :), intent(in) x,
    real(dp), dimension(:, :), intent(in) y,
    logical, dimension(:, :), intent(in), optional mask )
```

16.75.1.3 nse_dp_3d()

```
real(dp) function mo_errormeasures::nse::nse_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

16.75.1.4 nse_sp_1d()

```
real(sp) function mo_errormeasures::nse::nse_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.75.1.5 nse_sp_2d()

```
real(sp) function mo_errormeasures::nse::nse_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.75.1.6 nse_sp_3d()

```
real(sp) function mo_errormeasures::nse::nse_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.76 mo_string_utils::num2str Interface Reference

Convert to string.

Public Member Functions

- pure character(len=10) function [i42str](#) (nn, form)
- pure character(len=20) function [i82str](#) (nn, form)
- pure character(len=32) function [sp2str](#) (rr, form)
- pure character(len=32) function [dp2str](#) (rr, form)
- pure character(len=10) function [log2str](#) (ll, form)

16.76.1 Detailed Description

Convert to string.

Convert a number or logical to a string with an optional format.

Parameters

in	<i>integer(i4/i8)/real(sp/dp)/logical :: num</i>	Number or logical
in	<i>character(len=*), optional :: form</i>	Format string Defaults are: i4 - 'I10' i8 - 'I20' sp/dp - 'G32.5' log - 'L10'

Returns

`character(len=X) :: str` — String of formatted input number or logical
 Output length X is:
 i4 - 10
 i8 - 20
 sp/dp - 32
 log - 10

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

Date

Dec 2011

16.76.2 Member Function/Subroutine Documentation

16.76.2.1 `dp2str()`

```
pure character(len=32) function mo_string_utils::num2str::dp2str (
    real(dp), intent(in) rr,
    character(len=*), intent(in), optional form )
```

16.76.2.2 `i42str()`

```
pure character(len=10) function mo_string_utils::num2str::i42str (
    integer(i4), intent(in) nn,
    character(len=*), intent(in), optional form )
```

16.76.2.3 `i82str()`

```
pure character(len=20) function mo_string_utils::num2str::i82str (
    integer(i8), intent(in) nn,
    character(len=*), intent(in), optional form )
```

16.76.2.4 log2str()

```
pure character(len=10) function mo_string_utils::num2str::log2str (
    logical, intent(in) ll,
    character(len=*), intent(in), optional form )
```

16.76.2.5 sp2str()

```
pure character(len=32) function mo_string_utils::num2str::sp2str (
    real(sp), intent(in) rr,
    character(len=*), intent(in), optional form )
```

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

- [mo_string_utils.f90](#)

16.77 mo_string_utils::numarray2str Interface Reference

Convert to string.

Public Member Functions

- character(len=size(arr)) function [i4array2str](#) (arr)

16.77.1 Detailed Description

Convert to string.

Convert a array of numbers or logicals to a string.

Parameters

in	<i>integer(i4/i8)/real(sp/dp)/logical :: num(:)</i>	Array of numbers or logicals
----	---	------------------------------

Returns

character(len=X) :: str — String of formatted input number or logical

Author

Matthias Cuntz - modified from Echam5, (C) MPI-MET, Hamburg, Germany

Date

Dec 2011

16.77.2 Member Function/Subroutine Documentation

16.77.2.1 i4array2str()

```
character(len=size(arr)) function mo_string_utils::numarray2str::i4array2str (
    integer(i4), dimension(:), intent(in) arr )
```

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

- [mo_string_utils.f90](#)

16.78 mo_orderpack::omedian Interface Reference

Public Member Functions

- real(kind=dp) function [d_median](#) (XDONT)
- real(kind=sp) function [r_median](#) (XDONT)
- integer(kind=i4) function [i_median](#) (XDONT)

16.78.1 Member Function/Subroutine Documentation

16.78.1.1 d_median()

```
real(kind=dp) function mo_orderpack::omedian::d_median (
    real(kind=dp), dimension (:), intent(in) XDONT )
```

16.78.1.2 i_median()

```
integer(kind=i4) function mo_orderpack::omedian::i_median (
    integer(kind=i4), dimension (:), intent(in) XDONT )
```

16.78.1.3 r_median()

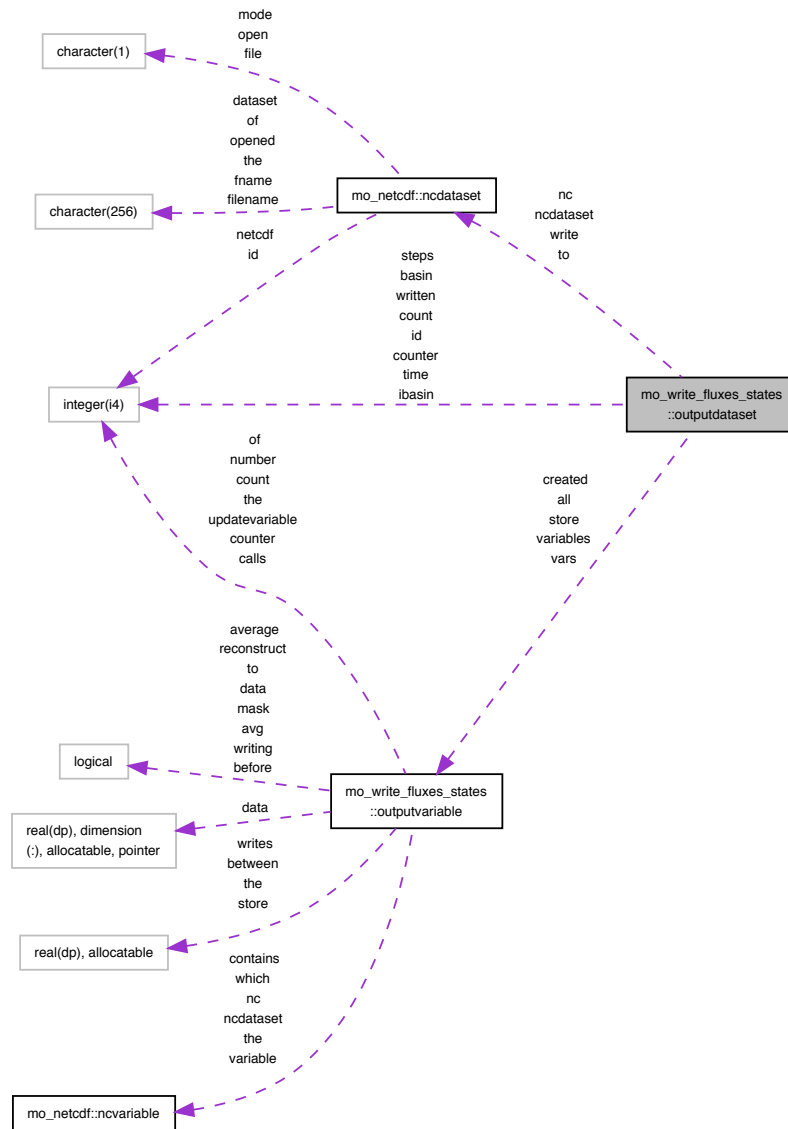
```
real(kind=sp) function mo_orderpack::omedian::r_median (
    real(kind=sp), dimension (:), intent(in) XDONT )
```

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

- [mo_orderpack.f90](#)

16.79 mo_write_fluxes_states::outputdataset Interface Reference

Collaboration diagram for mo_write_fluxes_states::outputdataset:



Public Member Functions

- procedure, public [updatedataset](#)
- procedure, public [writetimestep](#)
- procedure, public [close](#)

Public Attributes

- integer(i4) [ibasin](#)
- integer(i4) [basin](#)

- integer(i4) `id`
- type(ncdataset) `nc`
- type(ncdataset) `ncdataset`
- type(ncdataset) `to`
- type(ncdataset) `write`
- type(outputvariable), dimension(:), allocatable `vars`
- type(outputvariable), allocatable `store`
- type(outputvariable), allocatable `all`
- type(outputvariable), dimension(dynamic), allocatable `created`
- type(outputvariable), allocatable `variables`
- integer(i4) `counter` = 0
- integer(i4) `count`
- integer(i4) `written`
- integer(i4) `time`
- integer(i4) `steps`

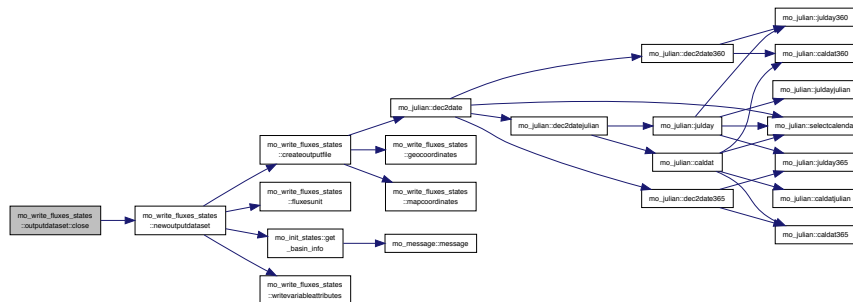
16.79.1 Member Function/Subroutine Documentation

16.79.1.1 `close()`

```
procedure, public mo_write_fluxes_states::outputdataset::close ( )
```

References `mo_write_fluxes_states::newoutputdataset()`.

Here is the call graph for this function:



16.79.1.2 `updatedataset()`

```
procedure, public mo_write_fluxes_states::outputdataset::updatedataset ( )
```

16.79.1.3 `writetimestep()`

```
procedure, public mo_write_fluxes_states::outputdataset::writetimestep ( )
```

16.79.2 Member Data Documentation

16.79.2.1 all

`type(outputvariable), allocatable mo_write_fluxes_states::outputdataset::all`

16.79.2.2 basin

`integer(i4) mo_write_fluxes_states::outputdataset::basin`

16.79.2.3 count

`integer(i4) mo_write_fluxes_states::outputdataset::count`

16.79.2.4 counter

`integer(i4) mo_write_fluxes_states::outputdataset::counter = 0`

16.79.2.5 created

`type(outputvariable), dimension (dynamic), allocatable mo_write_fluxes_states::outputdataset←
::created`

16.79.2.6 ibasin

`integer(i4) mo_write_fluxes_states::outputdataset::ibasin`

16.79.2.7 id

`integer(i4) mo_write_fluxes_states::outputdataset::id`

16.79.2.8 nc

`type(ncdataset) mo_write_fluxes_states::outputdataset::nc`

16.79.2.9 ncdataset

`type(ncdataset) mo_write_fluxes_states::outputdataset::ncdataset`

16.79.2.10 steps

`integer(i4) mo_write_fluxes_states::outputdataset::steps`

16.79.2.11 store

`type(outputvariable), allocatable mo_write_fluxes_states::outputdataset::store`

16.79.2.12 time

`integer(i4) mo_write_fluxes_states::outputdataset::time`

16.79.2.13 to

`type(ncdataset) mo_write_fluxes_states::outputdataset::to`

16.79.2.14 variables

`type(outputvariable), allocatable mo_write_fluxes_states::outputdataset::variables`

16.79.2.15 vars

`type(outputvariable), dimension(:), allocatable mo_write_fluxes_states::outputdataset::vars`

16.79.2.16 write

`type(ncdataset) mo_write_fluxes_states::outputdataset::write`

16.79.2.17 written

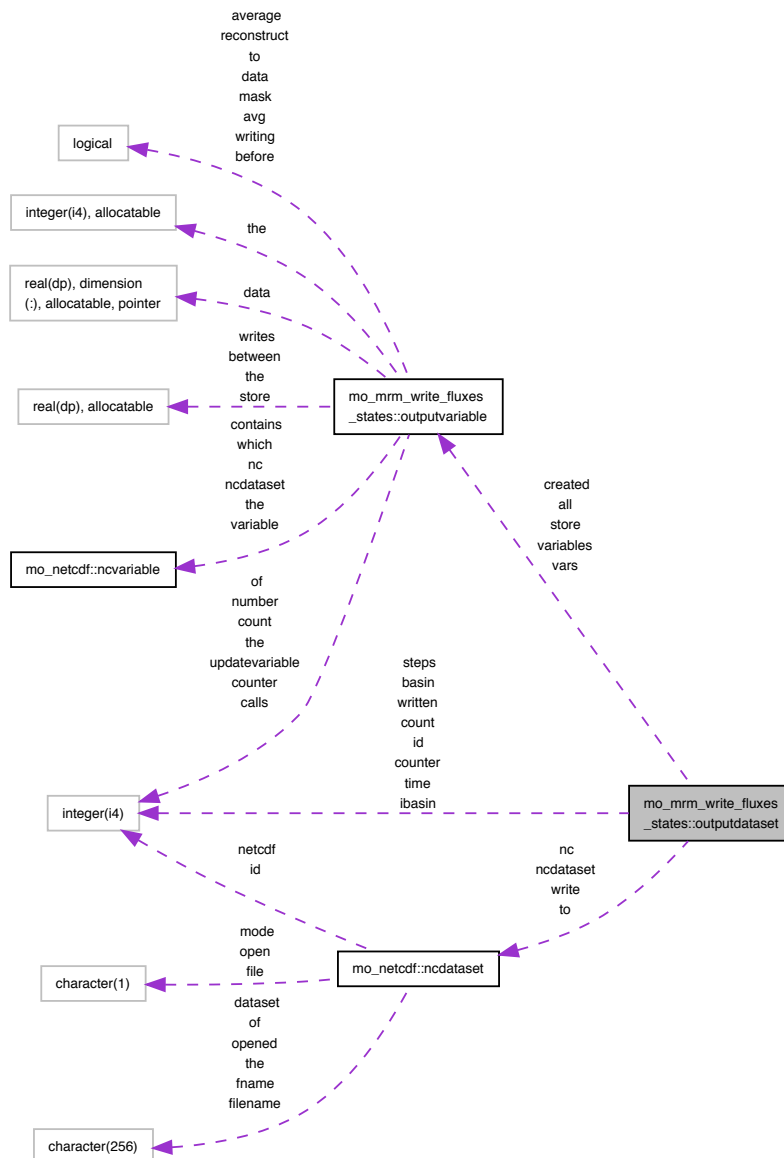
`integer(i4) mo_write_fluxes_states::outputdataset::written`

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

- [mo_write_fluxes_states.f90](#)

16.80 mo_mrm_write_fluxes_states::outputdataset Interface Reference

Collaboration diagram for mo_mrm_write_fluxes_states::outputdataset:



Public Member Functions

- procedure, public [updatedataset](#)
- procedure, public [writetimestep](#)
- procedure, public [close](#)

Public Attributes

- integer(i4) [ibasin](#)

- integer(i4) [basin](#)
- integer(i4) [id](#)
- type(ncdataset) [nc](#)
- type(ncdataset) [ncdataset](#)
- type(ncdataset) [to](#)
- type(ncdataset) [write](#)
- type(outputvariable), dimension(:), allocatable [vars](#)
- type(outputvariable), allocatable [store](#)
- type(outputvariable), allocatable [all](#)
- type(outputvariable), dimension(dynamic), allocatable [created](#)
- type(outputvariable), allocatable [variables](#)
- integer(i4) [counter](#) = 0
- integer(i4) [count](#)
- integer(i4) [written](#)
- integer(i4) [time](#)
- integer(i4) [steps](#)

16.80.1 Member Function/Subroutine Documentation

16.80.1.1 [close\(\)](#)

```
procedure, public mo_mrm_write_fluxes_states::outputdataset::close ( )
```

16.80.1.2 [updatedataset\(\)](#)

```
procedure, public mo_mrm_write_fluxes_states::outputdataset::updatedataset ( )
```

16.80.1.3 [writetimestep\(\)](#)

```
procedure, public mo_mrm_write_fluxes_states::outputdataset::writetimestep ( )
```

16.80.2 Member Data Documentation

16.80.2.1 [all](#)

```
type(outputvariable), allocatable mo_mrm_write_fluxes_states::outputdataset::all
```

16.80.2.2 [basin](#)

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::basin
```

16.80.2.3 count

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::count
```

16.80.2.4 counter

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::counter = 0
```

16.80.2.5 created

```
type(outputvariable), dimension (dynamic), allocatable mo_mrm_write_fluxes_states::outputdataset↵  
::created
```

16.80.2.6 ibasin

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::ibasin
```

16.80.2.7 id

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::id
```

16.80.2.8 nc

```
type(ncdataset) mo_mrm_write_fluxes_states::outputdataset::nc
```

16.80.2.9 ncdataset

```
type(ncdataset) mo_mrm_write_fluxes_states::outputdataset::ncdataset
```

16.80.2.10 steps

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::steps
```

16.80.2.11 store

```
type(outputvariable), allocatable mo_mrm_write_fluxes_states::outputdataset::store
```

16.80.2.12 time

```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::time
```

16.80.2.13 to

```
type(ncdataset) mo_mrm_write_fluxes_states::outputdataset::to
```

16.80.2.14 variables

```
type(outputvariable), allocatable mo_mrm_write_fluxes_states::outputdataset::variables
```

16.80.2.15 vars

```
type(outputvariable), dimension(:), allocatable mo_mrm_write_fluxes_states::outputdataset↵  
::vars
```

16.80.2.16 write

```
type(ncdataset) mo_mrm_write_fluxes_states::outputdataset::write
```

16.80.2.17 written

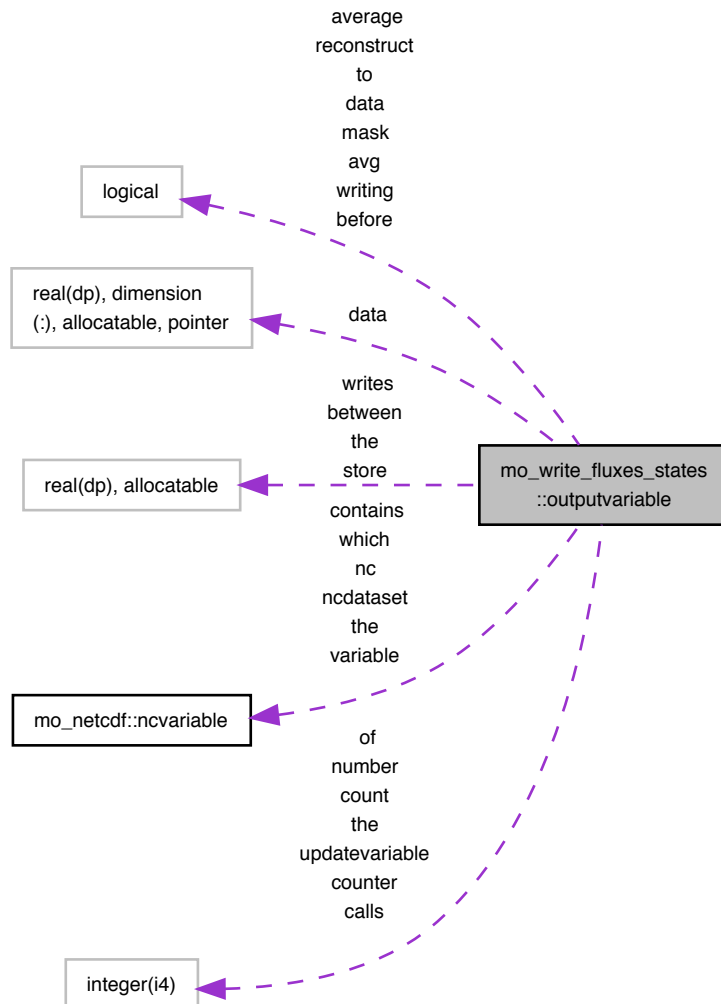
```
integer(i4) mo_mrm_write_fluxes_states::outputdataset::written
```

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

- [mo_mrm_write_fluxes_states.f90](#)

16.81 mo_write_fluxes_states::outputvariable Interface Reference

Collaboration diagram for mo_write_fluxes_states::outputvariable:



Public Member Functions

- procedure, public [updatevariable](#)
- procedure, public [writevariabletimestep](#)

Public Attributes

- type([ncvariable](#)) [nc](#)
- type([ncvariable](#)) [ncdataset](#)
- type([ncvariable](#)) [which](#)
- type([ncvariable](#)) [contains](#)
- type([ncvariable](#)), allocatable [the](#)

- type(ncvariable) [variable](#)
- logical [avg](#) = .false.
- logical [average](#)
- logical, dimension(:), allocatable, pointer [data](#)
- logical [before](#)
- logical [writing](#)
- logical, dimension(:, :), pointer [mask](#)
- logical, pointer [to](#)
- logical, pointer [reconstruct](#)
- real(dp), dimension(:), allocatable, pointer [data](#)
- real(dp), allocatable [store](#)
- real(dp), allocatable [the](#)
- real(dp), allocatable [between](#)
- real(dp), allocatable [writes](#)
- integer(i4) [counter](#) = 0
- integer(i4) [count](#)
- integer(i4), allocatable [the](#)
- integer(i4) [number](#)
- integer(i4) [of](#)
- integer(i4), public [updatevariable](#)
- integer(i4) [calls](#)

16.81.1 Member Function/Subroutine Documentation

16.81.1.1 updatevariable()

```
procedure, public mo_write_fluxes_states::outputvariable::updatevariable ( )
```

16.81.1.2 writevariabletimestep()

```
procedure, public mo_write_fluxes_states::outputvariable::writevariabletimestep ( )
```

16.81.2 Member Data Documentation

16.81.2.1 average

```
logical mo_write_fluxes_states::outputvariable::average
```

16.81.2.2 avg

```
logical mo_write_fluxes_states::outputvariable::avg = .false.
```

16.81.2.3 before

logical mo_write_fluxes_states::outputvariable::before

16.81.2.4 between

real(dp), allocatable mo_write_fluxes_states::outputvariable::between

16.81.2.5 calls

integer(i4) mo_write_fluxes_states::outputvariable::calls

16.81.2.6 contains

type(ncvariable) mo_write_fluxes_states::outputvariable::contains

16.81.2.7 count

integer(i4) mo_write_fluxes_states::outputvariable::count

16.81.2.8 counter

integer(i4) mo_write_fluxes_states::outputvariable::counter = 0

16.81.2.9 data [1/2]

logical, dimension(:), allocatable, pointer mo_write_fluxes_states::outputvariable::data

16.81.2.10 data [2/2]

real(dp), dimension(:), allocatable, pointer mo_write_fluxes_states::outputvariable::data

16.81.2.11 mask

logical, dimension(:, :), pointer mo_write_fluxes_states::outputvariable::mask

16.81.2.12 nc

```
type(ncvariable) mo_write_fluxes_states::outputvariable::nc
```

16.81.2.13 ncdataset

```
type(ncvariable) mo_write_fluxes_states::outputvariable::ncdataset
```

16.81.2.14 number

```
integer(i4) mo_write_fluxes_states::outputvariable::number
```

16.81.2.15 of

```
integer(i4) mo_write_fluxes_states::outputvariable::of
```

16.81.2.16 reconstruct

```
logical, pointer mo_write_fluxes_states::outputvariable::reconstruct
```

16.81.2.17 store

```
real(dp), allocatable mo_write_fluxes_states::outputvariable::store
```

16.81.2.18 the [1/3]

```
type(ncvariable), allocatable mo_write_fluxes_states::outputvariable::the
```

16.81.2.19 the [2/3]

```
real(dp), allocatable mo_write_fluxes_states::outputvariable::the
```

16.81.2.20 the [3/3]

```
integer(i4), allocatable mo_write_fluxes_states::outputvariable::the
```

16.81.2.21 to

```
logical, pointer mo_write_fluxes_states::outputvariable::to
```

16.81.2.22 updatevariable

```
integer(i4), public mo_write_fluxes_states::outputvariable::updatevariable
```

16.81.2.23 variable

```
type(ncvariable) mo_write_fluxes_states::outputvariable::variable
```

16.81.2.24 which

```
type(ncvariable) mo_write_fluxes_states::outputvariable::which
```

16.81.2.25 writes

```
real(dp), allocatable mo_write_fluxes_states::outputvariable::writes
```

16.81.2.26 writing

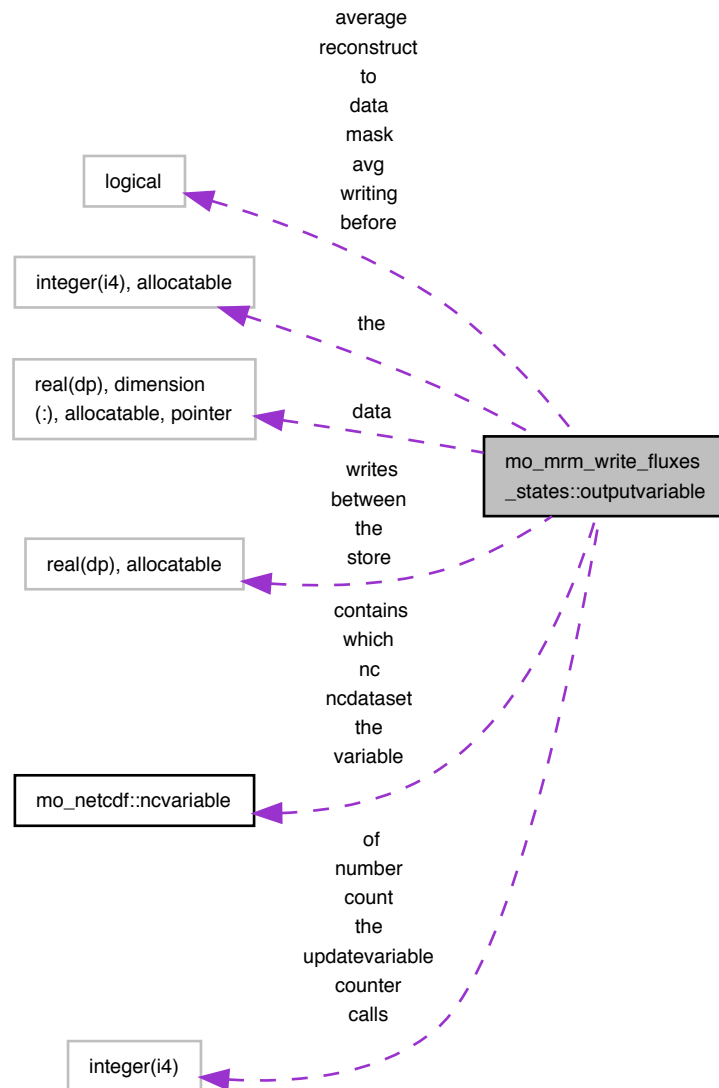
```
logical mo_write_fluxes_states::outputvariable::writing
```

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

- [mo_write_fluxes_states.f90](#)

16.82 mo_mrm_write_fluxes_states::outputvariable Interface Reference

Collaboration diagram for mo_mrm_write_fluxes_states::outputvariable:



Public Member Functions

- procedure, public [updatevariable](#)
- procedure, public [writevariabletimestep](#)

Public Attributes

- type([ncvariable](#)) [nc](#)
- type([ncvariable](#)) [ncdataset](#)

- type(ncvariable) which
- type(ncvariable) contains
- type(ncvariable), allocatable the
- type(ncvariable) variable
- logical avg = .false.
- logical average
- logical, dimension(:), allocatable, pointer data
- logical before
- logical writing
- logical, dimension(:, :), pointer mask
- logical, pointer to
- logical, pointer reconstruct
- real(dp), dimension(:), allocatable, pointer data
- real(dp), allocatable store
- real(dp), allocatable the
- real(dp), allocatable between
- real(dp), allocatable writes
- integer(i4) counter = 0
- integer(i4) count
- integer(i4), allocatable the
- integer(i4) number
- integer(i4) of
- integer(i4), public updatevariable
- integer(i4) calls

16.82.1 Member Function/Subroutine Documentation

16.82.1.1 updatevariable()

```
procedure, public mo_mrm_write_fluxes_states::outputvariable::updatevariable ( )
```

16.82.1.2 writevariabletimestep()

```
procedure, public mo_mrm_write_fluxes_states::outputvariable::writevariabletimestep ( )
```

16.82.2 Member Data Documentation

16.82.2.1 average

```
logical mo_mrm_write_fluxes_states::outputvariable::average
```

16.82.2.2 avg

```
logical mo_mrm_write_fluxes_states::outputvariable::avg = .false.
```

16.82.2.3 before

```
logical mo_mrm_write_fluxes_states::outputvariable::before
```

16.82.2.4 between

```
real(dp), allocatable mo_mrm_write_fluxes_states::outputvariable::between
```

16.82.2.5 calls

```
integer(i4) mo_mrm_write_fluxes_states::outputvariable::calls
```

16.82.2.6 contains

```
type(ncvariable) mo_mrm_write_fluxes_states::outputvariable::contains
```

16.82.2.7 count

```
integer(i4) mo_mrm_write_fluxes_states::outputvariable::count
```

16.82.2.8 counter

```
integer(i4) mo_mrm_write_fluxes_states::outputvariable::counter = 0
```

16.82.2.9 data [1/2]

```
logical, dimension(:), allocatable, pointer mo_mrm_write_fluxes_states::outputvariable::data
```

16.82.2.10 data [2/2]

```
real(dp), dimension(:), allocatable, pointer mo_mrm_write_fluxes_states::outputvariable::data
```

16.82.2.11 mask

```
logical, dimension(:, :), pointer mo_mrm_write_fluxes_states::outputvariable::mask
```

16.82.2.12 nc

```
type(ncvariable) mo_mrm_write_fluxes_states::outputvariable::nc
```

16.82.2.13 ncdataset

```
type(ncvariable) mo_mrm_write_fluxes_states::outputvariable::ncdataset
```

16.82.2.14 number

```
integer(i4) mo_mrm_write_fluxes_states::outputvariable::number
```

16.82.2.15 of

```
integer(i4) mo_mrm_write_fluxes_states::outputvariable::of
```

16.82.2.16 reconstruct

```
logical, pointer mo_mrm_write_fluxes_states::outputvariable::reconstruct
```

16.82.2.17 store

```
real(dp), allocatable mo_mrm_write_fluxes_states::outputvariable::store
```

16.82.2.18 the [1/3]

```
type(ncvariable), allocatable mo_mrm_write_fluxes_states::outputvariable::the
```

16.82.2.19 the [2/3]

```
real(dp), allocatable mo_mrm_write_fluxes_states::outputvariable::the
```

16.82.2.20 the [3/3]

```
integer(i4), allocatable mo_mrm_write_fluxes_states::outputvariable::the
```

16.82.2.21 to

logical, pointer `mo_mrm_write_fluxes_states::outputvariable::to`

16.82.2.22 updatevariable

integer(i4), public `mo_mrm_write_fluxes_states::outputvariable::updatevariable`

16.82.2.23 variable

type(ncvariable) `mo_mrm_write_fluxes_states::outputvariable::variable`

16.82.2.24 which

type(ncvariable) `mo_mrm_write_fluxes_states::outputvariable::which`

16.82.2.25 writes

real(dp), allocatable `mo_mrm_write_fluxes_states::outputvariable::writes`

16.82.2.26 writing

logical `mo_mrm_write_fluxes_states::outputvariable::writing`

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

- [mo_mrm_write_fluxes_states.f90](#)

16.83 mo_append::paste Interface Reference

Paste (columns) scalars, vectors, and matrixes onto existing array.

Public Member Functions

- subroutine [paste_i4_m_s](#) (mat1, sca2)
- subroutine [paste_i4_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_i8_m_s](#) (mat1, sca2)
- subroutine [paste_i8_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_sp_m_s](#) (mat1, sca2)
- subroutine [paste_sp_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_dp_m_s](#) (mat1, sca2)

- subroutine [paste_dp_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_char_m_s](#) (mat1, sca2)
- subroutine [paste_char_m_v](#) (mat1, vec2, fill_value)
- subroutine [paste_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [paste_lgt_m_s](#) (mat1, sca2)
- subroutine [paste_lgt_m_v](#) (mat1, vec2)
- subroutine [paste_lgt_m_m](#) (mat1, mat2)

16.83.1 Detailed Description

Paste (columns) scalars, vectors, and matrixes onto existing array.

Pastes one input to the columns of another, i.e. append on the second dimension.

The input might be a scalar, a vector or a matrix.

Possibilities are:

- (1) paste scalar to one-line matrix
- (3) paste vector to a matrix
- (5) paste matrix to matrix

Parameters

in	<i>input2</i>	values to paste. Can be INTEGER(I4/I8), REAL(SP/DP), or CHARACTER(len=*) and also scalar, DIMENSION(:), or DIMENSION(:,:) If not scalar then the rows have to agree with input1
in, out	<i>allocatable :: input1</i>	array to be pasted to. Can be INTEGER(I4/I8), REAL(SP/DP), or CHARACTER(len=*). Must be DIMENSION(:) or DIMENSION(:,:), and allocatable. If input2 is not scalar then it must be size(input1,1) = size(input2,1).

Author

Juliane Mai

Date

Aug 2012

16.83.2 Member Function/Subroutine Documentation

16.83.2.1 paste_char_m_m()

```
subroutine mo_append::paste::paste_char_m_m (
    character(len=*) , dimension(:, :), intent(inout), allocatable mat1,
    character(len=*) , dimension(:, :), intent(in) mat2,
    character(len=*) , intent(in), optional fill_value )
```

16.83.2.2 paste_char_m_s()

```
subroutine mo_append::paste::paste_char_m_s (
    character(len=*) , dimension(:, :), intent(inout), allocatable mat1,
    character(len=*) , intent(in) sca2 )
```

16.83.2.3 paste_char_m_v()

```

subroutine mo_append::paste::paste_char_m_v (
    character(len=*), dimension(:,:), intent(inout), allocatable mat1,
    character(len=*), dimension(:), intent(in) vec2,
    character(len=*), intent(in), optional fill_value )

```

16.83.2.4 paste_dp_m_m()

```

subroutine mo_append::paste::paste_dp_m_m (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    real(dp), intent(in), optional fill_value )

```

16.83.2.5 paste_dp_m_s()

```

subroutine mo_append::paste::paste_dp_m_s (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), intent(in) sca2 )

```

16.83.2.6 paste_dp_m_v()

```

subroutine mo_append::paste::paste_dp_m_v (
    real(dp), dimension(:,:), intent(inout), allocatable mat1,
    real(dp), dimension(:), intent(in) vec2,
    real(dp), intent(in), optional fill_value )

```

16.83.2.7 paste_i4_m_m()

```

subroutine mo_append::paste::paste_i4_m_m (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), dimension(:,:), intent(in) mat2,
    integer(i4), intent(in), optional fill_value )

```

16.83.2.8 paste_i4_m_s()

```

subroutine mo_append::paste::paste_i4_m_s (
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,
    integer(i4), intent(in) sca2 )

```

16.83.2.9 paste_i4_m_v()

```
subroutine mo_append::paste::paste_i4_m_v (  
    integer(i4), dimension(:,:), intent(inout), allocatable mat1,  
    integer(i4), dimension(:), intent(in) vec2,  
    integer(i4), intent(in), optional fill_value )
```

16.83.2.10 paste_i8_m_m()

```
subroutine mo_append::paste::paste_i8_m_m (  
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,  
    integer(i8), dimension(:,:), intent(in) mat2,  
    integer(i8), intent(in), optional fill_value )
```

16.83.2.11 paste_i8_m_s()

```
subroutine mo_append::paste::paste_i8_m_s (  
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,  
    integer(i8), intent(in) sca2 )
```

16.83.2.12 paste_i8_m_v()

```
subroutine mo_append::paste::paste_i8_m_v (  
    integer(i8), dimension(:,:), intent(inout), allocatable mat1,  
    integer(i8), dimension(:), intent(in) vec2,  
    integer(i8), intent(in), optional fill_value )
```

16.83.2.13 paste_lgt_m_m()

```
subroutine mo_append::paste::paste_lgt_m_m (  
    logical, dimension(:,:), intent(inout), allocatable mat1,  
    logical, dimension(:,:), intent(in) mat2 )
```

16.83.2.14 paste_lgt_m_s()

```
subroutine mo_append::paste::paste_lgt_m_s (  
    logical, dimension(:,:), intent(inout), allocatable mat1,  
    logical, intent(in) sca2 )
```

16.83.2.15 paste_lgt_m_v()

```
subroutine mo_append::paste::paste_lgt_m_v (  
    logical, dimension(:,:), intent(inout), allocatable mat1,  
    logical, dimension(:), intent(in) vec2 )
```

16.83.2.16 paste_sp_m_m()

```
subroutine mo_append::paste::paste_sp_m_m (
    real(sp), dimension(:, :), intent(inout), allocatable mat1,
    real(sp), dimension(:, :), intent(in) mat2,
    real(sp), intent(in), optional fill_value )
```

16.83.2.17 paste_sp_m_s()

```
subroutine mo_append::paste::paste_sp_m_s (
    real(sp), dimension(:, :), intent(inout), allocatable mat1,
    real(sp), intent(in) sca2 )
```

16.83.2.18 paste_sp_m_v()

```
subroutine mo_append::paste::paste_sp_m_v (
    real(sp), dimension(:, :), intent(inout), allocatable mat1,
    real(sp), dimension(:), intent(in) vec2,
    real(sp), intent(in), optional fill_value )
```

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

- [mo_append.f90](#)

16.84 mo_spatialsimilarity::pd Interface Reference

Calculates pattern dissimilarity (PD) measure.

Public Member Functions

- real(sp) function [pd_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [pd_dp](#) (mat1, mat2, mask, valid)

16.84.1 Detailed Description

Calculates pattern dissimilarity (PD) measure.

$PD = 1 - \text{sum}(\text{dissimilarity}(\text{mat1}, \text{mat2})) / \text{count}(\text{mask})$ $\text{dissimilarity}(\text{mat1}, \text{mat2}) = \text{comparison if pixel is larger than its neighbouring values}$

An array element value is compared with its 8 neighbouring cells to check if these cells are larger than the array element value. The result is a 3x3 matrix in which larger cells are indicated by a true value. For comparison this is done with both input arrays. The resulting array is the sum of xor values of the 3x3 matrices for each of the both arrays. This means only neighbourhood comparisons which are different in the 2 matrices are counted. This resulting matrix is afterwards normalized to its available neighbors. Furthermore an average over the entire field is calculated. The valid interval of the values for PD is [0..1]. In which 1 indicates full agreement and 0 full mismatching.

EXAMPLE:~

```

mat1 = | 12 17 1 | , mat2 = | 7 9 12 |
      | 4 10 11 |          | 12 11 11 |
      | 15 2 20 |          | 5 13 7 |

booleans determined for every grid cell following fortran array scrolling
i.e. (/col1_row1, col1_row2, col1_row3, col2_row1, .. ,col3_row3/), (/3,3/)

comp1 = | FFF FFF FTF, FFF FFF FFF, FTT FFT FFF |
        | FFF TFT TTF, TFT TFF FTT, TFF FFT FFF |
        | FFF FFF FFF, TTF TFF TTF, FFF FFF FFF |

comp2 = | FFF FFT FTT, FFT FFT FTT, FFF FFF FFF |
        | FFF FFF FFT, FTF FFT TFF, FFT TFF FFF |
        | FFF TFF TTF, FFF FFF FFF, TTF TFF FFF |

xor=neq = | FFF FFT FFT, FFT FFT FTT, FTT FFT FFF |
          | FFF TFT TTT, TTT TFT TTT, TFT TFT FFF |
          | FFF TFF TTF, TTF TFF TTF, TTF TFF FFF |

          DISSIMILAR / VALID NEIGH CELLS
PDMatrix = | 2, 4, 3 | / | 3, 5, 3 | = | 0.66, 0.80, 1.00 |
          | 5, 8, 4 |   | 5, 8, 5 |   | 1.00, 1.00, 0.80 |
          | 3, 5, 3 |   | 3, 5, 3 |   | 1.00, 1.00, 1.00 |

PD = 1 - sum(PDMatrix) / count(mask) = 1 - (8.2666666 / 9) = 0.08148

```

If an optional mask is given, the calculations are over those locations that correspond to true values in the mask. mat1 and mat2 can be single or double precision. The result will have the same numerical precision.

Parameters

in	<i>real(sp/dp), dimension(:,) :: mat1</i>	2D-array with input numbers
in	<i>real(sp/dp), dimension(:,) :: mat2</i>	2D-array with input numbers
in	<i>logical,dimension(:,),optional :: mask</i>	2D-array of logical values with size(mat1/mat2) If present, only those locations in mat1/mat2 having true values in mask are evaluated.
out	<i>logical,optional :: valid</i>	indicates if the function could determine a valid value result can be invalid if entire mask is .false. for ex. in this case PD is set to 0 (worst case)

Returns

real(sp/dp) :: PD — pattern dissimilarity measure

Author

Matthias Zink and Juliane Mai

Date

Jan 2013

16.84.2 Member Function/Subroutine Documentation

16.84.2.1 pd_dp()

```
real(dp) function mo_spatialsimilarity::pd::pd_dp (
    real(dp), dimension(:,:), intent(in) mat1,
    real(dp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

16.84.2.2 pd_sp()

```
real(sp) function mo_spatialsimilarity::pd::pd_sp (
    real(sp), dimension(:,:), intent(in) mat1,
    real(sp), dimension(:,:), intent(in) mat2,
    logical, dimension(:,:), intent(in), optional mask,
    logical, intent(out), optional valid )
```

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

- [mo_spatialsimilarity.f90](#)

16.85 mo_percentile::percentile Interface Reference**Public Member Functions**

- real(sp) function [percentile_0d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function [percentile_0d_dp](#) (arrin, k, mask, mode_in)
- real(sp) function, dimension(size(k)) [percentile_1d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function, dimension(size(k)) [percentile_1d_dp](#) (arrin, k, mask, mode_in)

16.85.1 Member Function/Subroutine Documentation**16.85.1.1 percentile_0d_dp()**

```
real(dp) function mo_percentile::percentile::percentile_0d_dp (
    real(dp), dimension(:), intent(in) arrin,
    real(dp), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in )
```

16.85.1.2 percentile_0d_sp()

```
real(sp) function mo_percentile::percentile::percentile_0d_sp (
    real(sp), dimension(:), intent(in) arrin,
    real(sp), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in )
```

16.85.1.3 percentile_1d_dp()

```
real(dp) function, dimension(size(k)) mo_percentile::percentile::percentile_1d_dp (
    real(dp), dimension(:), intent(in) arrin,
    real(dp), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in )
```

16.85.1.4 percentile_1d_sp()

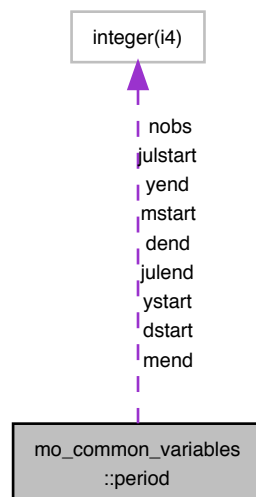
```
real(sp) function, dimension(size(k)) mo_percentile::percentile::percentile_1d_sp (
    real(sp), dimension(:), intent(in) arrin,
    real(sp), dimension(:), intent(in) k,
    logical, dimension(:), intent(in), optional mask,
    integer(i4), intent(in), optional mode_in )
```

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

- [mo_percentile.f90](#)

16.86 mo_common_variables::period Type Reference

Collaboration diagram for mo_common_variables::period:



Public Attributes

- integer(i4) [dstart](#)
- integer(i4) [mstart](#)
- integer(i4) [ystart](#)
- integer(i4) [dend](#)

- integer(i4) [mend](#)
- integer(i4) [yend](#)
- integer(i4) [julstart](#)
- integer(i4) [julend](#)
- integer(i4) [nobs](#)

16.86.1 Member Data Documentation

16.86.1.1 dend

integer(i4) mo_common_variables::period::dend

16.86.1.2 dstart

integer(i4) mo_common_variables::period::dstart

16.86.1.3 julend

integer(i4) mo_common_variables::period::julend

16.86.1.4 julstart

integer(i4) mo_common_variables::period::julstart

16.86.1.5 mend

integer(i4) mo_common_variables::period::mend

16.86.1.6 mstart

integer(i4) mo_common_variables::period::mstart

16.86.1.7 nobs

integer(i4) mo_common_variables::period::nobs

16.86.1.8 yend

```
integer(i4) mo_common_variables::period::yend
```

16.86.1.9 ystart

```
integer(i4) mo_common_variables::period::ystart
```

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

- [mo_common_variables.f90](#)

16.87 mo_percentile::qmedian Interface Reference

Public Member Functions

- real(sp) function [qmedian_sp](#) (dat)
- real(dp) function [qmedian_dp](#) (dat)

16.87.1 Member Function/Subroutine Documentation

16.87.1.1 qmedian_dp()

```
real(dp) function mo_percentile::qmedian::qmedian_dp (
    real(dp), dimension(:), intent(inout) dat )
```

16.87.1.2 qmedian_sp()

```
real(sp) function mo_percentile::qmedian::qmedian_sp (
    real(sp), dimension(:), intent(inout) dat )
```

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

- [mo_percentile.f90](#)

16.88 mo_orderpack::rapknr Interface Reference

Public Member Functions

- subroutine [d_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine [r_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine [i_rapknr](#) (XDONT, IRNGT, NORD)

16.88.1 Member Function/Subroutine Documentation

16.88.1.1 d_rapknr()

```

subroutine mo_orderpack::rapknr::d_rapknr (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )

```

16.88.1.2 i_rapknr()

```

subroutine mo_orderpack::rapknr::i_rapknr (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )

```

16.88.1.3 r_rapknr()

```

subroutine mo_orderpack::rapknr::r_rapknr (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )

```

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

- [mo_orderpack.f90](#)

16.89 mo_read_spatial_data::read_spatial_data_ascii Interface Reference

Reads spatial data files of ASCII format.

Public Member Functions

- subroutine [read_spatial_data_ascii_i4](#) (filename, fileunit, header_ncols, header_nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)
- subroutine [read_spatial_data_ascii_dp](#) (filename, fileunit, header_ncols, header_nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)

16.89.1 Detailed Description

Reads spatial data files of ASCII format.

Reads spatial input data, e.g. dem, aspect, flow direction.

Parameters

in	<i>character(len=*) :: filename</i>	Name of file and its location
in	<i>integer(i4) :: fileunit</i>	File unit for open file
in	<i>integer(i4) :: header_ncols</i>	Reference number of columns
in	<i>integer(i4) :: header_nrows</i>	Reference number of rows
in	<i>real(dp) :: header_xllcorner</i>	Reference lower left corner (x)
in	<i>real(dp) :: header_yllcorner</i>	Reference lower left corner (y)

Parameters

in	<i>integer(i4) :: header_cellsize</i>	Reference cell size [m]
out	<i>real(i4/dp), dimension(:, :) :: data</i>	Data matrix dim_1 = longitude, dim_2 = latitude
out	<i>logical, dimension(:, :) :: mask</i>	Mask of data matrix dim_1 = longitude, dim_2 = latitude

Author

Juliane Mai

Date

Jan 2013

16.89.2 Member Function/Subroutine Documentation

16.89.2.1 read_spatial_data_ascii_dp()

```

subroutine mo_read_spatial_data::read_spatial_data_ascii::read_spatial_data_ascii_dp (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(in) header_ncols,
    integer(i4), intent(in) header_nrows,
    real(dp), intent(in) header_xllcorner,
    real(dp), intent(in) header_yllcorner,
    real(dp), intent(in) header_cellsize,
    real(dp), dimension(:, :), intent(out), allocatable data,
    logical, dimension(:, :), intent(out), allocatable mask )

```

16.89.2.2 read_spatial_data_ascii_i4()

```

subroutine mo_read_spatial_data::read_spatial_data_ascii::read_spatial_data_ascii_i4 (
    character(len=*), intent(in) filename,
    integer(i4), intent(in) fileunit,
    integer(i4), intent(in) header_ncols,
    integer(i4), intent(in) header_nrows,
    real(dp), intent(in) header_xllcorner,
    real(dp), intent(in) header_yllcorner,
    real(dp), intent(in) header_cellsize,
    integer(i4), dimension(:, :), intent(out), allocatable data,
    logical, dimension(:, :), intent(out), allocatable mask )

```

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

- [mo_read_spatial_data.f90](#)

16.90 mo_corr::realft Interface Reference

Private Member Functions

- subroutine [realft_sp](#) (data, isign, zdata)

- subroutine [realft_dp](#) (data, isign, zdata)

16.90.1 Member Function/Subroutine Documentation

16.90.1.1 [realft_dp\(\)](#)

```
subroutine mo_corr::realft::realft_dp (
    real(dp), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign,
    complex(dpc), dimension(:), optional, target zdata ) [private]
```

16.90.1.2 [realft_sp\(\)](#)

```
subroutine mo_corr::realft::realft_sp (
    real(sp), dimension(:), intent(inout) data,
    integer(i4), intent(in) isign,
    complex(spc), dimension(:), optional, target zdata ) [private]
```

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

- [mo_corr.f90](#)

16.91 [mo_orderpack::refpar](#) Interface Reference

Public Member Functions

- subroutine [d_refpar](#) (XDONT, IRNGT, NORD)
- subroutine [r_refpar](#) (XDONT, IRNGT, NORD)
- subroutine [i_refpar](#) (XDONT, IRNGT, NORD)

16.91.1 Member Function/Subroutine Documentation

16.91.1.1 [d_refpar\(\)](#)

```
subroutine mo_orderpack::refpar::d_refpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.91.1.2 [i_refpar\(\)](#)

```
subroutine mo_orderpack::refpar::i_refpar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.91.1.3 r_refpar()

```
subroutine mo_orderpack::refpar::r_refpar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.92 mo_orderpack::refsr Interface Reference

Public Member Functions

- subroutine [d_refsr](#) (XDONT)
- subroutine [r_refsr](#) (XDONT)
- subroutine [i_refsr](#) (XDONT)

16.92.1 Member Function/Subroutine Documentation

16.92.1.1 d_refsr()

```
subroutine mo_orderpack::refsr::d_refsr (
    real(kind=dp), dimension (:), intent(inout) XDONT )
```

16.92.1.2 i_refsr()

```
subroutine mo_orderpack::refsr::i_refsr (
    integer(kind=i4), dimension (:), intent(inout) XDONT )
```

16.92.1.3 r_refsr()

```
subroutine mo_orderpack::refsr::r_refsr (
    real(kind=sp), dimension (:), intent(inout) XDONT )
```

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

- [mo_orderpack.f90](#)

16.93 mo_orderpack::rinpar Interface Reference

Public Member Functions

- subroutine [d_rinpar](#) (XDONT, IRNGT, NORD)

- subroutine [r_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine [i_rinpar](#) (XDONT, IRNGT, NORD)

16.93.1 Member Function/Subroutine Documentation

16.93.1.1 d_rinpar()

```
subroutine mo_orderpack::rinpar::d_rinpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.93.1.2 i_rinpar()

```
subroutine mo_orderpack::rinpar::i_rinpar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.93.1.3 r_rinpar()

```
subroutine mo_orderpack::rinpar::r_rinpar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.94 mo_errormasures::rmse Interface Reference

Public Member Functions

- real(sp) function [rmse_sp_1d](#) (x, y, mask)
- real(dp) function [rmse_dp_1d](#) (x, y, mask)
- real(sp) function [rmse_sp_2d](#) (x, y, mask)
- real(dp) function [rmse_dp_2d](#) (x, y, mask)
- real(sp) function [rmse_sp_3d](#) (x, y, mask)
- real(dp) function [rmse_dp_3d](#) (x, y, mask)

16.94.1 Member Function/Subroutine Documentation

16.94.1.1 rmse_dp_1d()

```
real(dp) function mo_errormeasures::rmse::rmse_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.94.1.2 rmse_dp_2d()

```
real(dp) function mo_errormeasures::rmse::rmse_dp_2d (  
    real(dp), dimension(:, :), intent(in) x,  
    real(dp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.94.1.3 rmse_dp_3d()

```
real(dp) function mo_errormeasures::rmse::rmse_dp_3d (  
    real(dp), dimension(:, :, :), intent(in) x,  
    real(dp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

16.94.1.4 rmse_sp_1d()

```
real(sp) function mo_errormeasures::rmse::rmse_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.94.1.5 rmse_sp_2d()

```
real(sp) function mo_errormeasures::rmse::rmse_sp_2d (  
    real(sp), dimension(:, :), intent(in) x,  
    real(sp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.94.1.6 rmse_sp_3d()

```
real(sp) function mo_errormeasures::rmse::rmse_sp_3d (  
    real(sp), dimension(:, :, :), intent(in) x,  
    real(sp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.95 mo_orderpack::rnkpar Interface Reference

Public Member Functions

- subroutine [d_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine [r_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine [i_rnkpar](#) (XDONT, IRNGT, NORD)

16.95.1 Member Function/Subroutine Documentation

16.95.1.1 d_rnkpar()

```
subroutine mo_orderpack::rnkpar::d_rnkpar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.95.1.2 i_rnkpar()

```
subroutine mo_orderpack::rnkpar::i_rnkpar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

16.95.1.3 r_rnkpar()

```
subroutine mo_orderpack::rnkpar::r_rnkpar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.96 mo_errormeasures::sae Interface Reference

Public Member Functions

- real(sp) function [sae_sp_1d](#) (x, y, mask)
- real(dp) function [sae_dp_1d](#) (x, y, mask)
- real(sp) function [sae_sp_2d](#) (x, y, mask)
- real(dp) function [sae_dp_2d](#) (x, y, mask)
- real(sp) function [sae_sp_3d](#) (x, y, mask)
- real(dp) function [sae_dp_3d](#) (x, y, mask)

16.96.1 Member Function/Subroutine Documentation

16.96.1.1 sae_dp_1d()

```
real(dp) function mo_errormeasures::sae::sae_dp_1d (  
    real(dp), dimension(:), intent(in) x,  
    real(dp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.96.1.2 sae_dp_2d()

```
real(dp) function mo_errormeasures::sae::sae_dp_2d (  
    real(dp), dimension(:, :), intent(in) x,  
    real(dp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.96.1.3 sae_dp_3d()

```
real(dp) function mo_errormeasures::sae::sae_dp_3d (  
    real(dp), dimension(:, :, :), intent(in) x,  
    real(dp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

16.96.1.4 sae_sp_1d()

```
real(sp) function mo_errormeasures::sae::sae_sp_1d (  
    real(sp), dimension(:), intent(in) x,  
    real(sp), dimension(:), intent(in) y,  
    logical, dimension(:), intent(in), optional mask )
```

16.96.1.5 sae_sp_2d()

```
real(sp) function mo_errormeasures::sae::sae_sp_2d (  
    real(sp), dimension(:, :), intent(in) x,  
    real(sp), dimension(:, :), intent(in) y,  
    logical, dimension(:, :), intent(in), optional mask )
```

16.96.1.6 sae_sp_3d()

```
real(sp) function mo_errormeasures::sae::sae_sp_3d (  
    real(sp), dimension(:, :, :), intent(in) x,  
    real(sp), dimension(:, :, :), intent(in) y,  
    logical, dimension(:, :, :), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.97 mo_julian::setcalendar Interface Reference

Private Member Functions

- subroutine [setcalendarinteger](#) (selector)
Set module private variable calendar.
- subroutine [setcalendarstring](#) (selector)
Set module private variable calendar.

16.97.1 Member Function/Subroutine Documentation

16.97.1.1 setcalendarinteger()

```
subroutine mo_julian::setcalendar::setcalendarinteger (
    integer(i4), intent(in) selector ) [private]
```

Set module private variable calendar.

Parameters

in	<i>integer(i4) :: selector</i>	{1 2 3}
----	--------------------------------	---------

Author

Written, David Schaefer

Date

Jan 2015

16.97.1.2 setcalendarstring()

```
subroutine mo_julian::setcalendar::setcalendarstring (
    character(*), intent(in) selector ) [private]
```

Set module private variable calendar.

Parameters

in	<i>character(len=*) :: selector</i>	{"julian" "365day" "360day"}
----	-------------------------------------	------------------------------

Author

Written, David Schaefer

Date

Jan 2015

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

- [mo_julian.f90](#)

16.98 mo_moment::skewness Interface Reference

Public Member Functions

- real(sp) function [skewness_sp](#) (dat, mask)
- real(dp) function [skewness_dp](#) (dat, mask)

16.98.1 Member Function/Subroutine Documentation

16.98.1.1 skewness_dp()

```
real(dp) function mo_moment::skewness::skewness_dp (  
    real(dp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

16.98.1.2 skewness_sp()

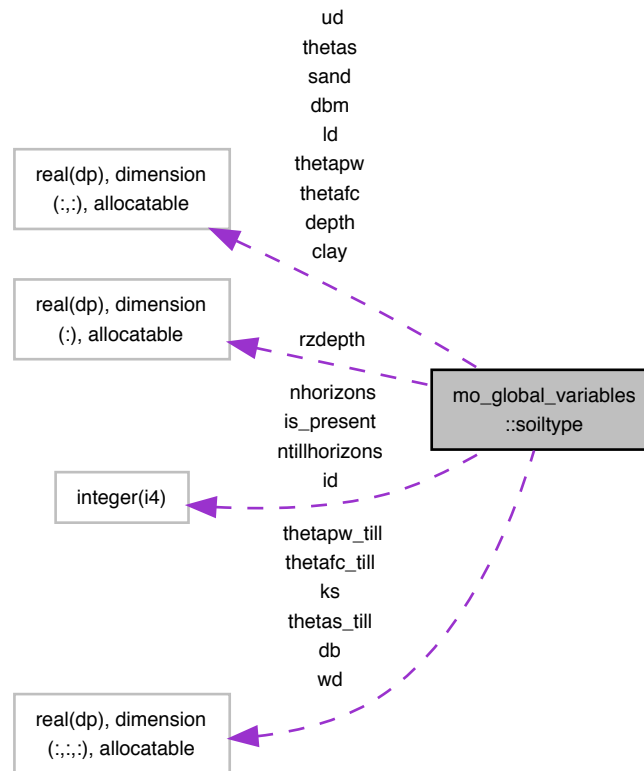
```
real(sp) function mo_moment::skewness::skewness_sp (  
    real(sp), dimension(:), intent(in) dat,  
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.99 mo_global_variables::soiltype Type Reference

Collaboration diagram for mo_global_variables::soiltype:



Public Attributes

- integer(i4), dimension(:), allocatable `id`
- integer(i4), dimension(:), allocatable `nhorizons`
- integer(i4), dimension(:), allocatable `is_present`
- real(dp), dimension(:, :), allocatable `ud`
- real(dp), dimension(:, :), allocatable `ld`
- real(dp), dimension(:, :), allocatable `clay`
- real(dp), dimension(:, :), allocatable `sand`
- real(dp), dimension(:, :), allocatable `dbm`
- real(dp), dimension(:, :), allocatable `depth`
- real(dp), dimension(:), allocatable `rzdepth`
- real(dp), dimension(:, :, :), allocatable `wd`
- integer(i4), dimension(:), allocatable `ntillhorizons`
- real(dp), dimension(:, :, :), allocatable `thetas_till`
- real(dp), dimension(:, :), allocatable `thetas`
- real(dp), dimension(:, :, :), allocatable `db`
- real(dp), dimension(:, :, :), allocatable `thetafc_till`
- real(dp), dimension(:, :), allocatable `thetafc`

- real(dp), dimension(:,:,:), allocatable [thetapw_till](#)
- real(dp), dimension(:,:), allocatable [thetapw](#)
- real(dp), dimension(:,:,:), allocatable [ks](#)

16.99.1 Member Data Documentation

16.99.1.1 clay

real(dp), dimension(:,:), allocatable mo_global_variables::soiltype::clay

16.99.1.2 db

real(dp), dimension(:,:,:), allocatable mo_global_variables::soiltype::db

16.99.1.3 dbm

real(dp), dimension(:,:), allocatable mo_global_variables::soiltype::dbm

16.99.1.4 depth

real(dp), dimension(:,:), allocatable mo_global_variables::soiltype::depth

16.99.1.5 id

integer(i4), dimension(:), allocatable mo_global_variables::soiltype::id

16.99.1.6 is_present

integer(i4), dimension(:), allocatable mo_global_variables::soiltype::is_present

16.99.1.7 ks

real(dp), dimension(:,:,:), allocatable mo_global_variables::soiltype::ks

16.99.1.8 ld

real(dp), dimension(:,:), allocatable mo_global_variables::soiltype::ld

16.99.1.9 nhorizons

`integer(i4), dimension(:), allocatable mo_global_variables::soiltype::nhorizons`

16.99.1.10 ntillhorizons

`integer(i4), dimension(:), allocatable mo_global_variables::soiltype::ntillhorizons`

16.99.1.11 rzdepth

`real(dp), dimension(:), allocatable mo_global_variables::soiltype::rzdepth`

16.99.1.12 sand

`real(dp), dimension(:, :), allocatable mo_global_variables::soiltype::sand`

16.99.1.13 thetafc

`real(dp), dimension(:, :), allocatable mo_global_variables::soiltype::thetafc`

16.99.1.14 thetafc_till

`real(dp), dimension(:, :, :), allocatable mo_global_variables::soiltype::thetafc_till`

16.99.1.15 thetapw

`real(dp), dimension(:, :), allocatable mo_global_variables::soiltype::thetapw`

16.99.1.16 thetapw_till

`real(dp), dimension(:, :, :), allocatable mo_global_variables::soiltype::thetapw_till`

16.99.1.17 thetas

`real(dp), dimension(:, :), allocatable mo_global_variables::soiltype::thetas`

16.99.1.18 thetas_till

```
real(dp), dimension(:,:,:), allocatable mo_global_variables::soiltype::thetas_till
```

16.99.1.19 ud

```
real(dp), dimension(:,:), allocatable mo_global_variables::soiltype::ud
```

16.99.1.20 wd

```
real(dp), dimension(:,:,:), allocatable mo_global_variables::soiltype::wd
```

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

- [mo_global_variables.f90](#)

16.100 mo_orderpack::sort Interface Reference

Unconditional ranking.

Public Member Functions

- subroutine [d_refsor](#) (XDONT)
- subroutine [r_refsor](#) (XDONT)
- subroutine [i_refsor](#) (XDONT)

16.100.1 Detailed Description

Unconditional ranking.

Subroutine MRGRNK (XVALT, IMULT) Ranks array XVALT into index array IRNGT, using merge-sort For performance reasons, the first 2 passes are taken out of the standard loop, and use dedicated coding.

Subroutine MRGREF (XVALT, IRNGT) Ranks array XVALT into index array IRNGT, using merge-sort This version is not optimized for performance, and is thus not as difficult to read as the previous one.

Partial ranking

Subroutine RNKPAR (XVALT, IRNGT, NORD) Ranks partially XVALT by IRNGT, up to order NORD (refined for speed) This routine uses a pivoting strategy such as the one of finding the median based on the quicksort algorithm, but we skew the pivot choice to try to bring it to NORD as fast as possible. It uses 2 temporary arrays, one where it stores the indices of the values smaller than the pivot, and the other for the indices of values larger than the pivot that we might still need later on. It iterates until it can bring the number of values in ILOWT to exactly NORD, and then uses an insertion sort to rank this set, since it is supposedly small.

Subroutine RAPKNR (XVALT, IRNGT, NORD) Same as RNKPAR, but in decreasing order (RAPKNR = RNKPAR spelt backwards).

Subroutine REFPAR (XVALT, IRNGT, NORD) Ranks partially XVALT by IRNGT, up to order NORD This version is not optimized for performance, and is thus not as difficult to read as some other ones. It uses a pivoting strategy such as the one of finding the median based on the quicksort algorithm. It uses a temporary array, where it stores the partially ranked indices of the values. It iterates until it can bring the number of values lower than the pivot to exactly NORD, and then uses an insertion sort to rank this set, since it is supposedly small.

Subroutine RINPAR (XVALT, IRNGT, NORD) Ranks partially XVALT by IRNGT, up to order NORD This version is not optimized for performance, and is thus not as difficult to read as some other ones. It uses insertion sort, limiting insertion to the first NORD values. It does not use any work array and is faster when NORD is very small (2-5), but worst case behavior (initially inverse sorted) can easily happen. In many cases, the refined quicksort method is faster.

Integer Function INDNTH (XVALT, NORD) Returns the index of the NORDth value of XVALT (in increasing order) This routine uses a pivoting strategy such as the one of finding the median based on the quicksort algorithm, but we skew the pivot choice to try to bring it to NORD as fast as possible. It uses 2 temporary arrays, one where it stores the indices of the values smaller than the pivot, and the other for the indices of values larger than the pivot that we might still need later on. It iterates until it can bring the number of values in ILOWT to exactly NORD, and then takes out the original index of the maximum value in this set.

Subroutine INDMED (XVALT, INDM) Returns the index of the median $((\text{Size}(\text{XVALT})+1)/2)$ th value of XVALT This routine uses the recursive procedure described in Knuth, The Art of Computer Programming, vol. 3, 5.3.3 - This procedure is linear in time, and does not require to be able to interpolate in the set as the one used in INDNTH. It also has better worst case behavior than INDNTH, but is about 10% slower in average for random uniformly distributed values.

Note that in Orderpack 1.0, this routine was a Function procedure, and is now changed to a Subroutine.

Unique ranking

Subroutine UNIRNK (XVALT, IRNGT, NUNI) Ranks an array, removing duplicate entries (uses merge sort). The routine is similar to pure merge-sort ranking, but on the last pass, it discards indices that correspond to duplicate entries. For performance reasons, the first 2 passes are taken out of the standard loop, and use dedicated coding.

Subroutine UNIPAR (XVALT, IRNGT, NORD) Ranks partially XVALT by IRNGT, up to order NORD at most, removing duplicate entries This routine uses a pivoting strategy such as the one of finding the median based on the quicksort algorithm, but we skew the pivot choice to try to bring it to NORD as quickly as possible. It uses 2 temporary arrays, one where it stores the indices of the values smaller than the pivot, and the other for the indices of values larger than the pivot that we might still need later on. It iterates until it can bring the number of values in ILOWT to exactly NORD, and then uses an insertion sort to rank this set, since it is supposedly small. At all times, the NORD first values in ILOWT correspond to distinct values of the input array.

Subroutine UNIINV (XVALT, IGOEST) Inverse ranking of an array, with removal of duplicate entries The routine is similar to pure merge-sort ranking, but on the last pass, it sets indices in IGOEST to the rank of the original value in an ordered set with duplicates removed. For performance reasons, the first 2 passes are taken out of the standard loop, and use dedicated coding.

Subroutine MULCNT (XVALT, IMULT) Gives, for each array value, its multiplicity The number of times that a value appears in the array is computed by using inverse ranking, counting for each rank the number of values that "collide" to this rank, and returning this sum to the locations in the original set. Uses subroutine UNIINV.

Random permutation: an interesting use of ranking

A variation of the following problem was raised on the internet sci.math.num-analysis news group: Given an array, I would like to find a random permutation of this array that I could control with a `nearbyness''` parameter so that elements stay close to their initial locations. The `nearbyness''` parameter ranges from 0 to 1, with 0 such that no element moves from its initial location, and 1 such that the permutation is fully random.

Subroutine CTRPER (XVALT, PCLS) Permute array XVALT randomly, but leaving elements close to their initial locations The routine takes the $1 \dots \text{size}(\text{XVALT})$ index array as real values, takes a combination of these values and of random values as a perturbation of the index array, and sorts the initial set according to the ranks of these perturbed indices. The relative proportion of initial order and random order is $1 - \text{PCLS} / \text{PCLS}$, thus when $\text{PCLS} = 0$, there is no change in the order whereas the new order is fully random when $\text{PCLS} = 1$. Uses subroutine MRGRNK.

The above solution found another application when I was asked the following question: I am given two arrays, representing parents' incomes and their children's incomes, but I do not know which parents correspond to which children. I know from an independent source the value of the correlation coefficient between the incomes of the parents and of their children. I would like to pair the elements of these arrays so that the given correlation coefficient is attained, i.e. to reconstruct a realistic dataset, though very likely not to be the true one.

Program GIVCOR Given two arrays of equal length of unordered values, find a "matching value" in the second array for each value in the first so that the global correlation coefficient reaches exactly a given target. The routine first sorts the two arrays, so as to get the match of maximum possible correlation. It then iterates, applying the random permutation algorithm of controlled disorder ctrper to the second array. When the resulting correlation goes beyond (lower than) the target correlation, one steps back and reduces the disorder parameter of the permutation. When the resulting correlation lies between the current one and the target, one replaces the array with the newly permuted one. When the resulting correlation increases from the current value, one increases the disorder parameter. That way, the target correlation is approached from above, by a controlled increase in randomness. Since full randomness leads to zero correlation, the iterations meet the desired coefficient at some point. It may be noted that there could be some cases when one would get stuck in a sort of local minimum, where local perturbations cannot further reduce the correlation and where global ones lead to overpass the target. It seems easier to restart the program with a different seed when this occurs than to design an avoidance scheme. Also, should a negative correlation be desired, the program should be modified to start with one array in reverse order with respect to the other, i.e. correlation as close to -1 as possible.

Sorting

Full sorting

Subroutine INSSOR (XVALT) Sorts XVALT into increasing order (Insertion sort) This subroutine uses insertion sort. It does not use any work array and is faster when XVALT is of very small size (< 20), or already almost sorted, but worst case behavior (initially inverse sorted) can easily happen. In most cases, the quicksort or merge sort method is faster.

Subroutine REFSOR (XVALT) Sorts XVALT into increasing order (Quick sort) This version is not optimized for performance, and is thus not as difficult to read as some other ones. This subroutine uses quicksort in a recursive implementation, and insertion sort for the last steps with small subsets. It does not use any work array

Partial sorting

Subroutine INSPAR (XVALT, NORD) Sorts partially XVALT, bringing the NORD lowest values at the beginning of the array. This subroutine uses insertion sort, limiting insertion to the first NORD values. It does not use any work array and is faster when NORD is very small (2-5), but worst case behavior can happen fairly probably (initially inverse sorted). In many cases, the refined quicksort method is faster.

Function FNDNTH (XVALT, NORD) Finds out and returns the NORDth value in XVALT (ascending order) This subroutine uses insertion sort, limiting insertion to the first NORD values, and even less when one can know that the value that is considered will not be the NORDth. It uses only a work array of size NORD and is faster when NORD is very small (2-5), but worst case behavior can happen fairly probably (initially inverse sorted). In many cases, the refined quicksort method implemented by VALNTH / INDNTH is faster, though much more difficult to read and understand.

Function VALNTH (XVALT, NORD) Finds out and returns the NORDth value in XVALT (ascending order) This subroutine simply calls INDNTH.

Function VALMED (XVALT) Finds out and returns the median $((\text{Size}(\text{XVALT})+1)/2\text{th value})$ of XVALT This routine uses the recursive procedure described in Knuth, The Art of Computer Programming, vol. 3, 5.3.3 - This procedure is linear in time, and does not require to be able to interpolate in the set as the one used in VALNTH/INDNTH. It also has better worst case behavior than VALNTH/INDNTH, and is about 20% faster in average for random uniformly distributed values.

Function OMEDIAN (XVALT) It is a modified version of VALMED that provides the average between the two middle values in the case $\text{Size}(\text{XVALT})$ is even.

Unique sorting

Subroutine UNISTA (XVALT, NUNI) Removes duplicates from an array This subroutine uses merge sort unique inverse ranking. It leaves in the initial set only those entries that are unique, packing the array, and leaving the order of the retained values unchanged.

16.100.2 Member Function/Subroutine Documentation

16.100.2.1 d_refsor()

```
subroutine mo_orderpack::sort::d_refsor (
    real(kind=dp), dimension (:), intent(inout) XDONT )
```

16.100.2.2 i_refsor()

```
subroutine mo_orderpack::sort::i_refsor (
    integer(kind=i4), dimension (:), intent(inout) XDONT )
```

16.100.2.3 r_refsor()

```
subroutine mo_orderpack::sort::r_refsor (
    real(kind=sp), dimension (:), intent(inout) XDONT )
```

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

- [mo_orderpack.f90](#)

16.101 mo_orderpack::sort_index Interface Reference**Public Member Functions**

- integer(i4) function, dimension(size(arr)) [sort_index_dp](#) (arr)
- integer(i4) function, dimension(size(arr)) [sort_index_sp](#) (arr)
- integer(i4) function, dimension(size(arr)) [sort_index_i4](#) (arr)

16.101.1 Member Function/Subroutine Documentation**16.101.1.1 sort_index_dp()**

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index::sort_index_dp (
    real(dp), dimension(:), intent(in) arr )
```

16.101.1.2 sort_index_i4()

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index::sort_index_i4 (
    integer(i4), dimension(:), intent(in) arr )
```

16.101.1.3 sort_index_sp()

```
integer(i4) function, dimension(size(arr)) mo_orderpack::sort_index::sort_index_sp (
    real(sp), dimension(:), intent(in) arr )
```

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

- [mo_orderpack.f90](#)

16.102 mo_spatial_agg_disagg_forcing::spatial_aggregation Interface Reference

Spatial aggregation of meteorological variables.

Public Member Functions

- subroutine [spatial_aggregation_3d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [spatial_aggregation_4d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)

16.102.1 Detailed Description

Spatial aggregation of meteorological variables.

Aggregate (or upscale) the given level-2 meteorological data to the required level-1 spatial resolution for the mHM run.

Parameters

in	<i>real(dp), dimension(:, :, :) :: data2</i>	Level-2 meteorological data dim1=nRows, dim2=nCols, dim3=nTimeSteps
in	<i>real(dp) :: cellsize2</i>	Level-2 resolution
in	<i>real(dp) :: cellsize1</i>	Level-1 resolution
in	<i>logical, dimension(:, :) :: mask1</i>	Level-1 basin mask
in	<i>logical, dimension(:, :) :: mask2</i>	Level-2 basin mask dim1=nRows, dim2=nCols
out	<i>real(dp), allocatable, dimension(:, :, :) :: data1</i>	return level-1 meteorological data; DIMENSION [nRows_L1, nCols_L1, nTimeSteps]

Author

Rohini Kumar

Date

Jan 2013

16.102.2 Member Function/Subroutine Documentation

16.102.2.1 spatial_aggregation_3d()

```
subroutine mo_spatial_agg_disagg_forcing::spatial_aggregation::spatial_aggregation_3d (
    real(dp), dimension(:, :, :), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:, :), intent(in) mask1,
    logical, dimension(:, :), intent(in) mask2,
    real(dp), dimension(:, :, :), intent(out), allocatable data1 )
```

16.102.2.2 spatial_aggregation_4d()

```
subroutine mo_spatial_agg_disagg_forcing::spatial_aggregation::spatial_aggregation_4d (
    real(dp), dimension(:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:), intent(out), allocatable data1 )
```

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

- [mo_spatial_agg_disagg_forcing.f90](#)

16.103 mo_spatial_agg_disagg_forcing::spatial_disaggregation Interface Reference

Spatial disaggregation of meteorological variables.

Public Member Functions

- subroutine [spatial_disaggregation_3d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [spatial_disaggregation_4d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)

16.103.1 Detailed Description

Spatial disaggregation of meteorological variables.

Disaggregate (or downscale) the given level-2 meteorological data to the required level-1 spatial resolution for the mHM run.

Parameters

in	<i>real(dp), dimension(:,:,:) :: data2</i>	Level-2 meteorological data dim1=nRows, dim2=nCols, dim3=nTimeSteps
in	<i>real(dp) :: cellsize2</i>	Level-2 resolution
in	<i>real(dp) :: cellsize1</i>	Level-1 resolution
in	<i>logical, dimension(:,:) :: mask1</i>	Level-1 basin mask
in	<i>logical, dimension(:,:) :: mask2</i>	Level-2 basin mask dim1=nRows, dim2=nCols
out	<i>real(dp), allocatable, dimension(:,:,:) :: data1</i>	return level-1 meteorological data; DIMENSION [nRows_L1, nCols_L1, nTimeSteps]

Author

Rohini Kumar

Date

Jan 2013

16.103.2 Member Function/Subroutine Documentation

16.103.2.1 spatial_disaggregation_3d()

```

subroutine mo_spatial_agg_disagg_forcing::spatial_disaggregation::spatial_disaggregation_3d (
    real(dp), dimension(:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:), intent(out), allocatable data1 )

```

16.103.2.2 spatial_disaggregation_4d()

```

subroutine mo_spatial_agg_disagg_forcing::spatial_disaggregation::spatial_disaggregation_4d (
    real(dp), dimension(:,:,:,:), intent(in) data2,
    real(dp), intent(in) cellsize2,
    real(dp), intent(in) cellsize1,
    logical, dimension(:,:), intent(in) mask1,
    logical, dimension(:,:), intent(in) mask2,
    real(dp), dimension(:,:,:,:), intent(out), allocatable data1 )

```

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

- [mo_spatial_agg_disagg_forcing.f90](#)

16.104 mo_utils::special_value Interface Reference

Special IEEE values.

Public Member Functions

- real(sp) function [special_value_sp](#) (x, ieee)
- real(dp) function [special_value_dp](#) (x, ieee)

16.104.1 Detailed Description

Special IEEE values.

Returns special IEEE values such as Infinity or Not-a-Number.

Wraps to function `ieee_value` of the intrinsic module `ieee_arithmetic` but gives alternatives for gfortran, which does not provide `ieee_arithmetic`.

Quiet and signaling NaN are the same in case of gfortran;
also denormal values are the same as `inf`.

Current special values are:

```

IEEE_SIGNALING_NAN
IEEE_QUIET_NAN
IEEE_NEGATIVE_INF
IEEE_POSITIVE_INF
IEEE_NEGATIVE_DENORMAL
IEEE_POSITIVE_DENORMAL
IEEE_NEGATIVE_NORMAL
IEEE_POSITIVE_NORMAL
IEEE_NEGATIVE_ZERO
IEEE_POSITIVE_ZERO

```

Parameters

in	<i>real(sp/dp) :: x</i>	dummy for kind of output
in	<i>"character(le=*)</i>	:: name ieee signal nanme

Returns

`real(sp/dp) :: special_value` — IEEE special value
`IEEE_SIGNALING_NAN`
`IEEE_QUIET_NAN` (==`IEEE_SIGNALING_NAN` for gfortran)
`IEEE_NEGATIVE_INF`
`IEEE_POSITIVE_INF`
`IEEE_NEGATIVE_DENORMAL` (==`-0.0` for gfortran)
`IEEE_POSITIVE_DENORMAL` (==`0.0` for gfortran)
`IEEE_NEGATIVE_NORMAL` (==`-1.0` for gfortran)
`IEEE_POSITIVE_NORMAL` (==`1.0` for gfortran)
`IEEE_NEGATIVE_ZERO`
`IEEE_POSITIVE_ZERO`

Authors

Matthias Cuntz

Date

Mar 2015

16.104.2 Member Function/Subroutine Documentation

16.104.2.1 `special_value_dp()`

```

real(dp) function mo_utils::special_value::special_value_dp (
    real(dp), intent(in) x,
    character(len=*), intent(in) ieee )

```

16.104.2.2 `special_value_sp()`

```

real(sp) function mo_utils::special_value::special_value_sp (
    real(sp), intent(in) x,
    character(len=*), intent(in) ieee )

```

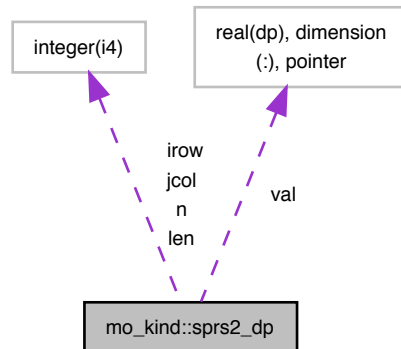
The documentation for this interface was generated from the following file:

- [mo_utils.f90](#)

16.105 `mo_kind::sprs2_dp` Type Reference

Double Precision Numerical Recipes types for sparse arrays.

Collaboration diagram for mo_kind::sprs2_dp:



Public Attributes

- integer([i4](#)) [n](#)
- integer([i4](#)) [len](#)
- real([dp](#)), dimension(:), pointer [val](#)
- integer([i4](#)), dimension(:), pointer [irow](#)
- integer([i4](#)), dimension(:), pointer [jcol](#)

16.105.1 Detailed Description

Double Precision Numerical Recipes types for sparse arrays.

16.105.2 Member Data Documentation

16.105.2.1 irow

```
integer(i4), dimension(:), pointer mo_kind::sprs2_dp::irow
```

16.105.2.2 jcol

```
integer(i4), dimension(:), pointer mo_kind::sprs2_dp::jcol
```

16.105.2.3 len

```
integer(i4) mo_kind::sprs2_dp::len
```

16.105.2.4 `n`

```
integer(i4) mo_kind::sprs2_dp::n
```

16.105.2.5 `val`

```
real(dp), dimension(:), pointer mo_kind::sprs2_dp::val
```

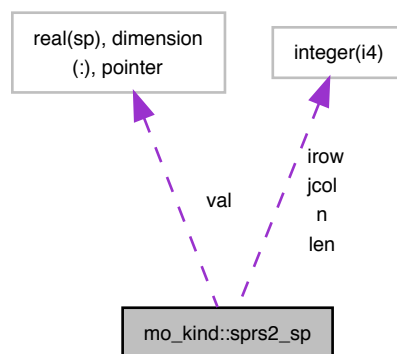
The documentation for this type was generated from the following file:

- [mo_kind.f90](#)

16.106 `mo_kind::sprs2_sp` Type Reference

Single Precision Numerical Recipes types for sparse arrays.

Collaboration diagram for `mo_kind::sprs2_sp`:



Public Attributes

- integer(i4) `n`
- integer(i4) `len`
- real(sp), dimension(:), pointer `val`
- integer(i4), dimension(:), pointer `irow`
- integer(i4), dimension(:), pointer `jcol`

16.106.1 Detailed Description

Single Precision Numerical Recipes types for sparse arrays.

16.106.2 Member Data Documentation

16.106.2.1 irow

```
integer(i4), dimension(:), pointer mo_kind::sprs2_sp::irow
```

16.106.2.2 jcol

```
integer(i4), dimension(:), pointer mo_kind::sprs2_sp::jcol
```

16.106.2.3 len

```
integer(i4) mo_kind::sprs2_sp::len
```

16.106.2.4 n

```
integer(i4) mo_kind::sprs2_sp::n
```

16.106.2.5 val

```
real(sp), dimension(:), pointer mo_kind::sprs2_sp::val
```

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

- [mo_kind.f90](#)

16.107 mo_errormeasures::sse Interface Reference**Public Member Functions**

- real(sp) function [sse_sp_1d](#) (x, y, mask)
- real(dp) function [sse_dp_1d](#) (x, y, mask)
- real(sp) function [sse_sp_2d](#) (x, y, mask)
- real(dp) function [sse_dp_2d](#) (x, y, mask)
- real(sp) function [sse_sp_3d](#) (x, y, mask)
- real(dp) function [sse_dp_3d](#) (x, y, mask)

16.107.1 Member Function/Subroutine Documentation**16.107.1.1 sse_dp_1d()**

```
real(dp) function mo_errormeasures::sse::sse_dp_1d (
    real(dp), dimension(:), intent(in) x,
    real(dp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.107.1.2 sse_dp_2d()

```
real(dp) function mo_errormeasures::sse::sse_dp_2d (
    real(dp), dimension(:,:), intent(in) x,
    real(dp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.107.1.3 sse_dp_3d()

```
real(dp) function mo_errormeasures::sse::sse_dp_3d (
    real(dp), dimension(:,:,:), intent(in) x,
    real(dp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

16.107.1.4 sse_sp_1d()

```
real(sp) function mo_errormeasures::sse::sse_sp_1d (
    real(sp), dimension(:), intent(in) x,
    real(sp), dimension(:), intent(in) y,
    logical, dimension(:), intent(in), optional mask )
```

16.107.1.5 sse_sp_2d()

```
real(sp) function mo_errormeasures::sse::sse_sp_2d (
    real(sp), dimension(:,:), intent(in) x,
    real(sp), dimension(:,:), intent(in) y,
    logical, dimension(:,:), intent(in), optional mask )
```

16.107.1.6 sse_sp_3d()

```
real(sp) function mo_errormeasures::sse::sse_sp_3d (
    real(sp), dimension(:,:,:), intent(in) x,
    real(sp), dimension(:,:,:), intent(in) y,
    logical, dimension(:,:,:), intent(in), optional mask )
```

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

- [mo_errormeasures.f90](#)

16.108 mo_standard_score::standard_score Interface Reference

Calculates the standard score / normalization (anomaly) / z-score.

Public Member Functions

- `real(sp) function, dimension(size(data, dim=1))` [standard_score_sp](#) (data, mask)
- `real(dp) function, dimension(size(data, dim=1))` [standard_score_dp](#) (data, mask)

16.108.1 Detailed Description

Calculates the standard score / normalization (anomaly) / z-score.

In statistics, the standard score is the (signed) number of standard deviations an observation or datum is above the mean. Thus, a positive standard score indicates a datum above the mean, while a negative standard score indicates a datum below the mean. It is a dimensionless quantity obtained by subtracting the population mean from an individual raw score and then dividing the difference by the population standard deviation. This conversion process is called standardizing or normalizing (however, "normalizing" can refer to many types of ratios).

Standard scores are also called z-values, z-scores, normal scores, and standardized variables; the use of "Z" is because the normal distribution is also known as the "Z distribution". They are most frequently used to compare a sample to a standard normal deviate, though they can be defined without assumptions of normality (Wikipedia, May 2015).

$$standard_score = \frac{x - \mu_x}{\sigma_x}$$

where μ_x is the mean of a population x and σ_x its standard deviation.

If an optimal mask is given, the calculations are over those locations that correspond to true values in the mask. data can be single or double precision. The result will have the same numerical precision.

Parameters

in	<i>real(sp/dp), dimension(:) :: data</i>	data to calculate the standard score for
in	<i>logical, dimension(:), optional :: mask</i>	indication which cells to use for calculation If present, only those locations in mask having true values in mask are evaluated.

Returns

real(sp/dp) :: [standard_score](#) — standard score / normalization (anomaly) / z-score

Note

Richard J. Larsen and Morris L. Marx (2000) An Introduction to Mathematical Statistics and Its Applications, Third Edition, ISBN 0-13-922303-7. p. 282.

Author

Matthias Zink

Date

May 2015

16.108.2 Member Function/Subroutine Documentation

16.108.2.1 `standard_score_dp()`

```
real(dp) function, dimension(size(data, dim=1)) mo_standard_score::standard_score::standard_↵
score_dp (
    real(dp), dimension(:), intent(in) data,
    logical, dimension(:), intent(in), optional mask )
```

16.108.2.2 `standard_score_sp()`

```
real(sp) function, dimension(size(data, dim=1)) mo_standard_score::standard_score↔
score_sp (
    real(sp), dimension(:), intent(in) data,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_standard_score.f90](#)

16.109 `mo_moment::stddev` Interface Reference

Public Member Functions

- real(sp) function [stddev_sp](#) (dat, mask)
- real(dp) function [stddev_dp](#) (dat, mask)

16.109.1 Member Function/Subroutine Documentation

16.109.1.1 `stddev_dp()`

```
real(dp) function mo_moment::stddev::stddev_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

16.109.1.2 `stddev_sp()`

```
real(sp) function mo_moment::stddev::stddev_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.110 `mo_corr::swap` Interface Reference

Private Member Functions

- subroutine [swap_1d_spc](#) (a, b)
- subroutine [swap_1d_dpc](#) (a, b)

16.110.1 Member Function/Subroutine Documentation

16.110.1.1 swap_1d_dpc()

```
subroutine mo_corr::swap::swap_1d_dpc (
    complex(dpc), dimension(:), intent(inout) a,
    complex(dpc), dimension(:), intent(inout) b ) [private]
```

16.110.1.2 swap_1d_spc()

```
subroutine mo_corr::swap::swap_1d_spc (
    complex(spc), dimension(:), intent(inout) a,
    complex(spc), dimension(:), intent(inout) b ) [private]
```

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

- [mo_corr.f90](#)

16.111 mo_utils::swap Interface Reference

Swap to values or two elements in array.

Public Member Functions

- elemental pure subroutine [swap_xy_dp](#) (x, y)
- elemental pure subroutine [swap_xy_sp](#) (x, y)
- elemental pure subroutine [swap_xy_i4](#) (x, y)
- subroutine [swap_vec_dp](#) (x, i1, i2)
- subroutine [swap_vec_sp](#) (x, i1, i2)
- subroutine [swap_vec_i4](#) (x, i1, i2)

16.111.1 Detailed Description

Swap to values or two elements in array.

Swaps either two entities, i.e. scalars, vectors, matrices, or two elements in a vector. The call is either call swap(x,y)

or

call swap(vec,i,j)

Parameters

in	<i>integer(i4) :: i</i>	Index of first element to be swapped with second [case swap(vec,i,j)]
in	<i>integer(i4) :: j</i>	Index of second element to be swapped with first [case swap(vec,i,j)]
in, out	<i>real(sp/dp/i4) :: x[:,...]</i>	First scalar or array to swap with second [case swap(x,y)]
in, out	<i>real(sp/dp/i4) :: y[:,...]</i>	Second scalar or array to swap with first [case swap(x,y)]
in, out	<i>real(sp/dp/i4) :: x(:)</i>	Vector of which to elements are swapped [case swap(vec,i,j)]

Author

Matthias Cuntz

Date

May 2014

16.111.2 Member Function/Subroutine Documentation

16.111.2.1 swap_vec_dp()

```
subroutine mo_utils::swap::swap_vec_dp (  
    real(dp), dimension(:), intent(inout) x,  
    integer(i4), intent(in) i1,  
    integer(i4), intent(in) i2 )
```

16.111.2.2 swap_vec_i4()

```
subroutine mo_utils::swap::swap_vec_i4 (  
    integer(i4), dimension(:), intent(inout) x,  
    integer(i4), intent(in) i1,  
    integer(i4), intent(in) i2 )
```

16.111.2.3 swap_vec_sp()

```
subroutine mo_utils::swap::swap_vec_sp (  
    real(sp), dimension(:), intent(inout) x,  
    integer(i4), intent(in) i1,  
    integer(i4), intent(in) i2 )
```

16.111.2.4 swap_xy_dp()

```
elemental pure subroutine mo_utils::swap::swap_xy_dp (  
    real(dp), intent(inout) x,  
    real(dp), intent(inout) y )
```

16.111.2.5 swap_xy_i4()

```
elemental pure subroutine mo_utils::swap::swap_xy_i4 (  
    integer(i4), intent(inout) x,  
    integer(i4), intent(inout) y )
```

16.111.2.6 swap_xy_sp()

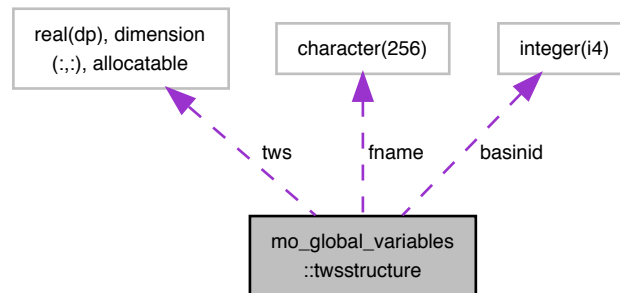
```
elemental pure subroutine mo_utils::swap::swap_xy_sp (  
    real(sp), intent(inout) x,  
    real(sp), intent(inout) y )
```

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

- [mo_utils.f90](#)

16.112 mo_global_variables::twstructure Type Reference

Collaboration diagram for mo_global_variables::twstructure:



Public Attributes

- integer(i4), dimension(:), allocatable [basinid](#)
- character(256), dimension(:), allocatable [fname](#)
- real(dp), dimension(:, :), allocatable [twstructure](#)

16.112.1 Member Data Documentation

16.112.1.1 basinid

integer(i4), dimension(:), allocatable mo_global_variables::twstructure::basinid

16.112.1.2 fname

character(256), dimension(:), allocatable mo_global_variables::twstructure::fname

16.112.1.3 twstructure

real(dp), dimension(:, :), allocatable mo_global_variables::twstructure::twstructure

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

- [mo_global_variables.f90](#)

16.113 mo_orderpack::uniinv Interface Reference

Public Member Functions

- subroutine [d_uniinv](#) (XDONT, IGOEST)
- subroutine [r_uniinv](#) (XDONT, IGOEST)
- subroutine [i_uniinv](#) (XDONT, IGOEST)

16.113.1 Member Function/Subroutine Documentation

16.113.1.1 d_uniinv()

```
subroutine mo_orderpack::uniinv::d_uniinv (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST )
```

16.113.1.2 i_uniinv()

```
subroutine mo_orderpack::uniinv::i_uniinv (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST )
```

16.113.1.3 r_uniinv()

```
subroutine mo_orderpack::uniinv::r_uniinv (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IGOEST )
```

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

- [mo_orderpack.f90](#)

16.114 mo_orderpack::unipar Interface Reference

Public Member Functions

- subroutine [d_unipar](#) (XDONT, IRNGT, NORD)
- subroutine [r_unipar](#) (XDONT, IRNGT, NORD)
- subroutine [i_unipar](#) (XDONT, IRNGT, NORD)

16.114.1 Member Function/Subroutine Documentation

16.114.1.1 d_unipar()

```
subroutine mo_orderpack::unipar::d_unipar (
    real(kind=dp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD )
```

16.114.1.2 i_unipar()

```
subroutine mo_orderpack::unipar::i_unipar (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD )
```

16.114.1.3 r_unipar()

```
subroutine mo_orderpack::unipar::r_unipar (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(inout) NORD )
```

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

- [mo_orderpack.f90](#)

16.115 mo_orderpack::unirnk Interface Reference

Public Member Functions

- subroutine [d_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine [r_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine [i_unirnk](#) (XVALT, IRNGT, NUNI)

16.115.1 Member Function/Subroutine Documentation

16.115.1.1 d_unirnk()

```
subroutine mo_orderpack::unirnk::d_unirnk (
    real(kind=dp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(out) NUNI )
```

16.115.1.2 i_unirnk()

```
subroutine mo_orderpack::unirnk::i_unirnk (
    integer(kind=i4), dimension (:), intent(in) XVALT,
```

```
integer(kind=i4), dimension (:), intent(out) IRNGT,
integer(kind=i4), intent(out) NUNI )
```

16.115.1.3 r_unirnk()

```
subroutine mo_orderpack::unirnk::r_unirnk (
    real(kind=sp), dimension (:), intent(in) XVALT,
    integer(kind=i4), dimension (:), intent(out) IRNGT,
    integer(kind=i4), intent(out) NUNI )
```

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

- [mo_orderpack.f90](#)

16.116 mo_orderpack::unista Interface Reference

Public Member Functions

- subroutine [d_unista](#) (XDONT, NUNI)
- subroutine [r_unista](#) (XDONT, NUNI)
- subroutine [i_unista](#) (XDONT, NUNI)

16.116.1 Member Function/Subroutine Documentation

16.116.1.1 d_unista()

```
subroutine mo_orderpack::unista::d_unista (
    real(kind=dp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI )
```

16.116.1.2 i_unista()

```
subroutine mo_orderpack::unista::i_unista (
    integer(kind=i4), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI )
```

16.116.1.3 r_unista()

```
subroutine mo_orderpack::unista::r_unista (
    real(kind=sp), dimension (:), intent(inout) XDONT,
    integer(kind=i4), intent(out) NUNI )
```

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

- [mo_orderpack.f90](#)

16.117 mo_orderpack::valmed Interface Reference

Public Member Functions

- recursive real(kind=dp) function [d_valmed](#) (XDONT)
- recursive real(kind=sp) function [r_valmed](#) (XDONT)
- recursive integer(kind=i4) function [i_valmed](#) (XDONT)

16.117.1 Member Function/Subroutine Documentation

16.117.1.1 d_valmed()

```
recursive real(kind=dp) function mo_orderpack::valmed::d_valmed (  
    real(kind=dp), dimension (:), intent(in) XDONT )
```

16.117.1.2 i_valmed()

```
recursive integer(kind=i4) function mo_orderpack::valmed::i_valmed (  
    integer(kind=i4), dimension (:), intent(in) XDONT )
```

16.117.1.3 r_valmed()

```
recursive real(kind=sp) function mo_orderpack::valmed::r_valmed (  
    real(kind=sp), dimension (:), intent(in) XDONT )
```

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

- [mo_orderpack.f90](#)

16.118 mo_orderpack::valnth Interface Reference

Public Member Functions

- real(kind=dp) function [d_valnth](#) (XDONT, NORD)
- real(kind=sp) function [r_valnth](#) (XDONT, NORD)
- integer(kind=i4) function [i_valnth](#) (XDONT, NORD)

16.118.1 Member Function/Subroutine Documentation

16.118.1.1 d_valnth()

```
real(kind=dp) function mo_orderpack::valnth::d_valnth (  
    real(kind=dp), dimension (:), intent(in) XDONT,  
    integer(kind=i4), intent(in) NORD )
```

16.118.1.2 i_valnth()

```
integer(kind=i4) function mo_orderpack::valnth::i_valnth (
    integer(kind=i4), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

16.118.1.3 r_valnth()

```
real(kind=sp) function mo_orderpack::valnth::r_valnth (
    real(kind=sp), dimension (:), intent(in) XDONT,
    integer(kind=i4), intent(in) NORD )
```

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

- [mo_orderpack.f90](#)

16.119 mo_ncwrite::var2nc Interface Reference

Extended [dump_netcdf](#) for multiple variables.

Public Member Functions

- subroutine [var2nc_1d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_1d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_1d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_2d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_2d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_2d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_3d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_4d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [var2nc_5d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)

16.119.1 Detailed Description

Extended [dump_netcdf](#) for multiple variables.

Write different variables including attributes to netcdf file. The attributes are restricted to long_name, units, and missing_value. It is also possible to append variables when an unlimited dimension is specified. call [var2nc](#)(f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, create)

Parameters

in	<i>character(*) :: f_name</i>	filename
in	<i>integer(i4)/real(sp,dp) :: arr(:,:[:[:[:[:[:]]]])</i>	array to write
in	<i>character(*) :: dnames(:)</i>	dimension names
in	<i>character(*) :: v_name</i>	variable name
in	<i>integer(i4), optional :: dim_unlimited</i>	index of unlimited dimension
in	<i>character(*), optional :: long_name</i>	attribute
in	<i>character(*), optional :: units</i>	attribute
in	<i>integer(i4)/real(sp,dp), optional :: missing_value</i>	attribute
in	<i>character(256), dimension(:,:), optional :: attributes</i>	two dimensional array of attributes size of first dimension equals number of attributes first entry of second dimension equals attribute name (e.g. long_name) second entry of second dimension equals attribute value (e.g. precipitation) every attribute is written as string with the exception of missing_value
in	<i>logical, optional :: create</i>	flag - specify whether a output file should be created, default
in, out	<i>integer(i4)/real(sp,dp), optional :: ncid</i>	if not given filename will be opened and closed if given and <0 then file will be opened and ncid will return the file unit. if given and >0 then file is assumed open and ncid is used as file unit.
in	<i>integer(i4), optional :: nrec</i>	if given: start point on unlimited dimension.

Note

It is not allowed to write the folloing numbers for the indicated type
number | kind

-2.1474836E+09 | integer(i4)

9.9692100E+36 | real(sp)

9.9692099683868690E+36 | real(dp)

These numbers are netcdf fortran 90 constants! They are used to determine the chunksize of the already written variable. Hence, this routine cannot append correctly to variables when these numbers are used. Only five dimensional variables can be written, only one unlimited dimension can be defined.

Author

Stephan Thober & Matthias Cuntz

Date

May 2014

16.119.2 Member Function/Subroutine Documentation

16.119.2.1 var2nc_1d_dp()

```

subroutine mo_ncwrite::var2nc::var2nc_1d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(dp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.2 var2nc_1d_i4()

```

subroutine mo_ncwrite::var2nc::var2nc_1d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.3 var2nc_1d_sp()

```

subroutine mo_ncwrite::var2nc::var2nc_1d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.4 var2nc_2d_dp()

```

subroutine mo_ncwrite::var2nc::var2nc_2d_dp (
    character(len=*), intent(in) f_name,

```

```

real(dp), dimension(:,:), intent(in) arr,
character(len=*), dimension(:), intent(in) dnames,
character(len=*), intent(in) v_name,
integer(i4), intent(in), optional dim_unlimited,
character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec )

```

16.119.2.5 var2nc_2d_i4()

```

subroutine mo_ncwrite::var2nc::var2nc_2d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.6 var2nc_2d_sp()

```

subroutine mo_ncwrite::var2nc::var2nc_2d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.7 var2nc_3d_dp()

```

subroutine mo_ncwrite::var2nc::var2nc_3d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,

```

```

character(len=*), intent(in), optional long_name,
character(len=*), intent(in), optional units,
real(dp), intent(in), optional missing_value,
character(256), dimension(:,:), intent(in), optional attributes,
logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec )

```

16.119.2.8 var2nc_3d_i4()

```

subroutine mo_ncwrite::var2nc::var2nc_3d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.9 var2nc_3d_sp()

```

subroutine mo_ncwrite::var2nc::var2nc_3d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.10 var2nc_4d_dp()

```

subroutine mo_ncwrite::var2nc::var2nc_4d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:,:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(dp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,

```

```

logical, intent(in), optional create,
integer(i4), intent(inout), optional ncid,
integer(i4), intent(in), optional nrec )

```

16.119.2.11 var2nc_4d_i4()

```

subroutine mo_ncwrite::var2nc::var2nc_4d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.12 var2nc_4d_sp()

```

subroutine mo_ncwrite::var2nc::var2nc_4d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.13 var2nc_5d_dp()

```

subroutine mo_ncwrite::var2nc::var2nc_5d_dp (
    character(len=*), intent(in) f_name,
    real(dp), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(dp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.14 var2nc_5d_i4()

```

subroutine mo_ncwrite::var2nc::var2nc_5d_i4 (
    character(len=*), intent(in) f_name,
    integer(i4), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    integer(i4), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

```

16.119.2.15 var2nc_5d_sp()

```

subroutine mo_ncwrite::var2nc::var2nc_5d_sp (
    character(len=*), intent(in) f_name,
    real(sp), dimension(:,:,:,,:), intent(in) arr,
    character(len=*), dimension(:), intent(in) dnames,
    character(len=*), intent(in) v_name,
    integer(i4), intent(in), optional dim_unlimited,
    character(len=*), intent(in), optional long_name,
    character(len=*), intent(in), optional units,
    real(sp), intent(in), optional missing_value,
    character(256), dimension(:,:), intent(in), optional attributes,
    logical, intent(in), optional create,
    integer(i4), intent(inout), optional ncid,
    integer(i4), intent(in), optional nrec )

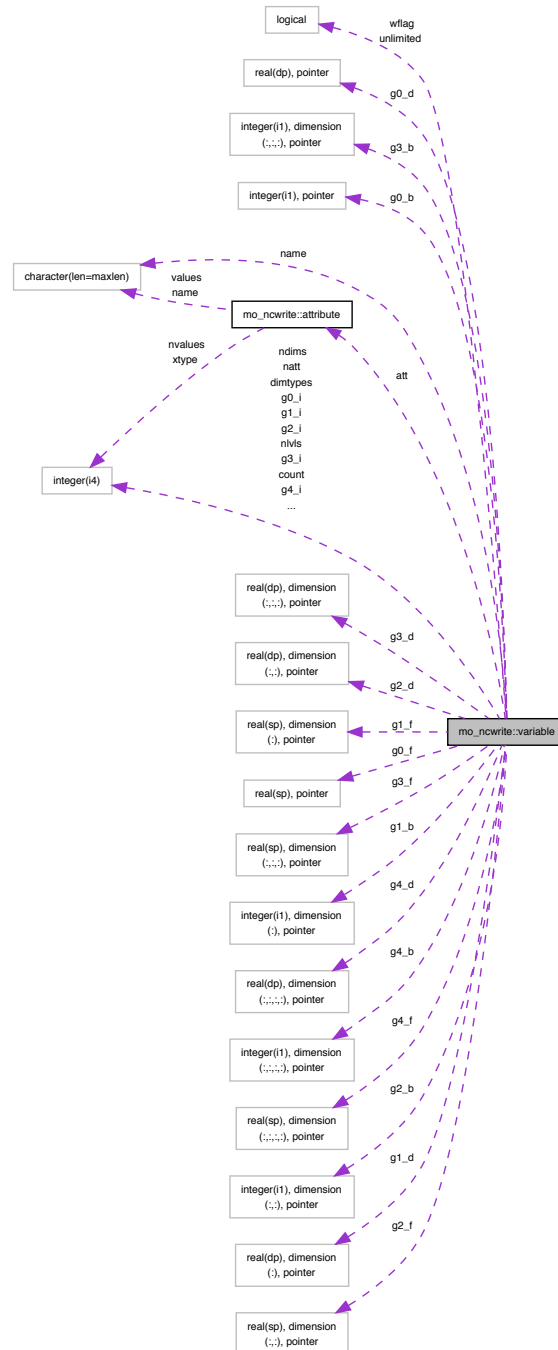
```

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

- [mo_ncwrite.f90](#)

16.120 mo_ncwrite::variable Type Reference

Collaboration diagram for mo_ncwrite::variable:



Public Attributes

- `character(len=maxlen)` `name`
- `integer(i4)` `xtype`
- `integer(i4)` `nlvs`

- integer(i4) [nsubs](#)
- logical [unlimited](#)
- integer(i4) [varid](#)
- integer(i4) [ndims](#)
- integer(i4), dimension(nmaxdim) [dimids](#)
- integer(i4), dimension(nmaxdim) [dimtypes](#)
- integer(i4) [natt](#)
- type([attribute](#)), dimension(nmaxatt) [att](#)
- integer(i4), dimension(nmaxdim) [start](#)
- integer(i4), dimension(nmaxdim) [count](#)
- logical [wflag](#)
- integer(i1), pointer [g0_b](#)
- integer(i1), dimension(:), pointer [g1_b](#)
- integer(i1), dimension(:, :), pointer [g2_b](#)
- integer(i1), dimension(:, :, :), pointer [g3_b](#)
- integer(i1), dimension(:, :, :, :), pointer [g4_b](#)
- integer(i4), pointer [g0_i](#)
- integer(i4), dimension(:), pointer [g1_i](#)
- integer(i4), dimension(:, :), pointer [g2_i](#)
- integer(i4), dimension(:, :, :), pointer [g3_i](#)
- integer(i4), dimension(:, :, :, :), pointer [g4_i](#)
- real(sp), pointer [g0_f](#)
- real(sp), dimension(:), pointer [g1_f](#)
- real(sp), dimension(:, :), pointer [g2_f](#)
- real(sp), dimension(:, :, :), pointer [g3_f](#)
- real(sp), dimension(:, :, :, :), pointer [g4_f](#)
- real(dp), pointer [g0_d](#)
- real(dp), dimension(:), pointer [g1_d](#)
- real(dp), dimension(:, :), pointer [g2_d](#)
- real(dp), dimension(:, :, :), pointer [g3_d](#)
- real(dp), dimension(:, :, :, :), pointer [g4_d](#)

16.120.1 Member Data Documentation

16.120.1.1 att

type([attribute](#)), dimension(nmaxatt) mo_ncwrite::variable::att

16.120.1.2 count

integer(i4), dimension(nmaxdim) mo_ncwrite::variable::count

16.120.1.3 dimids

integer(i4), dimension(nmaxdim) mo_ncwrite::variable::dimids

16.120.1.4 dimtypes

integer(i4), dimension(nmaxdim) mo_ncwrite::variable::dimtypes

16.120.1.5 g0_b

integer(i1), pointer mo_ncwrite::variable::g0_b

16.120.1.6 g0_d

real(dp), pointer mo_ncwrite::variable::g0_d

16.120.1.7 g0_f

real(sp), pointer mo_ncwrite::variable::g0_f

16.120.1.8 g0_i

integer(i4), pointer mo_ncwrite::variable::g0_i

16.120.1.9 g1_b

integer(i1), dimension(:), pointer mo_ncwrite::variable::g1_b

16.120.1.10 g1_d

real(dp), dimension(:), pointer mo_ncwrite::variable::g1_d

16.120.1.11 g1_f

real(sp), dimension(:), pointer mo_ncwrite::variable::g1_f

16.120.1.12 g1_i

integer(i4), dimension(:), pointer mo_ncwrite::variable::g1_i

16.120.1.13 g2_b

```
integer(i1), dimension(:,:) , pointer mo_ncwrite::variable::g2_b
```

16.120.1.14 g2_d

```
real(dp), dimension(:,:) , pointer mo_ncwrite::variable::g2_d
```

16.120.1.15 g2_f

```
real(sp), dimension(:,:) , pointer mo_ncwrite::variable::g2_f
```

16.120.1.16 g2_i

```
integer(i4), dimension(:,:) , pointer mo_ncwrite::variable::g2_i
```

16.120.1.17 g3_b

```
integer(i1), dimension(:,:,:) , pointer mo_ncwrite::variable::g3_b
```

16.120.1.18 g3_d

```
real(dp), dimension(:,:,:) , pointer mo_ncwrite::variable::g3_d
```

16.120.1.19 g3_f

```
real(sp), dimension(:,:,:) , pointer mo_ncwrite::variable::g3_f
```

16.120.1.20 g3_i

```
integer(i4), dimension(:,:,:) , pointer mo_ncwrite::variable::g3_i
```

16.120.1.21 g4_b

```
integer(i1), dimension(:,:,:), pointer mo_ncwrite::variable::g4_b
```

16.120.1.22 g4_d

```
real(dp), dimension(:,:,:,:), pointer mo_ncwrite::variable::g4_d
```

16.120.1.23 g4_f

```
real(sp), dimension(:,:,:,:), pointer mo_ncwrite::variable::g4_f
```

16.120.1.24 g4_i

```
integer(i4), dimension(:,:,:,:), pointer mo_ncwrite::variable::g4_i
```

16.120.1.25 name

```
character (len=maxlen) mo_ncwrite::variable::name
```

16.120.1.26 natt

```
integer(i4) mo_ncwrite::variable::natt
```

16.120.1.27 ndims

```
integer(i4) mo_ncwrite::variable::ndims
```

16.120.1.28 nlvls

```
integer(i4) mo_ncwrite::variable::nlvls
```

16.120.1.29 nsubs

```
integer(i4) mo_ncwrite::variable::nsubs
```

16.120.1.30 start

```
integer(i4), dimension(nmaxdim) mo_ncwrite::variable::start
```

16.120.1.31 unlimited

```
logical mo_ncwrite::variable::unlimited
```

16.120.1.32 varid

```
integer(i4) mo_ncwrite::variable::varid
```

16.120.1.33 wflag

```
logical mo_ncwrite::variable::wflag
```

16.120.1.34 xtype

```
integer(i4) mo_ncwrite::variable::xtype
```

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

- [mo_ncwrite.f90](#)

16.121 mo_moment::variance Interface Reference**Public Member Functions**

- real(sp) function [variance_sp](#) (dat, mask)
- real(dp) function [variance_dp](#) (dat, mask)

16.121.1 Member Function/Subroutine Documentation**16.121.1.1 variance_dp()**

```
real(dp) function mo_moment::variance::variance_dp (
    real(dp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

16.121.1.2 variance_sp()

```
real(sp) function mo_moment::variance::variance_sp (
    real(sp), dimension(:), intent(in) dat,
    logical, dimension(:), intent(in), optional mask )
```

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

- [mo_moment.f90](#)

16.122 mo_xor4096::xor4096 Interface Reference

Public Member Functions

- subroutine [xor4096s_0d](#) (seed, SingleIntegerRN, save_state)
- subroutine [xor4096s_1d](#) (seed, SingleIntegerRN, save_state)
- subroutine [xor4096f_0d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096f_1d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096l_0d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [xor4096l_1d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [xor4096d_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [xor4096d_1d](#) (seed, DoubleRealRN, save_state)

16.122.1 Member Function/Subroutine Documentation

16.122.1.1 xor4096d_0d()

```
subroutine mo_xor4096::xor4096::xor4096d_0d (
    integer(i8), intent(in) seed,
    real(dp), intent(out) DoubleRealRN,
    integer(i8), dimension(n_save_state), intent(inout), optional save_state )
```

16.122.1.2 xor4096d_1d()

```
subroutine mo_xor4096::xor4096::xor4096d_1d (
    integer(i8), dimension(:), intent(in) seed,
    real(dp), dimension(size(seed,1)), intent(out) DoubleRealRN,
    integer(i8), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state )
```

16.122.1.3 xor4096f_0d()

```
subroutine mo_xor4096::xor4096::xor4096f_0d (
    integer(i4), intent(in) seed,
    real(sp), intent(out) SingleRealRN,
    integer(i4), dimension(n_save_state), intent(inout), optional save_state )
```

16.122.1.4 xor4096f_1d()

```
subroutine mo_xor4096::xor4096::xor4096f_1d (
    integer(i4), dimension(:), intent(in) seed,
    real(sp), dimension(size(seed)), intent(out) SingleRealRN,
    integer(i4), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state )
```

16.122.1.5 xor4096l_0d()

```
subroutine mo_xor4096::xor4096::xor4096l_0d (
    integer(i8), intent(in) seed,
    integer(i8), intent(out) DoubleIntegerRN,
    integer(i8), dimension(n_save_state), intent(inout), optional save_state )
```

16.122.1.6 xor4096l_1d()

```
subroutine mo_xor4096::xor4096::xor4096l_1d (
    integer(i8), dimension(:), intent(in) seed,
    integer(i8), dimension(size(seed,1)), intent(out) DoubleIntegerRN,
    integer(i8), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state )
```

16.122.1.7 xor4096s_0d()

```
subroutine mo_xor4096::xor4096::xor4096s_0d (
    integer(i4), intent(in) seed,
    integer(i4), intent(out) SingleIntegerRN,
    integer(i4), dimension(n_save_state), intent(inout), optional save_state )
```

16.122.1.8 xor4096s_1d()

```
subroutine mo_xor4096::xor4096::xor4096s_1d (
    integer(i4), dimension(:), intent(in) seed,
    integer(i4), dimension(size(seed,1)), intent(out) SingleIntegerRN,
    integer(i4), dimension(size(seed,1),n_save_state), intent(inout), optional save_↵
state )
```

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

- [mo_xor4096.f90](#)

16.123 mo_xor4096::xor4096g Interface Reference**Public Member Functions**

- subroutine [xor4096gf_0d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096gf_1d](#) (seed, SingleRealRN, save_state)
- subroutine [xor4096gd_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [xor4096gd_1d](#) (seed, DoubleRealRN, save_state)

16.123.1 Member Function/Subroutine Documentation

16.123.1.1 xor4096gd_0d()

```
subroutine mo_xor4096::xor4096g::xor4096gd_0d (  
    integer(i8), intent(in) seed,  
    real(dp), intent(out) DoubleRealRN,  
    integer(i8), dimension(n_save_state), intent(inout), optional save_state )
```

16.123.1.2 xor4096gd_1d()

```
subroutine mo_xor4096::xor4096g::xor4096gd_1d (  
    integer(i8), dimension(:), intent(in) seed,  
    real(dp), dimension(size(seed)), intent(out) DoubleRealRN,  
    integer(i8), dimension(size(seed),n_save_state), intent(inout), optional save_←  
state )
```

16.123.1.3 xor4096gf_0d()

```
subroutine mo_xor4096::xor4096g::xor4096gf_0d (  
    integer(i4), intent(in) seed,  
    real(sp), intent(out) SingleRealRN,  
    integer(i4), dimension(n_save_state), intent(inout), optional save_state )
```

16.123.1.4 xor4096gf_1d()

```
subroutine mo_xor4096::xor4096g::xor4096gf_1d (  
    integer(i4), dimension(:), intent(in) seed,  
    real(sp), dimension(size(seed)), intent(out) SingleRealRN,  
    integer(i4), dimension(size(seed),n_save_state), intent(inout), optional save_←  
state )
```

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

- [mo_xor4096.f90](#)

Chapter 17

File Documentation

17.1 1-main.dox File Reference

17.2 2-get_started.dox File Reference

17.3 3-data_preparation.dox File Reference

17.4 4-visualise_out.dox File Reference

17.5 5-calibration.dox File Reference

17.6 6-style_guide.dox File Reference

17.7 7-test_basin.dox File Reference

17.8 8-protocols_for_setup_new_mHM_basin.dox File Reference

17.9 DEPENDENCIES.md File Reference

17.10 mhm_driver.f90 File Reference

Functions/Subroutines

- program [mhm_driver](#)

Distributed precipitation-runoff model mHM.

17.10.1 Function/Subroutine Documentation

software should be clearly marked, so as not to confuse it with the version available from UFZ. This code can be used for research purposes ONLY provided that the following sources are acknowledged:

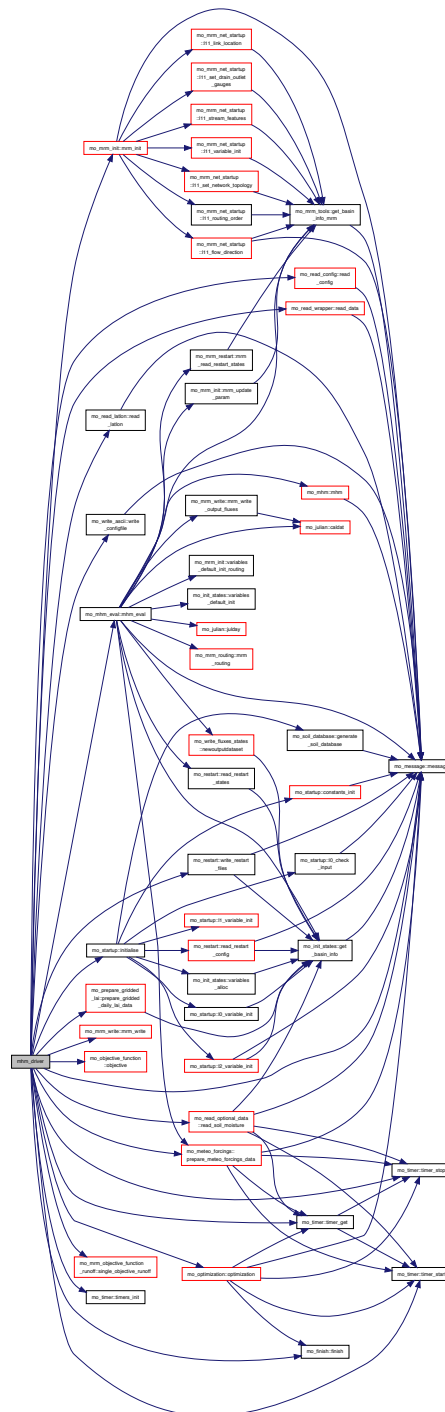
Samaniego L., Kumar R., Attinger S. (2010): Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale. *Water Resour. Res.*, 46, W05523, doi:10.1029/2008WR007327.

Kumar, R., L. Samaniego, and S. Attinger (2013), Implications of distributed hydrologic model parameterization on water fluxes at multiple scales and locations, *Water Resour. Res.*, 49, doi:10.1029/2012WR012195.

For commercial applications you have to consult the authorities of the UFZ.

References mo_kind::dp, mo_file::file_main, mo_finish::finish(), mo_kind::i4, mo_startup::initialise(), mo_message::message(), mo_message::message_text, mo_mhm_eval::mhm_eval(), mo_mrm_init::mrm_init(), mo_mrm_write::mrm_write(), mo_global_variables::nbasins, mo_objective_function::objective(), mo_optimization::optimization(), mo_prepare_gridded_lai::prepare_gridded_daily_lai_data(), mo_meteo_forcings::prepare_meteo_forcings_data(), mo_common_variables::processmatrix, mo_read_config::read_config(), mo_read_wrapper::read_data(), mo_read_latlon::read_latlon(), mo_read_optional_data::read_soil_moisture(), mo_string_utils::separator, mo_mrm_objective_function_runoff::single_objective_runoff(), mo_timer::timer_get(), mo_timer::timer_start(), mo_timer::timer_stop(), mo_timer::timers_init(), mo_global_variables::timestep_model_inputs, mo_file::version, mo_file::version_date, mo_write_ascii::write_configfile(), and mo_restart::write_restart_files().

Here is the call graph for this function:



17.11 mhm_papers.md File Reference

17.12 mo_anneal.f90 File Reference

Data Types

- interface [mo_anneal::anneal](#)
anneal
- interface [mo_anneal::gettemperature](#)
GetTemperature.
- interface [mo_anneal::generate_neighborhood_weight](#)

Modules

- module [mo_anneal](#)

Functions/Subroutines

- real(dp) function, dimension(size(para, 1)) [mo_anneal::anneal_dp](#) (cost, para, prange, prange_func, temp, Dt, nITERmax, Len, nST, eps, acc, seeds, printflag, maskpara, weight, changeParaMode, reflectionFlag, pertubFlexFlag, maxit, undef_funcval, tmp_file, funcbest, history)
- real(dp) function [mo_anneal::gettemperature_dp](#) (paraset, cost, acc_goal, prange, prange_func, samplesize, maskpara, seeds, printflag, weight, maxit, undef_funcval)
- real(dp) function [mo_anneal::pargen_anneal_dp](#) (old, dMax, oMin, oMax, RN)
- real(dp) function [mo_anneal::pargen_dds_dp](#) (old, perturb, oMin, oMax, RN)
- real(dp) function [mo_anneal::dchange_dp](#) (delta, iDigit, isZero)
- subroutine [mo_anneal::generate_neighborhood_weight_dp](#) (truepara, cum_weight, save_state_xor, iTotale←Counter, nITERmax, neighborhood)

17.13 mo_append.f90 File Reference

Data Types

- interface [mo_append::append](#)
Append (rows) scalars, vectors, and matrixes onto existing array.
- interface [mo_append::paste](#)
Paste (columns) scalars, vectors, and matrixes onto existing array.

Modules

- module [mo_append](#)
Append values on existing arrays.

Functions/Subroutines

- subroutine [mo_append::append_i4_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_i4_v_v](#) (vec1, vec2)
- subroutine [mo_append::append_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_i4_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_i8_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_i8_v_v](#) (vec1, vec2)
- subroutine [mo_append::append_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_i8_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_sp_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_sp_v_v](#) (vec1, vec2)

- subroutine [mo_append::append_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_sp_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_dp_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_dp_v_v](#) (vec1, vec2)
- subroutine [mo_append::append_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_dp_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_char_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_char_v_v](#) (vec1, vec2)
- subroutine [mo_append::append_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_char_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_lgt_v_s](#) (vec1, sca2)
- subroutine [mo_append::append_lgt_v_v](#) (vec1, vec2)
- subroutine [mo_append::append_lgt_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::append_lgt_3d](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_i4_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_i4_m_v](#) (mat1, vec2, fill_value)
- subroutine [mo_append::paste_i4_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_i8_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_i8_m_v](#) (mat1, vec2, fill_value)
- subroutine [mo_append::paste_i8_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_sp_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_sp_m_v](#) (mat1, vec2, fill_value)
- subroutine [mo_append::paste_sp_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_dp_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_dp_m_v](#) (mat1, vec2, fill_value)
- subroutine [mo_append::paste_dp_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_char_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_char_m_v](#) (mat1, vec2, fill_value)
- subroutine [mo_append::paste_char_m_m](#) (mat1, mat2, fill_value)
- subroutine [mo_append::paste_lgt_m_s](#) (mat1, sca2)
- subroutine [mo_append::paste_lgt_m_v](#) (mat1, vec2)
- subroutine [mo_append::paste_lgt_m_m](#) (mat1, mat2)

17.14 mo_canopy_interc.f90 File Reference

Modules

- module [mo_canopy_interc](#)
Canopy interception.

Functions/Subroutines

- elemental pure subroutine, public [mo_canopy_interc::canopy_interc](#) (pet, interc_max, precip, interc, through-fall, evap_canopy)
Canopy interception.

17.15 mo_common_variables.f90 File Reference

Data Types

- type [mo_common_variables::period](#)

Modules

- module [mo_common_variables](#)
Provides structures needed by mHM and mRM.

Variables

- integer(i4), parameter, public [mo_common_variables::nprocesses](#) = 10
- integer(i4), dimension(nprocesses, 3), public [mo_common_variables::processmatrix](#)
- integer(i4), public [mo_common_variables::opti_method](#)
- integer(i4), public [mo_common_variables::opti_function](#)
- logical, public [mo_common_variables::optimize](#)
- logical, public [mo_common_variables::optimize_restart](#)
- integer(i8), public [mo_common_variables::seed](#)
- integer(i4), public [mo_common_variables::niterations](#)
- real(dp), public [mo_common_variables::dds_r](#)
- real(dp), public [mo_common_variables::sa_temp](#)
- integer(i4), public [mo_common_variables::sce_ngs](#)
- integer(i4), public [mo_common_variables::sce_npg](#)
- integer(i4), public [mo_common_variables::sce_nps](#)
- logical, public [mo_common_variables::mcmc_opti](#)
- integer(i4), parameter, public [mo_common_variables::nerror_model](#) = 2
- real(dp), dimension(nerror_model), public [mo_common_variables::mcmc_error_params](#)
- real(dp), dimension(:, :), allocatable, public [mo_common_variables::global_parameters](#)
- character(256), dimension(:), allocatable, public [mo_common_variables::global_parameters_name](#)
- logical [mo_common_variables::alma_convention](#)

17.16 mo_constants.f90 File Reference

Modules

- module [mo_constants](#)
Provides computational, mathematical, physical, and file constants.

Variables

- real(dp), parameter [mo_constants::eps_dp](#) = epsilon(1.0_dp)
epsilon(1.0) in double precision
- real(sp), parameter [mo_constants::eps_sp](#) = epsilon(1.0_sp)
epsilon(1.0) in single precision
- real(dp), parameter [mo_constants::pi_dp](#) = 3.141592653589793238462643383279502884197_dp
Pi in double precision.
- real(sp), parameter [mo_constants::pi_sp](#) = 3.141592653589793238462643383279502884197_sp
Pi in single precision.
- real(dp), parameter [mo_constants::pio2_dp](#) = 1.57079632679489661923132169163975144209858_dp
Pi/2 in double precision.
- real(sp), parameter [mo_constants::pio2_sp](#) = 1.57079632679489661923132169163975144209858_sp
Pi/2 in single precision.
- real(dp), parameter [mo_constants::twopi_dp](#) = 6.283185307179586476925286766559005768394_dp
*2*Pi in double precision*
- real(sp), parameter [mo_constants::twopi_sp](#) = 6.283185307179586476925286766559005768394_sp

- real(dp), parameter `mo_constants::rho0_dp` = 1.225_dp
standard density [kg m⁻³] in double precision
- real(sp), parameter `mo_constants::rho0_sp` = 1.225_sp
standard density [kg m⁻³] in single precision
- real(dp), parameter `mo_constants::cp0_dp` = 1005.0_dp
specific heat capacity of air [J kg⁻¹ K⁻¹] in double precision
- real(sp), parameter `mo_constants::cp0_sp` = 1005.0_sp
specific heat capacity of air [J kg⁻¹ K⁻¹] in single precision
- real(dp), parameter `mo_constants::pi_d` = 3.141592653589793238462643383279502884197_dp
Pi in double precision.
- real(sp), parameter `mo_constants::pi` = 3.141592653589793238462643383279502884197_sp
Pi in single precision.
- real(dp), parameter `mo_constants::pio2_d` = 1.57079632679489661923132169163975144209858_dp
Pi/2 in double precision.
- real(sp), parameter `mo_constants::pio2` = 1.57079632679489661923132169163975144209858_sp
Pi/2 in single precision.
- real(dp), parameter `mo_constants::twopi_d` = 6.283185307179586476925286766559005768394_dp
*2*Pi in double precision*
- real(sp), parameter `mo_constants::twopi` = 6.283185307179586476925286766559005768394_sp
*2*Pi in single precision*
- real(dp), parameter `mo_constants::sqrt2_d` = 1.41421356237309504880168872420969807856967_dp
Square root of 2 in double precision.
- real(sp), parameter `mo_constants::sqrt2` = 1.41421356237309504880168872420969807856967_sp
Square root of 2 in single precision.
- real(dp), parameter `mo_constants::euler_d` = 0.5772156649015328606065120900824024310422_dp
Euler's constant in double precision.
- real(sp), parameter `mo_constants::euler` = 0.5772156649015328606065120900824024310422_sp
Euler's constant in single precision.
- integer, parameter `mo_constants::nin` = input_unit
Standard input file unit.
- integer, parameter `mo_constants::nout` = output_unit
Standard output file unit.
- integer, parameter `mo_constants::nerr` = error_unit
Standard error file unit.
- integer, parameter `mo_constants::nnml` = 100
Standard file unit for namelist.

17.17 mo_corr.f90 File Reference

Data Types

- interface `mo_corr::autocoeffk`
- interface `mo_corr::autocorr`
- interface `mo_corr::corr`
- interface `mo_corr::crosscoeffk`
- interface `mo_corr::crosscorr`
- interface `mo_corr::arth`
- interface `mo_corr::four1`
- interface `mo_corr::fourrow`
- interface `mo_corr::realft`
- interface `mo_corr::swap`

Modules

- module [mo_corr](#)

Functions/Subroutines

- real(sp) function, dimension(n) [mo_corr::arth_sp](#) (first, increment, n)
- real(dp) function, dimension(n) [mo_corr::arth_dp](#) (first, increment, n)
- integer(i4) function, dimension(n) [mo_corr::arth_i4](#) (first, increment, n)
- real(dp) function [mo_corr::autocoeffk_dp](#) (x, k, mask)
- real(sp) function [mo_corr::autocoeffk_sp](#) (x, k, mask)
- real(dp) function, dimension(size(k)) [mo_corr::autocoeffk_1d_dp](#) (x, k, mask)
- real(sp) function, dimension(size(k)) [mo_corr::autocoeffk_1d_sp](#) (x, k, mask)
- real(dp) function [mo_corr::autocorr_dp](#) (x, k, mask)
- real(sp) function [mo_corr::autocorr_sp](#) (x, k, mask)
- real(dp) function, dimension(size(k)) [mo_corr::autocorr_1d_dp](#) (x, k, mask)
- real(sp) function, dimension(size(k)) [mo_corr::autocorr_1d_sp](#) (x, k, mask)
- real(dp) function, dimension(size(data1)) [mo_corr::corr_dp](#) (data1, data2, nadjust, nhigh, nwin)
- real(sp) function, dimension(size(data1)) [mo_corr::corr_sp](#) (data1, data2, nadjust, nhigh, nwin)
- real(dp) function [mo_corr::crosscoeffk_dp](#) (x, y, k, mask)
- real(sp) function [mo_corr::crosscoeffk_sp](#) (x, y, k, mask)
- real(dp) function [mo_corr::crosscorr_dp](#) (x, y, k, mask)
- real(sp) function [mo_corr::crosscorr_sp](#) (x, y, k, mask)
- subroutine [mo_corr::four1_sp](#) (data, isign)
- subroutine [mo_corr::four1_dp](#) (data, isign)
- subroutine [mo_corr::fourrow_sp](#) (data, isign)
- subroutine [mo_corr::fourrow_dp](#) (data, isign)
- subroutine [mo_corr::realft_dp](#) (data, isign, zdata)
- subroutine [mo_corr::realft_sp](#) (data, isign, zdata)
- subroutine [mo_corr::swap_1d_spc](#) (a, b)
- subroutine [mo_corr::swap_1d_dpc](#) (a, b)
- complex(dpc) function, dimension(nn) [mo_corr::zroots_unity_dp](#) (n, nn)
- complex(spc) function, dimension(nn) [mo_corr::zroots_unity_sp](#) (n, nn)

Variables

- integer(i4), parameter [mo_corr::npar_arth](#) =16
- integer(i4), parameter [mo_corr::npar2_arth](#) =8

17.18 mo_dds.f90 File Reference

Modules

- module [mo_dds](#)

Dynamically Dimensioned Search (DDS)

Functions/Subroutines

- real(dp) function, dimension(size(pini)), public [mo_dds::dds](#) (obj_func, pini, prange, r, seed, maxiter, maxit, mask, tmp_file, funcbest, history)
DDS.
- real(dp) function, dimension(size(pini)), public [mo_dds::mdds](#) (obj_func, pini, prange, seed, maxiter, maxit, mask, tmp_file, funcbest, history)
MDDS.
- subroutine [mo_dds::neigh_value](#) (x_cur, x_min, x_max, r, new_value)

17.19 mo_errormeasures.f90 File Reference

Data Types

- interface [mo_errormeasures::bias](#)
- interface [mo_errormeasures::kge](#)
Kling-Gupta-Efficiency measure.
- interface [mo_errormeasures::kgenocorr](#)
Kling-Gupta-Efficiency measure without correlation.
- interface [mo_errormeasures::lnnse](#)
- interface [mo_errormeasures::mae](#)
- interface [mo_errormeasures::mse](#)
- interface [mo_errormeasures::nse](#)
- interface [mo_errormeasures::sae](#)
- interface [mo_errormeasures::sse](#)
- interface [mo_errormeasures::rmse](#)

Modules

- module [mo_errormeasures](#)

Functions/Subroutines

- real(sp) function [mo_errormeasures::bias_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::bias_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::bias_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::bias_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::bias_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::bias_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kge_sp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kge_sp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kge_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kge_dp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kge_dp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kge_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kgenocorr_sp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kgenocorr_sp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::kgenocorr_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kgenocorr_dp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kgenocorr_dp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::kgenocorr_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::lnnse_sp_1d](#) (x, y, mask)

- real(dp) function [mo_errormeasures::lnnse_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::lnnse_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::lnnse_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::lnnse_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::lnnse_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mae_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mae_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mae_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mae_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mae_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mae_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mse_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mse_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mse_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mse_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::mse_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::mse_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::nse_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::nse_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::nse_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::nse_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::nse_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::nse_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sae_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sae_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sae_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sae_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sae_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sae_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sse_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sse_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sse_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sse_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::sse_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::sse_dp_3d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::rmse_sp_1d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::rmse_dp_1d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::rmse_sp_2d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::rmse_dp_2d](#) (x, y, mask)
- real(sp) function [mo_errormeasures::rmse_sp_3d](#) (x, y, mask)
- real(dp) function [mo_errormeasures::rmse_dp_3d](#) (x, y, mask)

17.20 mo_file.f90 File Reference

Modules

- module [mo_file](#)

Provides file names and units for mHM.

Variables

- character(len= *), parameter `mo_file::version` = '5.8'
Current mHM model version.
- character(len= *), parameter `mo_file::version_date` = 'December 2017'
Time of current mHM model version release.
- character(len= *), parameter `mo_file::file_main` = 'mhm_driver.f90'
Driver file.
- character(len= *), parameter `mo_file::file_namelist` = 'mhm.nml'
Namelist file name.
- integer, parameter `mo_file::unamelist` = 30
Unit for namelist.
- character(len= *), parameter `mo_file::file_namelist_param` = 'mhm_parameter.nml'
Parameter namelists file name.
- integer, parameter `mo_file::unamelist_param` = 31
Unit for namelist.
- character(len= *), parameter `mo_file::file_meteo_header` = 'header.txt'
Input nCols and nRows of binary meteo files are in header file.
- integer, parameter `mo_file::umeteo_header` = 50
Unit for meteo header file.
- character(len= *), parameter `mo_file::file_meteo_binary_end` = '.bin'
File ending of meteo files.
- integer, parameter `mo_file::umeteo` = 51
Unit for meteo files.
- character(len= *), parameter `mo_file::file_soil_database` = 'soil_classdefinition.txt'
Soil database file (iFlag_soilDB = 0) = classical mHM format.
- character(len= *), parameter `mo_file::file_soil_database_1` = 'soil_classdefinition_iFlag_soilDB_1.txt'
Soil database file (iFlag_soilDB = 1)
- integer, parameter `mo_file::usoil_database` = 52
Unit for soil data base.
- character(len= *), parameter `mo_file::file_dem` = 'dem.asc'
DEM input data file.
- integer, parameter `mo_file::udem` = 53
Unit for DEM input data file.
- character(len= *), parameter `mo_file::file_slope` = 'slope.asc'
slope input data file
- integer, parameter `mo_file::uslope` = 54
Unit for slope input data file.
- character(len= *), parameter `mo_file::file_aspect` = 'aspect.asc'
aspect input data file
- integer, parameter `mo_file::uaspect` = 55
Unit for aspect input data file.
- character(len= *), parameter `mo_file::file_facc` = 'facc.asc'
flow accumulation input data file
- integer, parameter `mo_file::ufacc` = 56
Unit for flow accumulation input data file.
- character(len= *), parameter `mo_file::file_fdir` = 'fdir.asc'
flow direction input data file
- integer, parameter `mo_file::ufdir` = 57
Unit for flow direction input data file.
- character(len= *), parameter `mo_file::file_hydrogeoclass` = 'geology_class.asc'

- hydrogeological classes input data file*

 - integer, parameter `mo_file::uhydrogeoclass` = 58
Unit for hydrogeological classes input data file.
- character(len= *), parameter `mo_file::file_soilclass` = 'soil_class.asc'
soil classes input data file

 - integer, parameter `mo_file::usoilclass` = 59
Unit for soil classes input data file.
- character(len= *), parameter `mo_file::file_laiclass` = 'LAI_class.asc'
LAI classes input data file.

 - integer, parameter `mo_file::ulaiclass` = 60
Unit for LAI input data file.
- integer, parameter `mo_file::ulcoverclass` = 61
Unit for LCover input data file.
- character(len= *), parameter `mo_file::file_geolut` = 'geology_classdefinition.txt'
geological formation lookup table file

 - integer, parameter `mo_file::ugeolut` = 64
Unit for geological formation lookup table file.
- character(len= *), parameter `mo_file::file_lailut` = 'LAI_classdefinition.txt'
LAI classes lookup table file.

 - integer, parameter `mo_file::ulailut` = 65
Unit for LAI classes lookup table file.
- integer, parameter `mo_file::udischarge` = 66
unit for discharge time series
- character(len= *), parameter `mo_file::file_defoutput` = 'mhm_outputs.nml'
file defining mHM's outputs

 - integer, parameter `mo_file::udefoutput` = 67
Unit for file defining mHM's outputs.
- character(len= *), parameter `mo_file::file_config` = 'ConfigFile.log'
file containing mHM configuration

 - integer, parameter `mo_file::uconfig` = 68
Unit for file containing mHM configuration.
- character(len= *), parameter `mo_file::file_opti` = 'FinalParam.out'
file defining optimization outputs (objective and parameter set)

 - integer, parameter `mo_file::uopti` = 72
Unit for file optimization outputs (objective and parameter set)
- character(len= *), parameter `mo_file::file_opti_nml` = 'FinalParam.nml'
file defining optimization outputs in a namelist format (parameter set)

 - integer, parameter `mo_file::uopti_nml` = 73
Unit for file optimization outputs in a namelist format (parameter set)
- character(len= *), parameter `mo_file::file_daily_discharge` = 'daily_discharge.out'
file defining optimization outputs

 - integer, parameter `mo_file::udaily_discharge` = 74
Unit for file optimization outputs.
- character(len= *), parameter `mo_file::file_lai_header` = 'header.txt'
Input nCols and nRows of binary gridded LAI files are in header file.

 - integer, parameter `mo_file::ulai_header` = 75
Unit for LAI header file.
- character(len= *), parameter `mo_file::file_lai_binary_end` = '.bin'
Binary file ending of LAI files.

 - integer, parameter `mo_file::ulai` = 76
Unit for binary LAI files.
- integer, parameter `mo_file::utws` = 77
unit for tws time series

17.21 mo_finish.f90 File Reference

Modules

- module `mo_finish`
Finish a program gracefully.

Functions/Subroutines

- subroutine, public `mo_finish::finish` (name, text, unit)
Finish a program gracefully.

17.22 mo_global_variables.f90 File Reference

Data Types

- type `mo_global_variables::soiltype`
- type `mo_global_variables::twstructure`
- type `mo_global_variables::gridgeoref`
- type `mo_global_variables::basininfo`

Modules

- module `mo_global_variables`
Global variables ONLY used in reading, writing and startup.

Variables

- character(1024), public `mo_global_variables::project_details`
- character(1024), public `mo_global_variables::setup_description`
- character(1024), public `mo_global_variables::simulation_type`
- character(256), public `mo_global_variables::conventions`
- character(1024), public `mo_global_variables::contact`
- character(1024), public `mo_global_variables::mhm_details`
- character(1024), public `mo_global_variables::history`
- integer(i4), public `mo_global_variables::timestep`
- integer(i4), dimension(:), allocatable, public `mo_global_variables::timestep_model_inputs`
- real(dp), dimension(:), allocatable, public `mo_global_variables::resolutionhydrology`
- real(dp), dimension(:), allocatable, public `mo_global_variables::resolutionrouting`
- integer(i4), dimension(:), allocatable, public `mo_global_variables::l0_basin`
- logical, public `mo_global_variables::read_restart`
- logical, public `mo_global_variables::write_restart`
- logical, public `mo_global_variables::perform_mpr`
- logical, public `mo_global_variables::read_meteo_weights`
- character(256), public `mo_global_variables::inputformat_meteo_forcings`
- character(256), public `mo_global_variables::inputformat_gridded_lai`
- integer(i4), public `mo_global_variables::timestep_lai_input`
- integer(i4), public `mo_global_variables::timestep_sm_input`
- integer(i4), public `mo_global_variables::timestep_neutrons_input`
- integer(i4), public `mo_global_variables::timestep_et_input`
- integer(i4), public `mo_global_variables::iflag_cordinate_sys`

- integer(i4), public [mo_global_variables::iflag_soildb](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirmorpho](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirlcover](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirprecipitation](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirtemperature](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirminttemperature](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirmaxtemperature](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirnetradiation](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirabsvappressure](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirwindspeed](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirreferencecet](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirout](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirrestartout](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirrestartin](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirgridded_lai](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::filelatlon](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirsoil_moisture](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::filetw](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::dirneutrons](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::direvapotranspiration](#)
- character(256), public [mo_global_variables::dirconfigout](#)
- character(256), public [mo_global_variables::dircommonfiles](#)
- real(dp), dimension(:), allocatable, target, public [mo_global_variables::ycoor](#)
- real(dp), dimension(:), allocatable, target, public [mo_global_variables::xcoor](#)
- real(dp), dimension(:,:), allocatable, target, public [mo_global_variables::lons](#)
- real(dp), dimension(:,:), allocatable, target, public [mo_global_variables::lats](#)
- real(dp), public [mo_global_variables::c2tstu](#)
- integer(i4), public [mo_global_variables::ntstepday](#)
- integer(i4), parameter, public [mo_global_variables::routingstates](#) = 2
- real(dp), public [mo_global_variables::tillagedepth](#)
- integer(i4), public [mo_global_variables::nsoilhorizons_mhm](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::horizondepth_mhm](#)
- integer(i4), public [mo_global_variables::nsoiltypes](#)
- type(soiltype), public [mo_global_variables::soildb](#)
- type(twsstructure), public [mo_global_variables::basin_avg_tws_obs](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::basin_avg_tws_sim](#)
- integer(i4), public [mo_global_variables::nmeasperday_tws](#)
- integer(i4), public [mo_global_variables::ngeounits](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::geounitlist](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::geounitkar](#)
- real(dp), public [mo_global_variables::fracsealed_cityarea](#)
- integer(i4), public [mo_global_variables::nlcoverscene](#)
- character(256), dimension(:), allocatable, public [mo_global_variables::lcfilename](#)
- integer(i4), dimension(:,:), allocatable, public [mo_global_variables::lcyearid](#)
- integer(i4), public [mo_global_variables::nlaiclass](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::laiunitlist](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::laiut](#)
- type(gridgeoref), public [mo_global_variables::level0](#)
- type(gridgeoref), public [mo_global_variables::level1](#)
- type(gridgeoref), public [mo_global_variables::level2](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_longitude](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_latitude](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_longitude](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_latitude](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_rect_longitude](#)

- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_rect_latitude](#)
- type(period), dimension(:), allocatable, public [mo_global_variables::warmper](#)
- type(period), dimension(:), allocatable, public [mo_global_variables::evalper](#)
- type(period), dimension(:), allocatable, public [mo_global_variables::simper](#)
- type(period), public [mo_global_variables::readper](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::warmingdays](#)
- integer(i4), public [mo_global_variables::nbasins](#)
- type(basininfo), public [mo_global_variables::basin](#)
- logical, dimension(:), allocatable, target [mo_global_variables::l0_mask](#)
- real(dp), dimension(:), allocatable, target, public [mo_global_variables::l0_elev](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_slope](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_asp](#)
- integer(i4), dimension(:,:), allocatable, public [mo_global_variables::l0_soilid](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l0_geounit](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l0_lcover_lai](#)
- integer(i4), dimension(:,:), allocatable, target, public [mo_global_variables::l0_lcover](#)
- integer(i4), public [mo_global_variables::l0_ncells](#)
- integer(i4), dimension(:,:), allocatable, public [mo_global_variables::l0_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l0_id](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_slope_emp](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l0_gridded_lai](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l0_areacell](#)
- integer(i4), public [mo_global_variables::l1_ncells](#)
- integer(i4), dimension(:,:), allocatable, public [mo_global_variables::l1_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_id](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_areacell](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_upbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_downbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_leftbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_rightbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_global_variables::l1_ntcells_l0](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_temp_weights](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_pet_weights](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_pre_weights](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_pre](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_temp](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_pet](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_tmin](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_tmax](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_netrad](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_absvappress](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_windspeed](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_sm](#)
- logical, dimension(:,:), allocatable, public [mo_global_variables::l1_sm_mask](#)
- integer(i4) [mo_global_variables::ntimesteps_l1_sm](#)
- integer(i4) [mo_global_variables::nsoilhorizons_sm_input](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_neutronsdata](#)
- logical, dimension(:,:), allocatable, public [mo_global_variables::l1_neutronsdata_mask](#)
- integer(i4) [mo_global_variables::ntimesteps_l1_neutrons](#)
- real(dp), dimension(:,:), allocatable, public [mo_global_variables::l1_et](#)
- logical, dimension(:,:), allocatable, public [mo_global_variables::l1_et_mask](#)
- integer(i4) [mo_global_variables::ntimesteps_l1_et](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_fsealed](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_fforest](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_fperm](#)

- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_inter](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_snowpack](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_sealstw](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_soilmoist](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_unsatstw](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_satstw](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_neutrons](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_pet_calc](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_aetsoil](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_aetcanopy](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_aetsealed](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_baseflow](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_infilsoil](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_fastrunoff](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_melt](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_percol](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_preeffect](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_rain](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_runoffseal](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_slowrunoff](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_snow](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_throughfall](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_total_runoff](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_alpha](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_degdayinc](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_degdaymax](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_degdaynopre](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_degday](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_karstloss](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_fasp](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_petlaicorfactor](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_harsamcoeff](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_prietayalpha](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_aeroresist](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_surfresist](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_froots](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_maxinter](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_kfastflow](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_kslowflow](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_kbaseflow](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_kperco](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_soilmoistfc](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_soilmoistsat](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_soilmoistexp](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_jarvis_thresh_c1](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_tempthresh](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_unsatthresh](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::l1_sealedthresh](#)
- real(dp), dimension(:, :), allocatable, public [mo_global_variables::l1_wiltingpoint](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::evap_coeff](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fday_prec](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fnight_prec](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fday_pet](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fnight_pet](#)
- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fday_temp](#)

- real(dp), dimension(int(yearmonths, i4)), public [mo_global_variables::fnight_temp](#)
- integer(i4) [mo_global_variables::timestep_model_outputs](#)
- logical, dimension(noutflxstate) [mo_global_variables::outputflxstate](#)
- real(dp), dimension(:), allocatable, public [mo_global_variables::neutron_integral_afast](#)

17.23 mo_init_states.f90 File Reference

Modules

- module [mo_init_states](#)
Initialization of all state variables of mHM.

Functions/Subroutines

- subroutine, public [mo_init_states::variables_alloc](#) (iBasin)
Allocation of space for mHM related L1 and L11 variables.
- subroutine, public [mo_init_states::variables_default_init](#) ()
Default initialization mHM related L1 variables.
- subroutine, public [mo_init_states::get_basin_info](#) (iBasin, iLevel, nrows, ncols, ncells, iStart, iEnd, iStartMask, iEndMask, mask, xllcorner, yllcorner, cellsize)
Get basic basin information (e.g., nrows, ncols, indices, mask)
- subroutine, public [mo_init_states::calculate_grid_properties](#) (nrowsIn, ncolsIn, xllcornerIn, yllcornerIn, cellsizeIn, nodata_valueIn, aimingResolution, nrowsOut, ncolsOut, xllcornerOut, yllcornerOut, cellsizeOut, nodata_valueOut)
Calculates basic grid properties at a required coarser level using information of a given finer level.

17.24 mo_julian.f90 File Reference

Data Types

- interface [mo_julian::setcalendar](#)

Modules

- module [mo_julian](#)
Julian date conversion routines.

Functions/Subroutines

- subroutine [mo_julian::setcalendarstring](#) (selector)
Set module private variable calendar.
- subroutine [mo_julian::setcalendarinteger](#) (selector)
Set module private variable calendar.
- pure integer(i4) function [mo_julian::selectcalendar](#) (selector)
Select a calendar.
- elemental subroutine, public [mo_julian::caldat](#) (julian, dd, mm, yy, calendar)
Day, month and year from Julian day in the current or given calendar.
- elemental subroutine, public [mo_julian::dec2date](#) (julian, dd, mm, yy, hh, nn, ss, calendar)
Day, month, year, hour, minute, and second from fractional Julian day in the current or given calendar.

- elemental real(dp) function, public [mo_julian::date2dec](#) (dd, mm, yy, hh, nn, ss, calendar)
Fractional Julian day from day, month, year, hour, minute, second in the current calendar.
- elemental integer(i4) function, public [mo_julian::julday](#) (dd, mm, yy, calendar)
Julian day from day, month and year in the current or given calendar.
- elemental subroutine, public [mo_julian::caldatjulian](#) (julian, dd, mm, yy)
Day, month and year from Julian day.
- elemental real(dp) function [mo_julian::date2decjulian](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second.
- elemental subroutine [mo_julian::dec2datejulian](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day.
- elemental integer(i4) function [mo_julian::juldayjulian](#) (dd, mm, yy)
Julian day from day, month and year.
- elemental integer(i4) function, public [mo_julian::ndays](#) (dd, mm, yy)
IMSL Julian day from day, month and year.
- elemental subroutine, public [mo_julian::ndyin](#) (julian, dd, mm, yy)
Day, month and year from IMSL Julian day.
- elemental subroutine [mo_julian::caldat360](#) (julian, dd, mm, yy)
Day, month and year from Julian day in a 360 day calendar.
- elemental integer(i4) function [mo_julian::julday360](#) (dd, mm, yy)
Julian day from day, month and year in a 360_day calendar.
- elemental subroutine [mo_julian::dec2date360](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day in a 360_day calendar.
- elemental real(dp) function [mo_julian::date2dec360](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second in 360 day calendar.
- elemental subroutine [mo_julian::caldat365](#) (julian, dd, mm, yy)
Day, month and year from Julian day in a 365 day calendar.
- elemental integer(i4) function [mo_julian::julday365](#) (dd, mm, yy)
Julian day from day, month and year in a 365_day calendar.
- elemental subroutine [mo_julian::dec2date365](#) (julian, dd, mm, yy, hh, nn, ss)
Day, month, year, hour, minute, and second from fractional Julian day in a 365_day calendar.
- elemental real(dp) function [mo_julian::date2dec365](#) (dd, mm, yy, hh, nn, ss)
Fractional Julian day from day, month, year, hour, minute, second in 365 day calendar.

Variables

- integer(i4), save, private [mo_julian::calendar](#) = 1

17.25 mo_kind.f90 File Reference

Data Types

- type [mo_kind::sprs2_sp](#)
Single Precision Numerical Recipes types for sparse arrays.
- type [mo_kind::sprs2_dp](#)
Double Precision Numerical Recipes types for sparse arrays.

Modules

- module [mo_kind](#)
Define number representations.

Variables

- integer, parameter `mo_kind::i1` = SELECTED_INT_KIND(2)
1 Byte Integer Kind
- integer, parameter `mo_kind::i2` = c_short
2 Byte Integer Kind
- integer, parameter `mo_kind::i4` = c_int
4 Byte Integer Kind
- integer, parameter `mo_kind::i8` = c_long_long
8 Byte Integer Kind
- integer, parameter `mo_kind::sp` = c_float
Single Precision Real Kind.
- integer, parameter `mo_kind::dp` = c_double
Double Precision Real Kind.
- integer, parameter `mo_kind::spc` = c_float_complex
Single Precision Complex Kind.
- integer, parameter `mo_kind::dpc` = c_double_complex
Double Precision Complex Kind.
- integer, parameter `mo_kind::lgt` = KIND(.true.)
Logical Kind.

17.26 mo_linfit.f90 File Reference

Data Types

- interface `mo_linfit::linfit`
Fits a straight line to input data by minimizing χ^2 .

Modules

- module `mo_linfit`
Fitting a straight line.

Functions/Subroutines

- real(dp) function, dimension(:), allocatable `mo_linfit::linfit_dp` (x, y, a, b, siga, sigb, chi2, model2)
- real(sp) function, dimension(:), allocatable `mo_linfit::linfit_sp` (x, y, a, b, siga, sigb, chi2, model2)

17.27 mo_mcmc.f90 File Reference

Data Types

- interface `mo_mcmc::mcmc`
This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are either known or modeled).
- interface `mo_mcmc::mcmc_stddev`
This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution (measurement errors are neither known nor modeled).

Modules

- module [mo_mcmc](#)

This module is Monte Carlo Markov Chain sampling of a posterior parameter distribution.

Functions/Subroutines

- subroutine [mo_mcmc::mcmc_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, restart, restart_file, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)
- subroutine [mo_mcmc::mcmc_stddev_dp](#) (likelihood, para, rangePar, mcmc_paras, burnin_paras, seed_in, printflag_in, maskpara_in, tmp_file, loglike_in, ParaSelectMode_in, iter_burnin_in, iter_mcmc_in, chains_in, stepsize_in)
- real(dp) function [mo_mcmc::pargen_dp](#) (old, dMax, oMin, oMax, RN, inbound)
- real(dp) function [mo_mcmc::pargennorm_dp](#) (old, dMax, oMin, oMax, RN, inbound)
- recursive subroutine [mo_mcmc::generatenewparameterset_dp](#) (ParaSelectMode, paraold, truepara, rangePar, stepsize, save_state_2, save_state_3, paranew, ChangePara)

17.28 mo_message.f90 File Reference

Modules

- module [mo_message](#)

Write out concatenated strings.

Functions/Subroutines

- subroutine, public [mo_message::message](#) (t01, t02, t03, t04, t05, t06, t07, t08, t09, t10, uni, advance)
Write out several string concatenated either on screen or in a file.

Variables

- character(len=1024), public [mo_message::message_text](#) = "

17.29 mo_meteo_forcings.f90 File Reference

Modules

- module [mo_meteo_forcings](#)

Prepare meteorological forcings data for mHM.

Functions/Subroutines

- subroutine, public [mo_meteo_forcings::prepare_meteo_forcings_data](#) (iBasin, tt)
Prepare meteorological forcings data for a given variable.
- subroutine [mo_meteo_forcings::meteo_forcings_wrapper](#) (iBasin, dataPath, inputFormat, dataOut1, lower, upper, ncvarName)
Prepare meteorological forcings data for mHM at Level-1.
- subroutine [mo_meteo_forcings::meteo_weights_wrapper](#) (iBasin, read_meteo_weights, dataPath, dataOut1, lower, upper, ncvarName)

Prepare weights for meteorological forcings data for mHM at Level-1.

- subroutine [mo_meteo_forcings::chunk_config](#) (iBasin, tt, read_flag, readPer)
- logical function [mo_meteo_forcings::is_read](#) (iBasin, tt)

evaluate whether new chunk should be read at this timestep

- subroutine [mo_meteo_forcings::chunk_size](#) (iBasin, tt, readPer)

calculate beginning and end of read Period, i.e. that is length of current chunk to read

17.30 mo_mhm.f90 File Reference

Modules

- module [mo_mhm](#)

Call all main processes of mHM.

Functions/Subroutines

- subroutine, public [mo_mhm::mhm](#) (perform_mpr, read_states, fSealedInCity, timeStep_LAI_input, counter_↵_year, counter_month, counter_day, tt, time, processMatrix, c2TSTu, horizon_depth, nCells1, nHorizons_m_↵HM, ntimesteps_day, mask0, neutron_integral_AFast, iflag_soil_option, global_parameters, LCyearId, Geo_↵UnitList, GeoUnitKar, LAIUnitList, LAILUT, slope_emp0, L0_Latitude, cellId0, soilId0, L0_LCover_LAI, L_↵Cover0, Asp0, LAI0, geoUnit0, SDB_is_present, SDB_nHorizons, SDB_nTillHorizons, SDB_sand, SDB_↵clay, SDB_DbM, SDB_Wd, SDB_RZdepth, nTCells0_inL1, L0upBound_inL1, L0downBound_inL1, L0left_↵Bound_inL1, L0rightBound_inL1, latitude, evap_coeff, fday_prec, fnight_prec, fday_pet, fnight_pet, fday_↵temp, fnight_temp, temp_weights, pet_weights, pre_weights, read_meteo_weights, pet_in, tmin_in, tmax_↵in, netrad_in, absvappres_in, windspeed_in, prec_in, temp_in, yld, fForest1, fPerm1, fSealed1, interc, snowpack, sealedStorage, soilMoisture, unsatStorage, satStorage, neutrons, pet_calc, aet_soil, aet_canopy, aet_sealed, baseflow, infiltration, fast_interflow, melt, perc, prec_effect, rain, runoff_sealed, slow_interflow, snow, throughfall, total_runoff, alpha, deg_day_incr, deg_day_max, deg_day_noprec, deg_day, fAsp, pet_↵LAIcorFactorL1, HarSamCoeff, PrieTayAlpha, aeroResist, surfResist, frac_roots, interc_max, karst_loss, k0, k1, k2, kp, soil_moist_FC, soil_moist_sat, soil_moist_exponen, jarvis_thresh_c1, temp_thresh, unsat_thresh, water_thresh_sealed, wilting_point)

Pure mHM calculations.

17.31 mo_mhm_constants.f90 File Reference

Modules

- module [mo_mhm_constants](#)

Provides mHM specific constants.

Variables

- real(dp), parameter, public [mo_mhm_constants::h2odens](#) = 1000.0_dp
- integer(i4), parameter, public [mo_mhm_constants::nodata_i4](#) = -9999_i4
- real(dp), parameter, public [mo_mhm_constants::nodata_dp](#) = -9999._dp
- integer(i4), parameter, public [mo_mhm_constants::nlcover_class](#) = 3_i4
- integer(i4), parameter, public [mo_mhm_constants::ncolpars](#) = 5_i4
- integer(i4), parameter, public [mo_mhm_constants::maxnosoilhorizons](#) = 10_i4
- integer(i4), parameter, public [mo_mhm_constants::maxnobasins](#) = 50_i4
- integer(i4), parameter, public [mo_mhm_constants::maxnlcovers](#) = 50_i4
- integer(i4), parameter, public [mo_mhm_constants::maxgeounit](#) = 25_i4

- real(dp), parameter, public `mo_mhm_constants::p1_initstatefluxes` = 0.00_dp
- real(dp), parameter, public `mo_mhm_constants::p2_initstatefluxes` = 15.00_dp
- real(dp), parameter, public `mo_mhm_constants::p3_initstatefluxes` = 10.00_dp
- real(dp), parameter, public `mo_mhm_constants::p4_initstatefluxes` = 75.00_dp
- real(dp), parameter, public `mo_mhm_constants::p5_initstatefluxes` = 1500.00_dp
- real(dp), parameter, public `mo_mhm_constants::c1_initstatesm` = 0.25_dp
- integer(i4), parameter, public `mo_mhm_constants::noutflxstate` = 20_i4
- real(dp), parameter, public `mo_mhm_constants::dayhours` = 24.0_dp
- real(dp), parameter, public `mo_mhm_constants::yearmonths` = 12.0_dp
- integer(i4), parameter, public `mo_mhm_constants::yearmonths_i4` = 12
- real(dp), parameter, public `mo_mhm_constants::yeardays` = 365.0_dp
- real(dp), parameter, public `mo_mhm_constants::daysecs` = 86400.0_dp
- real(dp), parameter, public `mo_mhm_constants::bulkdens_orgmatter` = 0.224_dp
- real(dp), parameter, public `mo_mhm_constants::field_cap_c1` = -0.60_dp
- real(dp), parameter, public `mo_mhm_constants::field_cap_c2` = 2.0_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchten_sandtresh` = 66.5_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c1` = 1.392_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c2` = 0.418_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c3` = -0.024_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c4` = 1.212_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c5` = -0.704_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c6` = -0.648_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c7` = 0.023_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c8` = 0.044_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c9` = 3.168_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c10` = -2.562_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c11` = 7.0E-9_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c12` = 4.004_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c13` = 3.750_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c14` = -0.016_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c15` = -4.197_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c16` = 0.013_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c17` = 0.076_dp
- real(dp), parameter, public `mo_mhm_constants::vgenuchtenn_c18` = 0.276_dp
- real(dp), parameter, public `mo_mhm_constants::ks_c` = 10.0_dp
- real(dp), parameter, public `mo_mhm_constants::pwp_c` = 1.0_dp
- real(dp), parameter, public `mo_mhm_constants::pwp_matpot_thetar` = 15000.0_dp
- real(dp), parameter, public `mo_mhm_constants::stboltzmann` = 5.67e-08_dp
Stefan-Boltzmann constant [$W\ m^{-2}\ K^{-4}$].
- real(dp), parameter, public `mo_mhm_constants::harsamconst` = 17.800_dp
Hargreaves-Samani ref. ET formula [deg C].
- real(dp), parameter, public `mo_mhm_constants::windmeasheight` = 10.0_dp
assumed meteorol. measurement hight for estimation of aeroResist and surfResist
- real(dp), parameter, public `mo_mhm_constants::karman` = 0.41_dp
von karman constant
- real(dp), parameter, public `mo_mhm_constants::lai_factor_surfresi` = 0.3_dp
LAI factor for bulk surface resistance formulation.
- real(dp), parameter, public `mo_mhm_constants::lai_offset_surfresi` = 1.2_dp
LAI offset for bulk surface resistance formulation.
- real(dp), parameter, public `mo_mhm_constants::max_surfresist` = 250.0_dp
maximum bulk surface resistance
- real(dp), parameter, public `mo_mhm_constants::duffiedr` = 0.0330_dp
- real(dp), parameter, public `mo_mhm_constants::duffiedelta1` = 0.4090_dp

- real(dp), parameter, public [mo_mhm_constants::duffiedelta2](#) = 1.3900_dp
- real(dp), parameter, public [mo_mhm_constants::tetens_c1](#) = 0.6108_dp
Tetens's formula to calculate saturated vapour pressure.
- real(dp), parameter, public [mo_mhm_constants::tetens_c2](#) = 17.270_dp
- real(dp), parameter, public [mo_mhm_constants::tetens_c3](#) = 237.30_dp
- real(dp), parameter, public [mo_mhm_constants::satpressureslope1](#) = 4098.0_dp
calculation of the slope of the saturation vapour pressure curve following Tetens
- real(dp), parameter, public [mo_mhm_constants::desilets_a0](#) = 0.0808_dp
Neutrons and moisture: N0 formula, Desilets et al. 2010.
- real(dp), parameter, public [mo_mhm_constants::desilets_a1](#) = 0.372_dp
- real(dp), parameter, public [mo_mhm_constants::desilets_a2](#) = 0.115_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_bd](#) = 1.4020_dp
Neutrons and moisture: COSMIC, Shuttleworth et al. 2013.
- real(dp), parameter, public [mo_mhm_constants::cosmic_vwclat](#) = 0.0753_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_n](#) = 348.33_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_alpha](#) = 0.2392421548_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_l1](#) = 161.98621864_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_l2](#) = 129.14558985_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_l3](#) = 107.82204562_dp
- real(dp), parameter, public [mo_mhm_constants::cosmic_l4](#) = 3.1627190566_dp

17.32 mo_mhm_eval.f90 File Reference

Modules

- module [mo_mhm_eval](#)
Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

Functions/Subroutines

- subroutine, public [mo_mhm_eval::mhm_eval](#) (parameterset, runoff, sm_opti, basin_avg_tws, neutrons_opti, et_opti)
Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

17.33 mo_moment.f90 File Reference

Data Types

- interface [mo_moment::absdev](#)
- interface [mo_moment::average](#)
- interface [mo_moment::central_moment](#)
- interface [mo_moment::central_moment_var](#)
- interface [mo_moment::correlation](#)
- interface [mo_moment::covariance](#)
- interface [mo_moment::kurtosis](#)
- interface [mo_moment::mean](#)
- interface [mo_moment::mixed_central_moment](#)
- interface [mo_moment::mixed_central_moment_var](#)
- interface [mo_moment::moment](#)
- interface [mo_moment::skewness](#)
- interface [mo_moment::stddev](#)
- interface [mo_moment::variance](#)

Modules

- module [mo_moment](#)

Functions/Subroutines

- real(dp) function [mo_moment::absdev_dp](#) (dat, mask)
- real(sp) function [mo_moment::absdev_sp](#) (dat, mask)
- real(dp) function [mo_moment::average_dp](#) (dat, mask)
- real(sp) function [mo_moment::average_sp](#) (dat, mask)
- real(dp) function [mo_moment::central_moment_dp](#) (x, r, mask)
- real(sp) function [mo_moment::central_moment_sp](#) (x, r, mask)
- real(dp) function [mo_moment::central_moment_var_dp](#) (x, r, mask)
- real(sp) function [mo_moment::central_moment_var_sp](#) (x, r, mask)
- real(dp) function [mo_moment::correlation_dp](#) (x, y, mask)
- real(sp) function [mo_moment::correlation_sp](#) (x, y, mask)
- real(dp) function [mo_moment::covariance_dp](#) (x, y, mask)
- real(sp) function [mo_moment::covariance_sp](#) (x, y, mask)
- real(dp) function [mo_moment::kurtosis_dp](#) (dat, mask)
- real(sp) function [mo_moment::kurtosis_sp](#) (dat, mask)
- real(dp) function [mo_moment::mean_dp](#) (dat, mask)
- real(sp) function [mo_moment::mean_sp](#) (dat, mask)
- real(dp) function [mo_moment::mixed_central_moment_dp](#) (x, y, r, s, mask)
- real(sp) function [mo_moment::mixed_central_moment_sp](#) (x, y, r, s, mask)
- real(dp) function [mo_moment::mixed_central_moment_var_dp](#) (x, y, r, s, mask)
- real(sp) function [mo_moment::mixed_central_moment_var_sp](#) (x, y, r, s, mask)
- subroutine [mo_moment::moment_dp](#) (dat, average, variance, skewness, kurtosis, mean, stddev, absdev, mask)
- subroutine [mo_moment::moment_sp](#) (dat, average, variance, skewness, kurtosis, mean, stddev, absdev, mask)
- real(dp) function [mo_moment::stddev_dp](#) (dat, mask)
- real(sp) function [mo_moment::stddev_sp](#) (dat, mask)
- real(dp) function [mo_moment::skewness_dp](#) (dat, mask)
- real(sp) function [mo_moment::skewness_sp](#) (dat, mask)
- real(dp) function [mo_moment::variance_dp](#) (dat, mask)
- real(sp) function [mo_moment::variance_sp](#) (dat, mask)

17.34 mo_mpr_pet.f90 File Reference

Modules

- module [mo_mpr_pet](#)
scaling function for PET correction using LAI at level-0

Functions/Subroutines

- subroutine, public [mo_mpr_pet::pet_correctbylai](#) (param, nodata, LCOVER0, LAI0, mask0, cell_id0, upp_↵
row_L1, low_row_L1, lef_col_L1, rig_col_L1, nL0_in_L1, L1_petLAIcorFactor)
estimate PET correction factor based on LAI at L1
- subroutine, public [mo_mpr_pet::pet_correctbyasp](#) (ld0, latitude_l0, Asp0, param, nodata, fAsp0)
correction of PET

- subroutine, public [mo_mpr_pet::priestley_taylor_alpha](#) (LCover_LAI0, LAI_LUT, LAI_UnitList, param, mask0, nodata, cell_id0, nL0_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, priestley_taylor_alpha1)
Regionalization of priestley taylor alpha.
- subroutine, public [mo_mpr_pet::bulksurface_resistance](#) (LCover_LAI0, LAI_LUT, LAI_UnitList, param, mask0, nodata, cell_id0, nL0_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, bulksurface_resistance1)
Regionalization of bulk surface resistance.

17.35 mo_mpr_runoff.f90 File Reference

Modules

- module [mo_mpr_runoff](#)
multiscale parameter regionalization for runoff generation

Functions/Subroutines

- subroutine, public [mo_mpr_runoff::mpr_runoff](#) (LCOVER0, mask0, nodata, SMs_FC0, slope_emp0, KsVar_H0, param, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, nL0_in_L1, c2TSTu, L1_HL1, L1_K0, L1_K1, L1_alpha)
multiscale parameter regionalization for runoff parameters

17.36 mo_mpr_smhorizons.f90 File Reference

Modules

- module [mo_mpr_smhorizons](#)
setting up the soil moisture horizons

Functions/Subroutines

- subroutine, public [mo_mpr_smhorizons::mpr_smhorizons](#) (param, processMatrix, nodata, iFlag_soil, n_Horizons_mHM, HorizonDepth, LCOVER0, soilID0, nHorizons, nTillHorizons, thetaS_till, thetaFC_till, thetaPW_till, thetaS, thetaFC, thetaPW, Wd, Db, DbM, RZdepth, mask0, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, nL0_in_L1, L1_beta, L1_SMs, L1_FC, L1_PW, L1_fRoots)
upscale soil moisture horizons

17.37 mo_mpr_soilmoist.f90 File Reference

Modules

- module [mo_mpr_soilmoist](#)
Multiscale parameter regionalization (MPR) for soil moisture.

Functions/Subroutines

- subroutine, public [mo_mpr_soilmoist::mpr_sm](#) (param, nodata, iFlag_soil, is_present, nHorizons, nTill_Horizons, sand, clay, DbM, ID0, soilID0, LCover0, thetaS_till, thetaFC_till, thetaPW_till, thetaS, thetaFC, thetaPW, Ks, Db, KsVar_H0, KsVar_V0, SMs_FC0)
multiscale parameter regionalization for soil moisture

- elemental pure subroutine `mo_mpr_soilmoist::pwp` (Genu_Mual_n, Genu_Mual_alpha, thetaS, thetaPWP)
Permanent Wilting point.
- elemental pure subroutine `mo_mpr_soilmoist::field_cap` (thetaFC, Ks, thetaS, Genu_Mual_n)
calculates the field capacity
- subroutine `mo_mpr_soilmoist::genuchten` (thetaS, Genu_Mual_n, Genu_Mual_alpha, param, sand, clay, Db)
calculates the Genuchten shape parameter
- subroutine `mo_mpr_soilmoist::hydro_cond` (KS, param, sand, clay)
calculates the hydraulic conductivity Ks

17.38 mo_mrm_constants.f90 File Reference

Provides mRM specific constants.

Modules

- module `mo_mrm_constants`

Variables

- integer(i4), parameter, public `mo_mrm_constants::noutflxstate` = 1_i4
- integer(i4), parameter, public `mo_mrm_constants::nodata_i4` = -9999_i4
- real(dp), parameter, public `mo_mrm_constants::nodata_dp` = -9999._dp
- integer(i4), parameter, public `mo_mrm_constants::nroutingstates` = 2
- integer(i4), parameter, public `mo_mrm_constants::ncolpars` = 5_i4
- integer(i4), parameter, public `mo_mrm_constants::maxnogauges` = 50_i4
- integer(i4), parameter, public `mo_mrm_constants::maxnobasins` = 50_i4
- integer(i4), parameter, public `mo_mrm_constants::maxnlcovers` = 50_i4
- real(dp), parameter, public `mo_mrm_constants::hoursecs` = 3600.0_dp
- real(dp), parameter, public `mo_mrm_constants::p1_initstatefluxes` = 0.00_dp
- real(dp), parameter, public `mo_mrm_constants::rout_space_weight` = 0._dp
- real(dp), parameter, public `mo_mrm_constants::deltah` = 5.000_dp
- real(dp), dimension(19), parameter `mo_mrm_constants::given_ts` = (/ 60._dp, 120._dp, 180._dp, 240._dp, 300._dp, 360._dp, 600._dp, 720._dp, 900._dp, 1200._dp, 1800._dp, 3600._dp, 7200._dp, 10800._dp, 14400._dp, 21600._dp, 28800._dp, 43200._dp, 86400._dp/)

17.38.1 Detailed Description

Provides mRM specific constants.

Provides mRM specific constants such as flood plain elevation.

Author

Stephan Thober

Date

Aug 2015

17.39 mo_mrm_eval.f90 File Reference

Modules

- module [mo_mrm_eval](#)
Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

Functions/Subroutines

- subroutine, public [mo_mrm_eval::mrm_eval](#) (parameterset, runoff)
Runs mrm with a specific parameter set and returns required variables, e.g. runoff.

17.40 mo_mrm_file.f90 File Reference

Modules

- module [mo_mrm_file](#)
Provides file names and units for mRM.

Variables

- character(len= *), parameter [mo_mrm_file::version](#) = '1.1'
Current mHM model version.
- character(len= *), parameter [mo_mrm_file::version_date](#) = 'Nov 2016'
Time of current mHM model version release.
- character(len= *), parameter [mo_mrm_file::file_main](#) = 'mrm_driver.f90'
Driver file.
- character(len= *), parameter [mo_mrm_file::file_namelist_mrm](#) = 'mhm.nml'
Namelist file name.
- integer, parameter [mo_mrm_file::unamelist_mrm](#) = 40
Unit for namelist.
- character(len= *), parameter [mo_mrm_file::file_namelist_param_mrm](#) = 'mhm_parameter.nml'
Parameter namelists file name.
- integer, parameter [mo_mrm_file::unamelist_param](#) = 41
Unit for namelist.
- character(len= *), parameter [mo_mrm_file::file_dem](#) = 'dem.asc'
DEM input data file.
- integer, parameter [mo_mrm_file::udem](#) = 53
Unit for DEM input data file.
- character(len= *), parameter [mo_mrm_file::file_facc](#) = 'facc.asc'
flow accumulation input data file
- integer, parameter [mo_mrm_file::ufacc](#) = 56
Unit for flow accumulation input data file.
- character(len= *), parameter [mo_mrm_file::file_fdir](#) = 'fdir.asc'
flow direction input data file
- integer, parameter [mo_mrm_file::ufdir](#) = 57
Unit for flow direction input data file.
- integer, parameter [mo_mrm_file::ulcoverclass](#) = 61
Unit for LCover input data file.

- character(len= *), parameter `mo_mrm_file::file_gaugeloc` = 'idgauges.asc'
gauge location input data file
- integer, parameter `mo_mrm_file::ugaugeloc` = 62
Unit for gauge location input data file.
- integer, parameter `mo_mrm_file::udischarge` = 66
unit for discharge time series
- character(len= *), parameter `mo_mrm_file::file_defoutput` = 'mrm_outputs.nml'
file defining mRM's outputs
- integer, parameter `mo_mrm_file::udefoutput` = 67
Unit for file defining mRM's outputs.
- character(len= *), parameter `mo_mrm_file::file_config` = 'ConfigFile.log'
file defining mHM's outputs
- integer, parameter `mo_mrm_file::uconfig` = 68
Unit for file defining mHM's outputs.
- character(len= *), parameter `mo_mrm_file::file_opti` = 'FinalParam.out'
file defining optimization outputs (objective and p arameter set)
- integer, parameter `mo_mrm_file::uopti` = 72
Unit for file optimization outputs (objective and p arameter set)
- character(len= *), parameter `mo_mrm_file::file_opti_nml` = 'FinalParam.nml'
file defining optimization outputs in a namelist fo rmat (parameter set)
- integer, parameter `mo_mrm_file::uopti_nml` = 73
Unit for file optimization outputs in a namelist fo rmat (parameter set)
- character(len= *), parameter `mo_mrm_file::file_daily_discharge` = 'daily_discharge.out'
file defining optimazation outputs
- integer, parameter `mo_mrm_file::udaily_discharge` = 74
Unit for file optimazation outputs.
- character(len= *), parameter `mo_mrm_file::ncfile_discharge` = 'discharge.nc'
file defining optimazation outputs
- character(len= *), parameter `mo_mrm_file::file_mrm_output` = 'mRM_Fluxes_States.nc'
file containing mrm output

17.41 mo_mrm_global_variables.f90 File Reference

Data Types

- type `mo_mrm_global_variables::gridgeoref`
- type `mo_mrm_global_variables::gaugingstation`
- type `mo_mrm_global_variables::basininfo`

Modules

- module `mo_mrm_global_variables`
Global variables for mRM only.

Variables

- integer(i4) `mo_mrm_global_variables::mrm_coupling_mode`
- logical `mo_mrm_global_variables::is_start`
- character(1024), public `mo_mrm_global_variables::project_details`
- character(1024), public `mo_mrm_global_variables::setup_description`
- character(1024), public `mo_mrm_global_variables::simulation_type`
- character(256), public `mo_mrm_global_variables::conventions`
- character(1024), public `mo_mrm_global_variables::contact`
- character(1024), public `mo_mrm_global_variables::mhm_details`
- character(1024), public `mo_mrm_global_variables::history`
- integer(i4), public `mo_mrm_global_variables::timestep`
- integer(i4), dimension(:), allocatable, public `mo_mrm_global_variables::timestep_model_inputs`
- integer(i4), public `mo_mrm_global_variables::iflag_coordinate_sys`
- integer(i4), dimension(:), allocatable, public `mo_mrm_global_variables::l0_basin`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::resolutionrouting`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::resolutionhydrology`
- logical, public `mo_mrm_global_variables::read_restart`
- logical, public `mo_mrm_global_variables::write_restart`
- logical, public `mo_mrm_global_variables::perform_mpr`
- character(256), public `mo_mrm_global_variables::dirconfigout`
- character(256), public `mo_mrm_global_variables::dircommonfiles`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirmorpho`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirlcover`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirgauges`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirtotalrunoff`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirout`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirrestartout`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::dirrestartin`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::filelatlon`
- integer(i4), public `mo_mrm_global_variables::ntstepday`
- type(gridgeoref), public `mo_mrm_global_variables::level0`
- type(gridgeoref), public `mo_mrm_global_variables::level1`
- type(gridgeoref), public `mo_mrm_global_variables::level11`
- type(gridgeoref), public `mo_mrm_global_variables::level110`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::l0_longitude`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::l0_latitude`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::l11_rect_longitude`
- real(dp), dimension(:), allocatable, public `mo_mrm_global_variables::l11_rect_latitude`
- real(dp), dimension(:,:), allocatable, public `mo_mrm_global_variables::mrm_runoff`
- integer(i4), public `mo_mrm_global_variables::ngaugestotal`
- integer(i4), public `mo_mrm_global_variables::ninflowgaugestotal`
- integer(i4), public `mo_mrm_global_variables::nmeasperday`
- type(gaugingstation), public `mo_mrm_global_variables::gauge`
- type(gaugingstation), public `mo_mrm_global_variables::inflowgauge`
- real(dp), public `mo_mrm_global_variables::fracsealed_cityarea`
- integer(i4), public `mo_mrm_global_variables::nlcoverscene`
- character(256), dimension(:), allocatable, public `mo_mrm_global_variables::lcfilename`
- integer(i4), dimension(:,:), allocatable, public `mo_mrm_global_variables::lcyearid`
- type(period), dimension(:), allocatable, public `mo_mrm_global_variables::warmper`
- type(period), dimension(:), allocatable, public `mo_mrm_global_variables::evalper`
- type(period), dimension(:), allocatable, public `mo_mrm_global_variables::simper`
- type(period), dimension(:), allocatable, public `mo_mrm_global_variables::readper`
- integer(i4), dimension(:), allocatable, public `mo_mrm_global_variables::warmingdays_mrm`
- integer(i4), public `mo_mrm_global_variables::nbasins`

- type(basininfo), public [mo_mrm_global_variables::basin_mrm](#)
- logical, dimension(:), allocatable, target [mo_mrm_global_variables::l0_mask_mrm](#)
- real(dp), dimension(:), allocatable, target, public [mo_mrm_global_variables::l0_elev_read](#)
- real(dp), dimension(:), pointer, public [mo_mrm_global_variables::l0_elev_mrm](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_facc](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_fdir](#)
- integer(i4), dimension(:, :), pointer, public [mo_mrm_global_variables::l0_lcover_mrm](#)
- integer(i4), dimension(:, :), allocatable, target, public [mo_mrm_global_variables::l0_lcover_read](#)
- integer(i4), dimension(:, :), allocatable, public [mo_mrm_global_variables::l0_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_id](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_gaugeloc](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_inflowgaugeloc](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_l11_id](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l0_areacell](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_drasc](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_dracell](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_streamnet](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l0_floodplain](#)
- integer(i4) [mo_mrm_global_variables::l0_ncells](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_l1_id](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l1_areacell](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l1_id](#)
- integer(i4) [mo_mrm_global_variables::l1_ncells](#)
- real(dp), dimension(:, :), allocatable, public [mo_mrm_global_variables::l1_total_runoff_in](#)
- integer(i4), public [mo_mrm_global_variables::l11_ncells](#)
- integer(i4), dimension(:, :), allocatable, public [mo_mrm_global_variables::l11_cellcoor](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l1_l11_id](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_areacell](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_id](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_fdir](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_noutlets](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_upbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_downbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_leftbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_rightbound_l0](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_upbound_l1](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_downbound_l1](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_leftbound_l1](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_rightbound_l1](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_rowout](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_colout](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_qmod](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_qout](#)
- real(dp), dimension(:, :), allocatable, public [mo_mrm_global_variables::l11_qtin](#)
- real(dp), dimension(:, :), allocatable, public [mo_mrm_global_variables::l11_qtr](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_fracfpimp](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_fromn](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_ton](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_netperm](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_frow](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_fcol](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_trow](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_tcol](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_rorder](#)
- integer(i4), dimension(:), allocatable, public [mo_mrm_global_variables::l11_label](#)

- logical, dimension(:), allocatable, public [mo_mrm_global_variables::l11_sink](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_length](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_afloodplain](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_slope](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_k](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_xi](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_tsroute](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_c1](#)
- real(dp), dimension(:), allocatable, public [mo_mrm_global_variables::l11_c2](#)
- integer(i4) [mo_mrm_global_variables::timestep_model_outputs_mrm](#)
- logical, dimension(noutflxstate) [mo_mrm_global_variables::outputflxstate_mrm](#)

17.42 mo_mrm_init.f90 File Reference

Modules

- module [mo_mrm_init](#)
Wrapper for initializing Routing.

Functions/Subroutines

- subroutine, public [mo_mrm_init::mrm_init](#) (L0_mask, L0_elev, L0_LCover)
Initialize all mRM variables at all levels (i.e., L0, L1, and L11).
- subroutine [mo_mrm_init::print_startup_message](#) ()
- subroutine [mo_mrm_init::config_output](#) ()
- subroutine, public [mo_mrm_init::variables_default_init_routing](#) ()
Default initialization mRM related L11 variables.
- subroutine [mo_mrm_init::l0_check_input_routing](#) (iBasin)
- subroutine [mo_mrm_init::variables_alloc_routing](#) (iBasin)
- subroutine [mo_mrm_init::mrm_init_param](#) (iBasin, param)
- subroutine, public [mo_mrm_init::mrm_update_param](#) (iBasin, param)

17.43 mo_mrm_mpr.f90 File Reference

Modules

- module [mo_mrm_mpr](#)
Perform Multiscale Parameter Regionalization on Routing Parameters.

Functions/Subroutines

- subroutine, public [mo_mrm_mpr::reg_rout](#) (param, length, slope, fFPimp, TS, C1, C2)
Regionalized routing.

17.44 mo_mrm_net_startup.f90 File Reference

Modules

- module [mo_mrm_net_startup](#)
Startup drainage network for mHM.

Functions/Subroutines

- subroutine, public [mo_mrm_net_startup::l11_variable_init](#) (iBasin)
Cell numbering at ROUTING LEVEL-11.
- subroutine, public [mo_mrm_net_startup::l11_flow_direction](#) (iBasin)
Determine the flow direction of the upscaled river network at level L11.
- subroutine, public [mo_mrm_net_startup::l11_set_network_topology](#) (iBasin)
Set network topology.
- subroutine, public [mo_mrm_net_startup::l11_routing_order](#) (iBasin)
Find routing order, headwater cells and sink.
- subroutine, public [mo_mrm_net_startup::l11_link_location](#) (iBasin)
Estimate the LO (row,col) location for each routing link at level L11.
- subroutine, public [mo_mrm_net_startup::l11_set_drain_outlet_gauges](#) (iBasin)
Draining cell identification and Set gauging node.
- subroutine, public [mo_mrm_net_startup::l11_stream_features](#) (iBasin)
Stream features (stream network and floodplain)
- subroutine, public [mo_mrm_net_startup::l11_fraction_sealed_floodplain](#) (nLinks, LCover0, floodPlain0, areaCell0, nLinkAFloodPlain, LCClassImp, nLinkFracFPimp)
If a variable is added or removed here, then it also has to be added or removed in the subroutine L11_config_set in module [mo_restart](#) and in the subroutine set_L11_config in module [mo_set_netcdf_restart](#).
- subroutine [mo_mrm_net_startup::moveup](#) (elev0, fDir0, fi, fj, ss, nn)
- subroutine [mo_mrm_net_startup::movedownonecell](#) (fDir, iRow, jCol)
- subroutine [mo_mrm_net_startup::celllength](#) (iBasin, fDir, iRow, jCol, iCoorSystem, length)
- subroutine, public [mo_mrm_net_startup::get_distance_two_lat_lon_points](#) (lat1, long1, lat2, long2, distance_out)
estimate distance in [m] between two points in a lat-lon

17.45 mo_mrm_objective_function_runoff.f90 File Reference

Modules

- module [mo_mrm_objective_function_runoff](#)
Objective Functions for Optimization of mHM/mRM against runoff.

Functions/Subroutines

- real(dp) function, public [mo_mrm_objective_function_runoff::single_objective_runoff](#) (parameterset)
Wrapper for objective functions optimizing against runoff.
- subroutine, public [mo_mrm_objective_function_runoff::multi_objective_runoff](#) (parameterset, multi_↔ objectives)
Wrapper for multi-objective functions where at least one is regarding runoff.
- real(dp) function, public [mo_mrm_objective_function_runoff::loglikelihood](#) (parameterset)
Wrapper for loglikelihood functions.
- real(dp) function, public [mo_mrm_objective_function_runoff::loglikelihood_stddev](#) (parameterset, stddev, stddev_new, likeli_new)
Logarithmic likelihood function with removed linear trend and Lag(1)-autocorrelation.
- real(dp) function [mo_mrm_objective_function_runoff::loglikelihood_evin2013_2](#) (parameterset, regularize)
Logarithmised likelihood with linear error model and lag(1)-autocorrelation of the relative errors.
- real(dp) function [mo_mrm_objective_function_runoff::parameter_regularization](#) (paraset, prior, bounds, mask)

- real(dp) function [mo_mrm_objective_function_runoff::loglikelihood_trend_no_autocorr](#) (parameterset, stddev_old, stddev_new, likeli_new)
Logarithmic likelihood function with linear trend removed.
- real(dp) function [mo_mrm_objective_function_runoff::objective_lnnse](#) (parameterset)
Objective function of logarithmic NSE.
- real(dp) function [mo_mrm_objective_function_runoff::objective_sse](#) (parameterset)
Objective function of SSE.
- real(dp) function [mo_mrm_objective_function_runoff::objective_nse](#) (parameterset)
Objective function of NSE.
- real(dp) function [mo_mrm_objective_function_runoff::objective_equal_nse_lnnse](#) (parameterset)
Objective function equally weighting NSE and lnNSE.
- real(dp) function, dimension(2) [mo_mrm_objective_function_runoff::multi_objective_nse_lnnse](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [mo_mrm_objective_function_runoff::multi_objective_lnnse_highflow_lnnse_lowflow](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [mo_mrm_objective_function_runoff::multi_objective_lnnse_highflow_lnnse_lowflow_2](#) (parameterset)
Multi-objective function with NSE and lnNSE.
- real(dp) function, dimension(2) [mo_mrm_objective_function_runoff::multi_objective_ae_fdc_lsv_nse_djf](#) (parameterset)
Multi-objective function with absolute error of Flow Duration Curves low-segment volume and nse of DJF's discharge.
- real(dp) function [mo_mrm_objective_function_runoff::objective_power6_nse_lnnse](#) (parameterset)
Objective function of combined NSE and lnNSE with power of 5 i.e. the p-norm with p=5.
- real(dp) function [mo_mrm_objective_function_runoff::objective_kge](#) (parameterset)
Objective function of KGE.
- real(dp) function [mo_mrm_objective_function_runoff::objective_multiple_gauges_kge_power6](#) (parameterset)
combined objective function based on KGE raised to the power 6
- subroutine, public [mo_mrm_objective_function_runoff::extract_runoff](#) (gaugeld, runoff, runoff_agg, runoff_obs, runoff_obs_mask)
extracts runoff data from global variables
- subroutine [mo_mrm_objective_function_runoff::eval](#) (parameterset, runoff, basin_avg_tws)
returns mHM_eval or mRM_eval given preprocessor flag

17.46 mo_mrm_read_config.f90 File Reference

Modules

- module [mo_mrm_read_config](#)
read mRM config

Functions/Subroutines

- subroutine, public [mo_mrm_read_config::read_mrm_config_coupling](#) ()
Read the coupling mode of mRM.
- subroutine, public [mo_mrm_read_config::read_mrm_config](#) (do_message, readLatLon)
Read the general config of mRM.
- subroutine [mo_mrm_read_config::read_mrm_routing_params](#) (processCase, file_namelist)
- logical function [mo_mrm_read_config::in_bound](#) (params)

17.47 mo_mrm_read_data.f90 File Reference

Modules

- module [mo_mrm_read_data](#)
This module contains all routines to read mRM data from file.

Functions/Subroutines

- subroutine, public [mo_mrm_read_data::mrm_read_l0_data](#) (L0_mask, L0_elev, L0_LCover)
read L0 data from file
- subroutine, public [mo_mrm_read_data::mrm_l0_variable_init](#) (iBasin)
level 0 variable initialization
- subroutine, public [mo_mrm_read_data::mrm_l1_variable_init](#) (iBasin)
level 1 variable initialization
- subroutine, public [mo_mrm_read_data::mrm_read_discharge](#) ()
Read discharge timeseries from file.
- subroutine, public [mo_mrm_read_data::mrm_read_total_runoff](#) (iBasin)
read simulated runoff that is to be routed
- subroutine [mo_mrm_read_data::rotate_fdir_variable](#) (x)

17.48 mo_mrm_read_latlon.f90 File Reference

Modules

- module [mo_mrm_read_latlon](#)
reading latitude and longitude coordinates for each basin

Functions/Subroutines

- subroutine, public [mo_mrm_read_latlon::read_latlon](#) (ii)
reads latitude and longitude coordinates

17.49 mo_mrm_restart.f90 File Reference

Modules

- module [mo_mrm_restart](#)
Restart routines.

Functions/Subroutines

- subroutine, public [mo_mrm_restart::mrm_write_restart](#) (iBasin, OutPath)
write routing states and configuration
- subroutine, public [mo_mrm_restart::mrm_read_restart_states](#) (iBasin, InPath)
read routing states
- subroutine, public [mo_mrm_restart::mrm_read_restart_config](#) (iBasin, InPath)
reads Level 11 configuration from a restart directory

17.50 mo_mrm_routing.f90 File Reference

Modules

- module [mo_mrm_routing](#)
Performs runoff routing for mHM at level L11.

Functions/Subroutines

- subroutine, public [mo_mrm_routing::mrm_routing](#) (processCase, global_routing_param, L1_total_runoff, L1_areaCell, L1_L11_Id, L11_areaCell, L11_L1_Id, L11_netPerm, L11_fromN, L11_toN, L11_nOutlets, timestep, tsRoutFactor, nNodes, nInflowGauges, InflowGaugeIndexList, InflowGaugeHeadwater, InflowGaugeNodeList, InflowDischarge, nGauges, gaugeIndexList, gaugeNodeList, map_flag, L0_LCover, L0_floodPlain, L0_areaCell, L11_aFloodPlain, L11_length, L11_slope, L11_C1, L11_C2, L11_qOut, L11_qTIN, L11_qTR, L11_qMod, GaugeDischarge, L11_FracFPimp, do_mpr_routing)
route water given runoff
- subroutine [mo_mrm_routing::l11_runoff_acc](#) (qAll, efecArea, L1_L11_Id, L11_areaCell, L11_L1_Id, TS, map_flag, qAcc)
total runoff accumulation at L11.
- subroutine [mo_mrm_routing::add_inflow](#) (nInflowGauges, InflowIndexList, InflowHeadwater, InflowNodeList, QInflow, qOut)
Adds inflow discharge to the runoff produced at the cell where the inflow is occurring.
- subroutine [mo_mrm_routing::l11_routing](#) (nNodes, nLinks, netPerm, netLink_fromN, netLink_toN, netLink_C1, netLink_C2, netNode_qOUT, nInflowGauges, InflowHeadwater, InflowNodeList, netNode_qTIN, netNode_qTR, netNode_Qmod)
Performs runoff routing for mHM at L11 upscaled network ([Routing Network](#)).

17.51 mo_mrm_signatures.f90 File Reference

Modules

- module [mo_mrm_signatures](#)
Module with calculations for several hydrological signatures.

Functions/Subroutines

- real(dp) function, dimension(size(lags, 1)), public [mo_mrm_signatures::autocorrelation](#) (data, lags, mask)
Autocorrelation of a given data series.
- real(dp) function, dimension(size(quantiles, 1)), public [mo_mrm_signatures::flowdurationcurve](#) (data, quantiles, mask, concavity_index, mid_segment_slope, mhigh_segment_volume, high_segment_volume, low_segment_volume)
Flow duration curves.
- subroutine, public [mo_mrm_signatures::limb_densities](#) (data, mask, RLD, DLD)
Calculates limb densities.
- real(dp) function [mo_mrm_signatures::maximummonthlyflow](#) (data, mask, yr_start, mo_start, dy_start)
Maximum of average flows per months.
- subroutine, public [mo_mrm_signatures::moments](#) (data, mask, mean_data, stddev_data, median_data, max_data, mean_log, stddev_log, median_log, max_log)
Moments of data and log-transformed data, e.g. mean and standard deviation.
- real(dp) function, dimension(size(quantiles, 1)), public [mo_mrm_signatures::peakdistribution](#) (data, quantiles, mask, slope_peak_distribution)

Calculates the peak distribution.

- real(dp) function, public [mo_mrm_signatures::runoffratio](#) (data, basin_area, mask, precip_series, precip_↵sum, log_data)

Runoff ratio (accumulated daily discharge [mm/d] / accumulated daily precipitation [mm/d]).

- real(dp) function, public [mo_mrm_signatures::zeroflowratio](#) (data, mask)

Ratio of zero values to total number of data points.

17.52 mo_mrm_tools.f90 File Reference

Modules

- module [mo_mrm_tools](#)

Provide utility routines used within mRM.

Functions/Subroutines

- subroutine, public [mo_mrm_tools::get_basin_info_mrm](#) (iBasin, iLevel, nrows, ncols, ncells, iStart, iEnd, i↵StartMask, iEndMask, mask, xllcorner, yllcorner, cellsize)

Get basic basin information (e.g., nrows, ncols, indices, mask)

- subroutine, public [mo_mrm_tools::calculate_grid_properties](#) (nrowsIn, ncolsIn, xllcornerIn, yllcornerIn, cellsizeIn, nodata_valueIn, aimingResolution, nrowsOut, ncolsOut, xllcornerOut, yllcornerOut, cellsize↵Out, nodata_valueOut)

Calculates basic grid properties at a required coarser level using information of a given finer level.

17.53 mo_mrm_write.f90 File Reference

Modules

- module [mo_mrm_write](#)

write of discharge and restart files

Functions/Subroutines

- subroutine, public [mo_mrm_write::mrm_write](#) ()

write discharge and restart files

- subroutine [mo_mrm_write::write_configfile](#) ()

This modules writes the results of the configuration into an ASCII-file.

- subroutine [mo_mrm_write::write_daily_obs_sim_discharge](#) (Qobs, Qsim)

Write a file for the daily observed and simulated discharge timeseries during the evaluation period for each gauging station.

- subroutine, public [mo_mrm_write::mrm_write_output_fluxes](#) (iBasin, timeStep_model_outputs, warming_↵Days_mrm, newTime, nTimeSteps, nTStepDay, tt, day, month, year, timestep, mask11, L11_qmod)

write fluxes to netcdf output files

- subroutine, public [mo_mrm_write::mrm_write_optifile](#) (best_OF, best_paramSet, param_names)

Write briefly final optimization results.

- subroutine, public [mo_mrm_write::mrm_write_optinamelist](#) (parameters, maskpara, parameters_name)

Write final, optimized parameter set in a namelist format.

Variables

- integer(i4) [mo_mrm_write::day_counter](#)
- integer(i4) [mo_mrm_write::month_counter](#)
- integer(i4) [mo_mrm_write::year_counter](#)
- integer(i4) [mo_mrm_write::average_counter](#)
- type(outputdataset) [mo_mrm_write::nc](#)

17.54 mo_mrm_write_fluxes_states.f90 File Reference

Data Types

- interface [mo_mrm_write_fluxes_states::outputvariable](#)
- interface [mo_mrm_write_fluxes_states::outputvariable](#)
- interface [mo_mrm_write_fluxes_states::outputdataset](#)
- interface [mo_mrm_write_fluxes_states::outputdataset](#)

Modules

- module [mo_mrm_write_fluxes_states](#)
Creates NetCDF output for different fluxes and state variables of mHM.

Functions/Subroutines

- type(outputvariable) function [mo_mrm_write_fluxes_states::newoutputvariable](#) (nc, name, dtype, dims, ncells, mask, avg)
Initialize OutputVariable.
- subroutine [mo_mrm_write_fluxes_states::updatevariable](#) (self, data)
Update OutputVariable.
- subroutine [mo_mrm_write_fluxes_states::writevariabletimestep](#) (self, timestep)
Write timestep to file.
- type(outputdataset) function [mo_mrm_write_fluxes_states::newoutputdataset](#) (ibasin, mask)
Initialize OutputDataset.
- subroutine [mo_mrm_write_fluxes_states::updatedataset](#) (self, sidx, eid, L11_Qmod)
Update all variables.
- subroutine [mo_mrm_write_fluxes_states::writetimestep](#) (self, timestep)
Write all accumulated data.
- subroutine [mo_mrm_write_fluxes_states::close](#) (self)
Close the file.
- type(ncdataset) function [mo_mrm_write_fluxes_states::createoutputfile](#) (ibasin)
Create and initialize output file.
- subroutine [mo_mrm_write_fluxes_states::writevariableattributes](#) (var, long_name, unit)
Write output variable attributes.
- subroutine [mo_mrm_write_fluxes_states::mapcoordinates](#) (ibasin, level, y, x)
Generate map coordinates.
- subroutine [mo_mrm_write_fluxes_states::geocoordinates](#) (ibasin, level, lat, lon)
Generate geographic coordinates.
- character(16) function [mo_mrm_write_fluxes_states::fluxesunit](#) (ibasin)
Generate a unit string.

17.55 mo_multi_param_reg.f90 File Reference

Modules

- module [mo_multi_param_reg](#)
Multiscale parameter regionalization (MPR).

Functions/Subroutines

- subroutine, public [mo_multi_param_reg::mpr](#) (proc_Mat, iFlag_soil, param, nodata, mask0, geoUnit0, geoUnitList, GeoUnitKar, LAI_LUT, LAI_UnitList, SDB_is_present, SDB_nHorizons, SDB_nTillHorizons, SDB_sand, SDB_clay, SDB_DbM, SDB_Wd, SDB_RZdepth, nHorizons_mHM, horizon_depth, c2TSTu, fForest1, flperm1, fPerm1, soilId0, Asp0, LCover_LAI0, LCover0, slope_emp0, cell_id0, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, nL0_in_L1, latitude, alpha1, IDDP1, DDmax1, DD1, fAsp1, HarSamCoeff1, PrieTayAlpha1, aeroResist1, surfResist1, fRoots1, K0_1, K1_1, K2_1, Kp1, karstic_loss, FC1, SMs1, beta1, jarvis_thresh_c1, TT1, HL1_1, HL3, PW1)
Regionalizing and Upscaling process parameters.
- subroutine [mo_multi_param_reg::baseflow_param](#) (param, geoUnit0, geoUnitList, nodata, k2_0)
baseflow recession parameter
- subroutine [mo_multi_param_reg::snow_acc_melt_param](#) (param, c2TSTu, fForest1, flperm1, fPerm1, TT1, DD1, IDDP1, DDmax1)
Calculates the snow parameters.
- subroutine [mo_multi_param_reg::iper_thres_runoff](#) (param, HL3)
sets the impervious layer threshold parameter for runoff generation
- subroutine [mo_multi_param_reg::karstic_layer](#) (param, geoUnitKar, geoUnit0, geoUnitlist, mask0, nodata, SMs_FC0, KsVar_V0, cell_id0, nL0_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, c2TSTu, karstic_loss, L1_Kp)
calculates the Karstic percolation loss
- subroutine, public [mo_multi_param_reg::canopy_intercept_param](#) (proc_Mat, param, LAI0, nL0_in_L1, upp_row_L1, low_row_L1, lef_col_L1, rig_col_L1, cell_id0, mask0, nodata, max_intercept1)
estimate effective maximum interception capacity at L1
- subroutine [mo_multi_param_reg::aerodynamical_resistance](#) (LCover0, LAI_LUT, param, mask0, nodata, cell_id0, nL0_in_L1, Upp_row_L1, Low_row_L1, Lef_col_L1, Rig_col_L1, aerodyn_resistance1)
Regionalization of aerodynamic resistance.

17.56 mo_ncread.f90 File Reference

Data Types

- interface [mo_ncread::get_ncvar](#)

Modules

- module [mo_ncread](#)

Functions/Subroutines

- integer(i4) function, dimension(5), public [mo_ncread::get_ncdim](#) (Filename, Variable, PrintInfo, ndims)
- subroutine, public [mo_ncread::get_ncdimatt](#) (Filename, Variable, DimName, DimLen)
- subroutine, public [mo_ncread::get_ncvaratt](#) (FileName, VarName, AttName, AttValues, fid, dtype)
- subroutine [mo_ncread::get_ncvar_0d_sp](#) (Filename, VarName, Dat, fid)

- subroutine [mo_ncread::get_ncvar_0d_dp](#) (Filename, VarName, Dat, fid)
- subroutine [mo_ncread::get_ncvar_1d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_1d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_2d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_2d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_3d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_3d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_4d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_4d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_5d_sp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_5d_dp](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_0d_i4](#) (Filename, VarName, Dat, fid)
- subroutine [mo_ncread::get_ncvar_1d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_2d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_3d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_4d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_5d_i4](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_0d_i1](#) (Filename, VarName, Dat, fid)
- subroutine [mo_ncread::get_ncvar_1d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_2d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_3d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_4d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- subroutine [mo_ncread::get_ncvar_5d_i1](#) (Filename, VarName, Dat, start, a_count, fid)
- integer(i4) function, public [mo_ncread::ncopen](#) (Fname)
- subroutine, public [mo_ncread::ncclose](#) (ncid)
- subroutine [mo_ncread::get_info](#) (Varname, ncid, varid, xtype, dl, Info, ndims)
- subroutine [mo_ncread::check](#) (status)

17.57 mo_ncwrite.f90 File Reference

Data Types

- type [mo_ncwrite::dims](#)
- type [mo_ncwrite::attribute](#)
- type [mo_ncwrite::variable](#)
- interface [mo_ncwrite::dump_netcdf](#)
- interface [mo_ncwrite::var2nc](#)

Extended [dump_netcdf](#) for multiple variables.

Modules

- module [mo_ncwrite](#)

Functions/Subroutines

- subroutine, public [mo_ncwrite::close_netcdf](#) (ncid)
- subroutine, public [mo_ncwrite::create_netcdf](#) (Filename, ncid, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_1d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_2d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_3d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_4d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_5d_sp](#) (filename, arr, append, lfs, netcdf4, deflate_level)

- subroutine [mo_ncwrite::dump_netcdf_1d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_2d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_3d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_4d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_5d_dp](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_1d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_2d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_3d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_4d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::dump_netcdf_5d_i4](#) (filename, arr, append, lfs, netcdf4, deflate_level)
- subroutine [mo_ncwrite::var2nc_1d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_1d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_1d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_2d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_2d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_2d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_3d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_3d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_3d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_4d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_4d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_4d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_5d_i4](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_5d_sp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine [mo_ncwrite::var2nc_5d_dp](#) (f_name, arr, dnames, v_name, dim_unlimited, long_name, units, missing_value, attributes, create, ncid, nrec)
- subroutine, public [mo_ncwrite::write_dynamic_netcdf](#) (ncid, irec)
- subroutine, public [mo_ncwrite::write_static_netcdf](#) (ncid)
- integer(i4) function [mo_ncwrite::open_netcdf](#) (f_name, create)
- subroutine [mo_ncwrite::check](#) (status)

Variables

- integer(i4), parameter, public [mo_ncwrite::nmaxdim](#) = 5
- integer(i4), parameter, public [mo_ncwrite::nmaxatt](#) = 20
- integer(i4), parameter, public [mo_ncwrite::maxlen](#) = 256
- integer(i4), parameter, public [mo_ncwrite::ngatt](#) = 20
- integer(i4), parameter, public [mo_ncwrite::nattdim](#) = 2
- integer(i4), public [mo_ncwrite::nvars](#)
- integer(i4), public [mo_ncwrite::ndims](#)
- type(dims), dimension(:), allocatable, public [mo_ncwrite::dnc](#)
- type(variable), dimension(:), allocatable, public [mo_ncwrite::v](#)
- type(attribute), dimension(ngatt), public [mo_ncwrite::gatt](#)

17.58 mo_netcdf.f90 File Reference

Data Types

- interface [mo_netcdf::ncdataset](#)
Provides basic file modification functionality.
- interface [mo_netcdf::ncdataset](#)
Provides basic file modification functionality.
- type [mo_netcdf::ncdimension](#)
Provides the dimension access functionality.
- interface [mo_netcdf::ncvariable](#)

Modules

- module [mo_netcdf](#)
NetCDF Fortran 90 interface wrapper.

Functions/Subroutines

- subroutine [mo_netcdf::initncvariable](#) (self, id, parent)
- subroutine [mo_netcdf::initncdimension](#) (self, id, parent)
- subroutine [mo_netcdf::initncdataset](#) (self, fname, mode)
- type(ncvariable) function [mo_netcdf::newncvariable](#) (id, parent)
- type(ncdimension) function [mo_netcdf::newncdimension](#) (id, parent)
- type(ncdataset) function [mo_netcdf::newncdataset](#) (fname, mode)
- subroutine [mo_netcdf::close](#) (self)
- integer(i4) function [mo_netcdf::getnvariables](#) (self)
- integer(i4) function, dimension(:), allocatable [mo_netcdf::getvariableids](#) (self)
- type(ncvariable) function, dimension(:), allocatable [mo_netcdf::getvariables](#) (self)
- character(len=256) function [mo_netcdf::getdimensionname](#) (self)
- integer(i4) function [mo_netcdf::getdimensionlength](#) (self)
- logical function [mo_netcdf::isdatasetunlimited](#) (self)
- type(ncdimension) function [mo_netcdf::getunlimiteddimension](#) (self)
- logical function [mo_netcdf::equalncdimensions](#) (dim1, dim2)
- logical function [mo_netcdf::isunlimiteddimension](#) (self)
- type(ncdimension) function [mo_netcdf::setdimension](#) (self, name, length)
- logical function [mo_netcdf::hasvariable](#) (self, name)
- logical function [mo_netcdf::hasdimension](#) (self, name)
- type(ncvariable) function [mo_netcdf::setvariablewithids](#) (self, name, dtype, dimensions, contiguous, chunk-sizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(ncvariable) function [mo_netcdf::setvariablewithnames](#) (self, name, dtype, dimensions, contiguous, chunk-sizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(ncvariable) function [mo_netcdf::setvariablewithtypes](#) (self, name, dtype, dimensions, contiguous, chunk-sizes, deflate_level, shuffle, fletcher32, endianness, cache_size, cache_nelems, cache_preemption)
- type(ncdimension) function [mo_netcdf::getdimensionbyid](#) (self, id)
- type(ncdimension) function [mo_netcdf::getdimensionbyname](#) (self, name)
- type(ncvariable) function [mo_netcdf::getvariablebyname](#) (self, name)
- character(len=256) function [mo_netcdf::getvariablename](#) (self)
- integer(i4) function [mo_netcdf::getnodimensions](#) (self)
- type(ncdimension) function, dimension(:), allocatable [mo_netcdf::getvariabledimensions](#) (self)
- integer(i4) function, dimension(:), allocatable [mo_netcdf::getvariablesshape](#) (self)
- character(3) function [mo_netcdf::getvariabledtype](#) (self)
- logical function [mo_netcdf::isunlimitedvariable](#) (self)

- logical function `mo_netcdf::hasattribute` (self, name)
- subroutine `mo_netcdf::setglobalattributechar` (self, name, data)
- subroutine `mo_netcdf::setglobalattributei8` (self, name, data)
- subroutine `mo_netcdf::setglobalattributei16` (self, name, data)
- subroutine `mo_netcdf::setglobalattributei32` (self, name, data)
- subroutine `mo_netcdf::setglobalattributef32` (self, name, data)
- subroutine `mo_netcdf::setglobalattributef64` (self, name, data)
- subroutine `mo_netcdf::getglobalattributechar` (self, name, avalue)
- subroutine `mo_netcdf::getglobalattributei8` (self, name, avalue)
- subroutine `mo_netcdf::getglobalattributei16` (self, name, avalue)
- subroutine `mo_netcdf::getglobalattributei32` (self, name, avalue)
- subroutine `mo_netcdf::getglobalattributef32` (self, name, avalue)
- subroutine `mo_netcdf::getglobalattributef64` (self, name, avalue)
- subroutine `mo_netcdf::setvariableattributechar` (self, name, data)
- subroutine `mo_netcdf::setvariableattributei8` (self, name, data)
- subroutine `mo_netcdf::setvariableattributei16` (self, name, data)
- subroutine `mo_netcdf::setvariableattributei32` (self, name, data)
- subroutine `mo_netcdf::setvariableattributef32` (self, name, data)
- subroutine `mo_netcdf::setvariableattributef64` (self, name, data)
- subroutine `mo_netcdf::getvariableattributechar` (self, name, avalue)
- subroutine `mo_netcdf::getvariableattributei8` (self, name, avalue)
- subroutine `mo_netcdf::getvariableattributei16` (self, name, avalue)
- subroutine `mo_netcdf::getvariableattributei32` (self, name, avalue)
- subroutine `mo_netcdf::getvariableattributef32` (self, name, avalue)
- subroutine `mo_netcdf::getvariableattributef64` (self, name, avalue)
- subroutine `mo_netcdf::setvariablefillvaluei8` (self, fvalue)
- subroutine `mo_netcdf::setvariablefillvaluei16` (self, fvalue)
- subroutine `mo_netcdf::setvariablefillvaluei32` (self, fvalue)
- subroutine `mo_netcdf::setvariablefillvaluef32` (self, fvalue)
- subroutine `mo_netcdf::setvariablefillvaluef64` (self, fvalue)
- subroutine `mo_netcdf::getvariablefillvaluei8` (self, fvalue)
- subroutine `mo_netcdf::getvariablefillvaluei16` (self, fvalue)
- subroutine `mo_netcdf::getvariablefillvaluei32` (self, fvalue)
- subroutine `mo_netcdf::getvariablefillvaluef32` (self, fvalue)
- subroutine `mo_netcdf::getvariablefillvaluef64` (self, fvalue)
- subroutine `mo_netcdf::setdatascalar_i8` (self, values, start)
- subroutine `mo_netcdf::setdata1di8` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata2di8` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata3di8` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata4di8` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata5di8` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdatascalar_i16` (self, values, start)
- subroutine `mo_netcdf::setdata1di16` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata2di16` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata3di16` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata4di16` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata5di16` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdatascalar_i32` (self, values, start)
- subroutine `mo_netcdf::setdata1di32` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata2di32` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata3di32` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata4di32` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdata5di32` (self, values, start, cnt, stride, map)
- subroutine `mo_netcdf::setdatascalar_f32` (self, values, start)
- subroutine `mo_netcdf::setdata1df32` (self, values, start, cnt, stride, map)

- subroutine [mo_netcdf::setdata2df32](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata3df32](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata4df32](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata5df32](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdatascalarf64](#) (self, values, start)
- subroutine [mo_netcdf::setdata1df64](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata2df64](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata3df64](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata4df64](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::setdata5df64](#) (self, values, start, cnt, stride, map)
- subroutine [mo_netcdf::getdatascalarf8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata1df8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata2df8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata3df8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata4df8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata5df8](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdatascalarf16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata1df16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata2df16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata3df16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata4df16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata5df16](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdatascalarf32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata1df32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata2df32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata3df32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata4df32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata5df32](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdatascalarf64](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata1df64](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata2df64](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata3df64](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata4df64](#) (self, data, start, cnt, stride, map)
- subroutine [mo_netcdf::getdata5df64](#) (self, data, start, cnt, stride, map)
- integer(i4) function, dimension(datarank) [mo_netcdf::getreadddatashape](#) (var, datarank, instart, incnt, instride)
- integer(i4) function [mo_netcdf::getdtypefromstring](#) (dtype)
- character(3) function [mo_netcdf::getdtypefrominteger](#) (dtype)
- subroutine [mo_netcdf::check](#) (status, msg)

17.59 mo_neutrons.f90 File Reference

Models to predict neutron intensities above soils.

Modules

- module [mo_neutrons](#)

THIS MODULE IS WORK IN PROGRESS, DO NOT USE FOR RESEARCH.

Functions/Subroutines

- subroutine, public [mo_neutrons::desiletsn0](#) (SoilMoisture, Horizons, N0, neutrons)
Calculate neutrons from soil moisture in the first layer.
- subroutine, public [mo_neutrons::cosmic](#) (SoilMoisture, Horizons, params, neutron_integral_AFast, neutrons)
Calculate neutrons from soil moisture in all layers.
- subroutine [mo_neutrons::oldintegration](#) (res, c)
- subroutine, public [mo_neutrons::tabularintegralafast](#) (integral, maxC)
Save approximation data for A_fast.
- subroutine [mo_neutrons::approx_mon_int](#) (res, f, c, xmin, xmax, eps, steps, fxmin, fxmax)
- recursive subroutine [mo_neutrons::approx_mon_int_steps](#) (res, f, c, xmin, xmax, eps, steps, fxmin, fxmax)
- recursive subroutine [mo_neutrons::approx_mon_int_eps](#) (res, f, c, xmin, xmax, eps, fxmin, fxmax)
- subroutine [mo_neutrons::lookupintegral](#) (res, integral, c)
- real(dp) function [mo_neutrons::intgrandfast](#) (c, phi)

17.59.1 Detailed Description

Models to predict neutron intensities above soils.

17.60 mo_nml.f90 File Reference

Modules

- module [mo_nml](#)
Deal with namelist files.

Functions/Subroutines

- subroutine, public [mo_nml::open_nml](#) (file, unit, quiet)
Open a namelist file.
- subroutine, public [mo_nml::close_nml](#) (unit)
Close a namelist file.
- subroutine, public [mo_nml::position_nml](#) (name, unit, status, first)
Position a namlist file.

Variables

- integer(i4), parameter, public [mo_nml::positioned](#) = 0
Information: file pointer set to namelist group.
- integer(i4), parameter, public [mo_nml::missing](#) = 1
Error: namelist group is missing.
- integer(i4), parameter, public [mo_nml::length_error](#) = 2
Error: namelist group name too long.
- integer(i4), parameter, public [mo_nml::read_error](#) = 3
Error occured during read of namelist file.
- integer, save, public [mo_nml::nunitnml](#) = -1

17.61 mo_objective_function.f90 File Reference

Modules

- module [mo_objective_function](#)
Objective Functions for Optimization of mHM.

Functions/Subroutines

- real(dp) function, public [mo_objective_function::objective](#) (parameterset)
Wrapper for objective functions.
- real(dp) function [mo_objective_function::objective_sm_kge_catchment_avg](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [mo_objective_function::objective_sm_corr](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [mo_objective_function::objective_sm_pd](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [mo_objective_function::objective_sm_sse_standard_score](#) (parameterset)
Objective function for soil moisture.
- real(dp) function [mo_objective_function::objective_kge_q_rmse_tws](#) (parameterset)
Objective function of KGE for runoff and RMSE for basin_avg TWS (standarized scores)
- real(dp) function [mo_objective_function::objective_neutrons_kge_catchment_avg](#) (parameterset)
Objective function for neutrons.
- real(dp) function [mo_objective_function::objective_et_kge_catchment_avg](#) (parameterset)
Objective function for evapotranspiration (et).
- real(dp) function [mo_objective_function::objective_kge_q_sm_corr](#) (parameterset)
Objective function of KGE for runoff and correlation for SM.
- real(dp) function [mo_objective_function::objective_kge_q_et](#) (parameterset)
Objective function of KGE for runoff and KGE for ET.
- real(dp) function [mo_objective_function::objective_kge_q_rmse_et](#) (parameterset)
Objective function of KGE for runoff and RMSE for basin_avg ET (standarized scores)
- subroutine [mo_objective_function::extract_basin_avg_tws](#) (basinId, tws, tws_sim, tws_obs, tws_obs_mask)
extracts basin average tws data from global variables

17.62 mo_optimization.f90 File Reference

Modules

- module [mo_optimization](#)
Wrapper subroutine for optimization against runoff and sm.

Functions/Subroutines

- subroutine, public [mo_optimization::optimization](#) (objective, dirConfigOut, funcBest, maskpara)
Wrapper for optimization.

17.63 mo_orderpack.f90 File Reference

Data Types

- interface [mo_orderpack::sort](#)
Unconditional ranking.
- interface [mo_orderpack::sort_index](#)
- interface [mo_orderpack::ctrper](#)
- interface [mo_orderpack::fndnth](#)
- interface [mo_orderpack::indmed](#)
- interface [mo_orderpack::indnth](#)
- interface [mo_orderpack::inspar](#)
- interface [mo_orderpack::inssor](#)
- interface [mo_orderpack::omedian](#)
- interface [mo_orderpack::mrgref](#)
- interface [mo_orderpack::mrgnrk](#)
- interface [mo_orderpack::mulcnt](#)
- interface [mo_orderpack::rapknr](#)
- interface [mo_orderpack::refpar](#)
- interface [mo_orderpack::refsor](#)
- interface [mo_orderpack::rinpar](#)
- interface [mo_orderpack::rnkpar](#)
- interface [mo_orderpack::uniinv](#)
- interface [mo_orderpack::nearless](#)
- interface [mo_orderpack::unipar](#)
- interface [mo_orderpack::unirnk](#)
- interface [mo_orderpack::unista](#)
- interface [mo_orderpack::valmed](#)
- interface [mo_orderpack::valnth](#)

Modules

- module [mo_orderpack](#)
Sort and ranking routines.

Functions/Subroutines

- integer(i4) function, dimension(size(arr)) [mo_orderpack::sort_index_dp](#) (arr)
- integer(i4) function, dimension(size(arr)) [mo_orderpack::sort_index_sp](#) (arr)
- integer(i4) function, dimension(size(arr)) [mo_orderpack::sort_index_i4](#) (arr)
- subroutine, private [mo_orderpack::d_ctrper](#) (XDONT, PCLS)
- subroutine, private [mo_orderpack::r_ctrper](#) (XDONT, PCLS)
- subroutine, private [mo_orderpack::i_ctrper](#) (XDONT, PCLS)
- real(kind=dp) function, private [mo_orderpack::d_fndnth](#) (XDONT, NORD)
- real(kind=sp) function, private [mo_orderpack::r_fndnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [mo_orderpack::i_fndnth](#) (XDONT, NORD)
- subroutine, private [mo_orderpack::d_indmed](#) (XDONT, INDM)
- recursive subroutine, private [mo_orderpack::d_med](#) (XDATT, IDATT, ires_med)
- subroutine, private [mo_orderpack::r_indmed](#) (XDONT, INDM)
- recursive subroutine, private [mo_orderpack::r_med](#) (XDATT, IDATT, ires_med)
- subroutine, private [mo_orderpack::i_indmed](#) (XDONT, INDM)
- recursive subroutine, private [mo_orderpack::i_med](#) (XDATT, IDATT, ires_med)
- integer(kind=i4) function, private [mo_orderpack::d_indnth](#) (XDONT, NORD)

- integer(kind=i4) function, private [mo_orderpack::r_indnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [mo_orderpack::i_indnth](#) (XDONT, NORD)
- subroutine, private [mo_orderpack::d_inspar](#) (XDONT, NORD)
- subroutine, private [mo_orderpack::r_inspar](#) (XDONT, NORD)
- subroutine, private [mo_orderpack::i_inspar](#) (XDONT, NORD)
- subroutine, private [mo_orderpack::d_inssor](#) (XDONT)
- subroutine, private [mo_orderpack::r_inssor](#) (XDONT)
- subroutine, private [mo_orderpack::i_inssor](#) (XDONT)
- real(kind=dp) function, private [mo_orderpack::d_median](#) (XDONT)
- real(kind=sp) function, private [mo_orderpack::r_median](#) (XDONT)
- integer(kind=i4) function, private [mo_orderpack::i_median](#) (XDONT)
- subroutine, private [mo_orderpack::d_mrgref](#) (XVALT, IRNGT)
- subroutine, private [mo_orderpack::r_mrgref](#) (XVALT, IRNGT)
- subroutine, private [mo_orderpack::i_mrgref](#) (XVALT, IRNGT)
- subroutine, private [mo_orderpack::d_mrgrnk](#) (XDONT, IRNGT)
- subroutine, private [mo_orderpack::r_mrgrnk](#) (XDONT, IRNGT)
- subroutine, private [mo_orderpack::i_mrgrnk](#) (XDONT, IRNGT)
- subroutine, private [mo_orderpack::d_mulcnt](#) (XDONT, IMULT)
- subroutine, private [mo_orderpack::r_mulcnt](#) (XDONT, IMULT)
- subroutine, private [mo_orderpack::i_mulcnt](#) (XDONT, IMULT)
- subroutine, private [mo_orderpack::d_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::r_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::i_rapknr](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::d_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::r_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::i_refpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::d_refsor](#) (XDONT)
- recursive subroutine, private [mo_orderpack::d_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [mo_orderpack::r_refsor](#) (XDONT)
- recursive subroutine, private [mo_orderpack::r_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [mo_orderpack::i_refsor](#) (XDONT)
- recursive subroutine, private [mo_orderpack::i_subsor](#) (XDONT, IDEB1, IFIN1)
- subroutine, private [mo_orderpack::d_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::r_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::i_rinpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::d_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::r_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::i_rnkpar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::d_uniinv](#) (XDONT, IGOEST)
- subroutine, private [mo_orderpack::r_uniinv](#) (XDONT, IGOEST)
- subroutine, private [mo_orderpack::i_uniinv](#) (XDONT, IGOEST)
- real(kind=dp) function, private [mo_orderpack::d_nearless](#) (XVAL)
- real(kind=sp) function, private [mo_orderpack::r_nearless](#) (XVAL)
- integer(kind=i4) function, private [mo_orderpack::i_nearless](#) (XVAL)
- subroutine, private [mo_orderpack::d_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::r_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::i_unipar](#) (XDONT, IRNGT, NORD)
- subroutine, private [mo_orderpack::d_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [mo_orderpack::r_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [mo_orderpack::i_unirnk](#) (XVALT, IRNGT, NUNI)
- subroutine, private [mo_orderpack::d_unista](#) (XDONT, NUNI)
- subroutine, private [mo_orderpack::r_unista](#) (XDONT, NUNI)
- subroutine, private [mo_orderpack::i_unista](#) (XDONT, NUNI)
- recursive real(kind=dp) function, private [mo_orderpack::d_valmed](#) (XDONT)
- recursive real(kind=sp) function, private [mo_orderpack::r_valmed](#) (XDONT)

- recursive integer(kind=i4) function, private [mo_orderpack::i_valmed](#) (XDONT)
- real(kind=dp) function, private [mo_orderpack::d_valnth](#) (XDONT, NORD)
- real(kind=sp) function, private [mo_orderpack::r_valnth](#) (XDONT, NORD)
- integer(kind=i4) function, private [mo_orderpack::i_valnth](#) (XDONT, NORD)

Variables

- integer(kind=i4), dimension(:), allocatable, save [mo_orderpack::idont](#)

17.64 mo_percentile.f90 File Reference

Data Types

- interface [mo_percentile::median](#)
- interface [mo_percentile::n_element](#)
- interface [mo_percentile::percentile](#)
- interface [mo_percentile::qmedian](#)

Modules

- module [mo_percentile](#)

Functions/Subroutines

- real(dp) function [mo_percentile::median_dp](#) (arrin, mask)
- real(sp) function [mo_percentile::median_sp](#) (arrin, mask)
- real(dp) function [mo_percentile::n_element_dp](#) (idat, n, mask, before, after, previous, next)
- real(sp) function [mo_percentile::n_element_sp](#) (idat, n, mask, before, after, previous, next)
- real(dp) function [mo_percentile::percentile_0d_dp](#) (arrin, k, mask, mode_in)
- real(sp) function [mo_percentile::percentile_0d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function, dimension(size(k)) [mo_percentile::percentile_1d_dp](#) (arrin, k, mask, mode_in)
- real(sp) function, dimension(size(k)) [mo_percentile::percentile_1d_sp](#) (arrin, k, mask, mode_in)
- real(dp) function [mo_percentile::qmedian_dp](#) (dat)
- real(sp) function [mo_percentile::qmedian_sp](#) (dat)

17.65 mo_pet.f90 File Reference

Modules

- module [mo_pet](#)

Module for calculating reference/potential evapotranspiration [mm s⁻¹].

Functions/Subroutines

- elemental pure real(dp) function, public [mo_pet::pet_hargreaves](#) (HarSamCoeff, HarSamConst, tavg, tmax, tmin, latitude, doy)
Reference Evapotranspiration after Hargreaves.
- elemental pure real(dp) function, public [mo_pet::pet_priestly](#) (PriTayParam, Rn, tavg)
Reference Evapotranspiration after Priestly-Taylor.

- elemental pure real(dp) function, public [mo_pet::pet_penman](#) (net_rad, tavg, act_vap_pressure, aerodyn_resistance, bulksurface_resistance, a_s, a_sh)
Reference Evapotranspiration after Penman-Monteith.
- elemental pure real(dp) function, private [mo_pet::extraterr_rad_approx](#) (doy, latitude)
Approximation of extraterrestrial radiation.
- elemental pure real(dp) function, private [mo_pet::slope_satpressure](#) (tavg)
slope of saturation vapour pressure curve
- elemental pure real(dp) function, private [mo_pet::sat_vap_pressure](#) (tavg)
calculation of the saturation vapour pressure

17.66 mo_prepare_gridded_lai.f90 File Reference

Modules

- module [mo_prepare_gridded_lai](#)
Prepare daily LAI fields (e.g., MODIS data) for mHM.

Functions/Subroutines

- subroutine, public [mo_prepare_gridded_lai::prepare_gridded_daily_lai_data](#) (iBasin)
Prepare gridded daily LAI data.
- subroutine, public [mo_prepare_gridded_lai::prepare_gridded_mean_monthly_lai_data](#) (iBasin)
prepare_gridded_mean_monthly_LAI_data

17.67 mo_read_config.f90 File Reference

Modules

- module [mo_read_config](#)
Reading of main model configurations.

Functions/Subroutines

- subroutine, public [mo_read_config::read_config](#) ()
Read main configurations for mHM.
- logical function [mo_read_config::in_bound](#) (params)

17.68 mo_read_forcing_nc.f90 File Reference

Modules

- module [mo_read_forcing_nc](#)
Reads forcing input data.

Functions/Subroutines

- subroutine, public [mo_read_forcing_nc::read_forcing_nc](#) (folder, nRows, nCols, periode, varName, data, mask, lower, upper, nctimestep, nocheck, maskout)
Reads forcing input in NetCDF file format.
- subroutine, public [mo_read_forcing_nc::read_weights_nc](#) (folder, nRows, nCols, varName, data, mask, lower, upper, nocheck, maskout)
Reads weights for meteo forcings input in NetCDF file format.
- subroutine [mo_read_forcing_nc::get_time](#) (fName, vName, julStart, julEnd, nctimestep)

17.69 mo_read_latlon.f90 File Reference

Modules

- module [mo_read_latlon](#)
reading latitude and longitude coordinates for each basin

Functions/Subroutines

- subroutine, public [mo_read_latlon::read_latlon](#) (ii)
reads latitude and longitude coordinates

17.70 mo_read_lut.f90 File Reference

Modules

- module [mo_read_lut](#)
Routines reading lookup tables (lut).

Functions/Subroutines

- subroutine, public [mo_read_lut::read_geoformation_lut](#) (filename, fileunit, nGeo, geo_unit, geo_karstic)
Reads LUT containing geological formation information.
- subroutine, public [mo_read_lut::read_lai_lut](#) (filename, fileunit, nLAI, LAIIDlist, LAI)
Reads LUT containing LAI information.

17.71 mo_read_meteo.f90 File Reference

Modules

- module [mo_read_meteo](#)
Reads meteorological input data.

Functions/Subroutines

- subroutine, public [mo_read_meteo::read_meteo_bin](#) (folder, nRows, nCols, periode, data, mask, lower, upper)
Reads binary meteorological files.

17.72 mo_read_optional_data.f90 File Reference

Modules

- module [mo_read_optional_data](#)
Read optional data for mHM calibration.

Functions/Subroutines

- subroutine, public [mo_read_optional_data::read_soil_moisture](#) (iBasin)
Read soil moisture data from NetCDF file for calibration.
- subroutine, public [mo_read_optional_data::read_basin_avg_tws](#) ()
Read basin average TWS timeseries from file, the same way runoff is read.
- subroutine, public [mo_read_optional_data::read_neutrons](#) (iBasin)
Read neutrons data from NetCDF file for calibration.
- subroutine, public [mo_read_optional_data::read_evapotranspiration](#) (iBasin)
Read evapotranspiration data from NetCDF file for calibration.

17.73 mo_read_spatial_data.f90 File Reference

Data Types

- interface [mo_read_spatial_data::read_spatial_data_ascii](#)
Reads spatial data files of ASCII format.

Modules

- module [mo_read_spatial_data](#)
Reads spatial input data.

Functions/Subroutines

- subroutine [mo_read_spatial_data::read_spatial_data_ascii_dp](#) (filename, fileunit, header_ncols, header_↵
nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)
- subroutine [mo_read_spatial_data::read_spatial_data_ascii_i4](#) (filename, fileunit, header_ncols, header_↵
nrows, header_xllcorner, header_yllcorner, header_cellsize, data, mask)
- subroutine, public [mo_read_spatial_data::read_header_ascii](#) (filename, fileunit, header_ncols, header_↵
nrows, header_xllcorner, header_yllcorner, header_cellsize, header_nodata)
Reads header lines of ASCII files.

17.74 mo_read_timeseries.f90 File Reference

Modules

- module [mo_read_timeseries](#)
Routines to read files containing timeseries data.

Functions/Subroutines

- subroutine, public [mo_read_timeseries::read_timeseries](#) (filename, fileunit, periodStart, periodEnd, optimize, opti_function, data, mask, nMeasPerDay)

Reads time series in ASCII format.

17.75 mo_read_wrapper.f90 File Reference

Modules

- module [mo_read_wrapper](#)

Wrapper for all reading routines.

Functions/Subroutines

- subroutine, public [mo_read_wrapper::read_data](#)
Reads data.
- subroutine [mo_read_wrapper::check_consistency_lut_map](#) (data, lookuptable, filename, unique_values)
Checks if classes in input maps appear in look up tables.

17.76 mo_restart.f90 File Reference

Modules

- module [mo_restart](#)

reading and writing states, fluxes and configuration for restart of mHM.

Functions/Subroutines

- subroutine, public [mo_restart::write_restart_files](#) (OutPath)
write restart files for each basin
- subroutine, public [mo_restart::read_restart_config](#) (iBasin, soilId_isPresent, InPath)
reads configuration apart from Level 11 configuration from a restart directory
- subroutine, public [mo_restart::read_restart_states](#) (iBasin, InPath)
reads fluxes and state variables from file

17.77 mo_runoff.f90 File Reference

Modules

- module [mo_runoff](#)

Runoff generation for the unsaturated zone, saturated zone (or groundwater zone), and runoff accumulation.

Functions/Subroutines

- subroutine, public [mo_runoff::runoff_unsat_zone](#) (k1, kp, k0, alpha, karst_loss, pefec_soil, unsat_thresh, sat_storage, unsat_storage, slow_interflow, fast_interflow, perc)
Runoff generation for the saturated zone.
- subroutine, public [mo_runoff::runoff_sat_zone](#) (k2, sat_storage, baseflow)
Runoff generation for the saturated zone.
- subroutine, public [mo_runoff::l1_total_runoff](#) (fSealed_area_fraction, fast_interflow, slow_interflow, baseflow, direct_runoff, total_runoff)
total runoff accumulation at level 1

17.78 mo_sce.f90 File Reference

Modules

- module [mo_sce](#)
Shuffled Complex Evolution optimization algorithm.

Functions/Subroutines

- real(dp) function, dimension(size(pini, 1)), public [mo_sce::sce](#) (functn, pini, prange, mymaxn, mymaxit, myk-stop, mypcento, mypeps, myseed, myngs, mynpg, mynps, mynspl, mymings, myiniflg, myprint, mymask, myalpha, mybeta, tmp_file, popul_file, popul_file_append, parallel, restart, restart_file, bestf, neval, history)
Shuffled Complex Evolution (SCE) algorithm for global optimization.
- subroutine [write_best_intermediate](#) (to_file)
- subroutine [write_best_final](#) ()
- subroutine [write_population](#) (to_file)
- subroutine [write_termination_case](#) (case)
- subroutine [set_optional](#) ()
- subroutine [mo_sce::parstt](#) (x, bound, peps, mask, xnstd, gnrg, ipcng)
- subroutine [mo_sce::comp](#) (ngs2, npg, a, af, b, bf)
- subroutine [mo_sce::sort_matrix](#) (rb, ra)
- subroutine [mo_sce::chkcst](#) (x, bl, bu, mask, ibound)
- subroutine [mo_sce::getpnt](#) (idist, bl, bu, std, xi, mask, save_state, x)
- subroutine [mo_sce::cce](#) (s, sf, bl, bu, maskpara, xnstd, icall, maxn, maxit, save_state_gauss, functn, alpha, beta, history, idot)

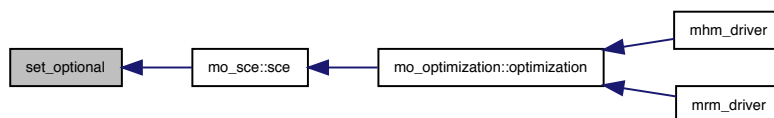
17.78.1 Function/Subroutine Documentation

17.78.1.1 set_optional()

```
subroutine sce::set_optional ( ) [private]
```

Referenced by [mo_sce::sce\(\)](#).

Here is the caller graph for this function:

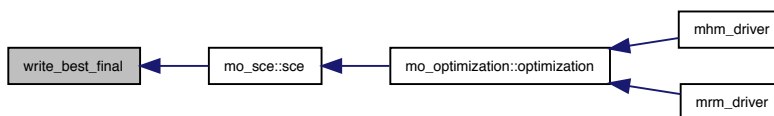


17.78.1.2 write_best_final()

```
subroutine sce::write_best_final ( ) [private]
```

Referenced by `mo_sce::sce()`.

Here is the caller graph for this function:

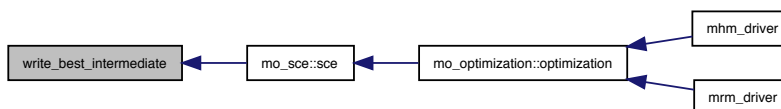


17.78.1.3 write_best_intermediate()

```
subroutine sce::write_best_intermediate (
    logical, intent(in) to_file )
```

Referenced by `mo_sce::sce()`.

Here is the caller graph for this function:

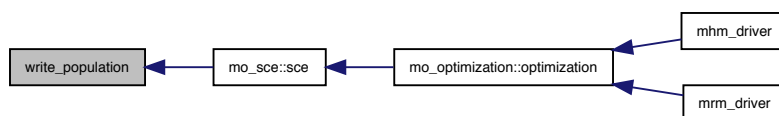


17.78.1.4 write_population()

```
subroutine sce::write_population (
    logical, intent(in) to_file ) [private]
```

Referenced by mo_sce::sce().

Here is the caller graph for this function:

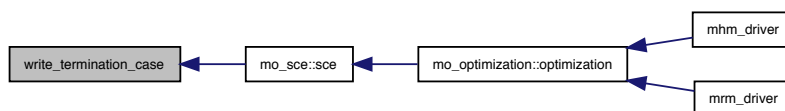


17.78.1.5 write_termination_case()

```
subroutine sce::write_termination_case (
    integer(i4), intent(in) case ) [private]
```

Referenced by mo_sce::sce().

Here is the caller graph for this function:



17.79 mo_set_netcdf_outputs.f90 File Reference

Modules

- module [mo_set_netcdf_outputs](#)
Defines the structure of the netCDF to write the output in.

Functions/Subroutines

- subroutine, public [mo_set_netcdf_outputs::set_netcdf](#) (NoNetcdfVars, nrows, ncols)
Runs mhm with a specific parameter set and returns required variables, e.g. runoff.

17.80 mo_snow_accum_melt.f90 File Reference

Modules

- module [mo_snow_accum_melt](#)
Snow melting and accumulation.

Functions/Subroutines

- subroutine, public [mo_snow_accum_melt::snow_accum_melt](#) (deg_day_incr, deg_day_max, deg_day_↔
noprec, prec, temperature, temperature_thresh, thrfall, snow_pack, deg_day, melt, prec_effect, rain, snow)
Snow melting and accumulation.

17.81 mo_soil_database.f90 File Reference

Modules

- module [mo_soil_database](#)
Generating soil database from input file.

Functions/Subroutines

- subroutine, public [mo_soil_database::read_soil_lut](#) (filename, iFlag_option, soilDB)
Reads the soil LUT file.
- subroutine, public [mo_soil_database::generate_soil_database](#) (soilDB, iFlag_option)
Generates soil database.

17.82 mo_soil_moisture.f90 File Reference

Modules

- module [mo_soil_moisture](#)
Soil moisture of the different layers.

Functions/Subroutines

- subroutine, public [mo_soil_moisture::soil_moisture](#) (processCase, frac_sealed, water_thresh_sealed, pet, evap_coeff, soil_moist_sat, frac_roots, soil_moist_FC, wilting_point, soil_moist_exponen, jarvis_thresh_c1, aet_canopy, prec_effec, runoff_sealed, storage_sealed, infiltration, soil_moist, aet, aet_sealed)
Soil moisture in different soil horizons.
- elemental pure real(dp) function, private [mo_soil_moisture::feddes_et_reduction](#) (soil_moist, soil_moist_FC, wilting_point, frac_roots)
stress factor for reducing evapotranspiration based on actual soil moisture
- elemental pure real(dp) function, private [mo_soil_moisture::jarvis_et_reduction](#) (soil_moist, soil_moist_sat, wilting_point, frac_roots, jarvis_thresh_c1)
stress factor for reducing evapotranspiration based on actual soil moisture

17.83 mo_spatial_agg_disagg_forcing.f90 File Reference

Data Types

- interface [mo_spatial_agg_disagg_forcing::spatial_aggregation](#)
Spatial aggregation of meteorological variables.
- interface [mo_spatial_agg_disagg_forcing::spatial_disaggregation](#)
Spatial disaggregation of meteorological variables.

Modules

- module [mo_spatial_agg_disagg_forcing](#)
Spatial aggregation or disaggregation of meteorological input data.

Functions/Subroutines

- subroutine [mo_spatial_agg_disagg_forcing::spatial_aggregation_3d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [mo_spatial_agg_disagg_forcing::spatial_aggregation_4d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [mo_spatial_agg_disagg_forcing::spatial_disaggregation_3d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)
- subroutine [mo_spatial_agg_disagg_forcing::spatial_disaggregation_4d](#) (data2, cellsize2, cellsize1, mask1, mask2, data1)

17.84 mo_spatialsimilarity.f90 File Reference

Data Types

- interface [mo_spatialsimilarity::nndv](#)
Calculates the number of neighboring dominating values, a measure for spatial dissimilarity.
- interface [mo_spatialsimilarity::pd](#)
Calculates pattern dissimilarity (PD) measure.

Modules

- module [mo_spatialsimilarity](#)
Routines for bias insensitive comparison of spatial patterns.

Functions/Subroutines

- real(sp) function [mo_spatialsimilarity::nndv_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [mo_spatialsimilarity::nndv_dp](#) (mat1, mat2, mask, valid)
- real(sp) function [mo_spatialsimilarity::pd_sp](#) (mat1, mat2, mask, valid)
- real(dp) function [mo_spatialsimilarity::pd_dp](#) (mat1, mat2, mask, valid)

17.85 mo_standard_score.f90 File Reference

Data Types

- interface [mo_standard_score::standard_score](#)
Calculates the standard score / normalization (anomaly) / z-score.
- interface [mo_standard_score::classified_standard_score](#)
Calculates the classified standard score (e.g. classes are months).

Modules

- module [mo_standard_score](#)
Routines for calculating the normalization (anomaly)/standard score/z score and the deseasonalized (standard score on monthly basis) values of a time series.

Functions/Subroutines

- real(sp) function, dimension(size(data, dim=1)) [mo_standard_score::standard_score_sp](#) (data, mask)
- real(dp) function, dimension(size(data, dim=1)) [mo_standard_score::standard_score_dp](#) (data, mask)
- real(sp) function, dimension(size(data, dim=1)) [mo_standard_score::classified_standard_score_sp](#) (data, classes, mask)
- real(dp) function, dimension(size(data, dim=1)) [mo_standard_score::classified_standard_score_dp](#) (data, classes, mask)

17.86 mo_startup.f90 File Reference

Modules

- module [mo_startup](#)
Startup procedures for mHM.

Functions/Subroutines

- subroutine, public [mo_startup::initialise](#) (iBasin)
Initialize main mHM variables.
- subroutine [mo_startup::constants_init](#) ()
Initialize mHM constants.
- subroutine [mo_startup::l0_check_input](#) (iBasin)
Check for errors in L0 input data.
- subroutine [mo_startup::l0_variable_init](#) (iBasin, soilId_isPresent)
level 0 variable initialization
- subroutine [mo_startup::l1_variable_init](#) (iBasin)
Level-1 variable initialization.
- subroutine [mo_startup::l2_variable_init](#) (iBasin)
Initialize Level-2 meteorological forcings data.

17.87 mo_string_utils.f90 File Reference

Data Types

- interface [mo_string_utils::num2str](#)
Convert to string.
- interface [mo_string_utils::numarray2str](#)
Convert to string.

Modules

- module [mo_string_utils](#)
String utilities.

Functions/Subroutines

- character(len(whitespaces)) function, public [mo_string_utils::compress](#) (whiteSpaces, n)
- subroutine, public [mo_string_utils::divide_string](#) (string, delim, strArr)
Divide string in substrings.
- logical function, public [mo_string_utils::equalstrings](#) (string1, string2)
- logical function, public [mo_string_utils::nonnull](#) (str)
Checks if string was already used.
- character(len=256) function, dimension(:), allocatable, public [mo_string_utils::splitstring](#) (string, delim)
- logical function, public [mo_string_utils::startswith](#) (string, start)
- character(len=len_trim(upper)) function, public [mo_string_utils::tolower](#) (upper)
Convert to lower case.
- character(len=len_trim(lower)) function, public [mo_string_utils::toupper](#) (lower)
- pure character(len=10) function [mo_string_utils::i42str](#) (nn, form)
- pure character(len=20) function [mo_string_utils::i82str](#) (nn, form)
- pure character(len=32) function [mo_string_utils::sp2str](#) (rr, form)
- pure character(len=32) function [mo_string_utils::dp2str](#) (rr, form)
- pure character(len=10) function [mo_string_utils::log2str](#) (ll, form)
- character(len=size(arr)) function [mo_string_utils::i4array2str](#) (arr)
- integer(i4) function, dimension(:), allocatable, public [mo_string_utils::str2num](#) (string)

Variables

- character(len= *), parameter, public [mo_string_utils::separator](#) = repeat('-', 70)

17.88 mo_template.f90 File Reference

Data Types

- interface [mo_template::mean](#)
The average.

Modules

- module [mo_template](#)
Template for future module developments.

Functions/Subroutines

- elemental pure real(dp) function, public [mo_template::circum](#) (radius)
Circumference of a circle.
- real(dp) function [mo_template::mean_dp](#) (dat, mask)
- real(sp) function [mo_template::mean_sp](#) (dat, mask)

Variables

- real(dp), parameter, public [mo_template::pi_dp](#) = 3.141592653589793238462643383279502884197_dp
Constant Pi in double precision.
- real(sp), parameter, public [mo_template::pi_sp](#) = 3.141592653589793238462643383279502884197_sp
Constant Pi in single precision.
- integer(i4), parameter [mo_template::itest](#) = 1

17.89 mo_temporal_aggregation.f90 File Reference

Data Types

- interface [mo_temporal_aggregation::day2mon_average](#)
Day-to-month average ([day2mon_average](#))
- interface [mo_temporal_aggregation::hour2day_average](#)
Hour-to-day average ([hour2day_average](#))

Modules

- module [mo_temporal_aggregation](#)
Temporal aggregation for time series (averaging)

Functions/Subroutines

- subroutine [mo_temporal_aggregation::day2mon_average_dp](#) (daily_data, yearS, monthS, dayS, mon_avg, misval, rm_misval)
- subroutine [mo_temporal_aggregation::hour2day_average_dp](#) (hourly_data, yearS, monthS, dayS, hourS, day_avg, misval, rm_misval)

17.90 mo_temporal_disagg_forcing.f90 File Reference

Modules

- module [mo_temporal_disagg_forcing](#)
Temporal disaggregation of daily input values.

Functions/Subroutines

- elemental pure subroutine, public [mo_temporal_disagg_forcing::temporal_disagg_forcing](#) (isday, ntimesteps, ←
_day, prec_day, pet_day, temp_day, fday_prec, fday_pet, fday_temp, fnight_prec, fnight_pet, fnight_temp,
temp_weights, pet_weights, pre_weights, read_meteo_weights, prec, pet, temp)
Temporally distribute daily mean forcings onto time step.

17.91 mo_timer.f90 File Reference

Modules

- module [mo_timer](#)
Timing routines.

Functions/Subroutines

- subroutine, public [mo_timer::timer_check](#) (timer)
Check a timer.
- subroutine, public [mo_timer::timer_clear](#) (timer)
Reset a timer.
- real(sp) function, public [mo_timer::timer_get](#) (timer)
Return a timer.
- subroutine, public [mo_timer::timer_print](#) (timer)
Print a timer.
- subroutine, public [mo_timer::timer_start](#) (timer)
Start a timer.
- subroutine, public [mo_timer::timer_stop](#) (timer)
Stop a timer.
- subroutine, public [mo_timer::timers_init](#)
Initialise timer module.

Variables

- integer(i4), parameter, public [mo_timer::max_timers](#) = 99
max number of timers allowed
- integer(i4), save, public [mo_timer::cycles_max](#)
max value of clock allowed by system
- real(sp), save, public [mo_timer::clock_rate](#)
clock_rate in seconds for each cycle
- integer(i4), dimension(max_timers), save, public [mo_timer::cycles1](#)
cycle number at start for each timer
- integer(i4), dimension(max_timers), save, public [mo_timer::cycles2](#)
cycle number at stop for each timer
- real(sp), dimension(max_timers), save, public [mo_timer::cputime](#)
accumulated cpu time in each timer
- character(len=8), dimension(max_timers), save, public [mo_timer::status](#)
timer status string

17.92 mo_upscaling_operators.f90 File Reference

Modules

- module [mo_upscaling_operators](#)
Module containing upscaling operators.

Functions/Subroutines

- integer(i4) function, dimension(size(l1_upper_rowId_cell, 1)), public [mo_upscaling_operators::majority_statistics](#) (nClass, L1_upper_rowId_cell, L1_lower_rowId_cell, L1_left_colonId_cell, L1_right_colonId_cell, L0_fineScale_2D_data)
majority statistics
- real(dp) function, dimension(size(l0upbound_inLx, 1)), public [mo_upscaling_operators::l0_fractionalcover_in_Lx](#) (dataIn0, classId, mask0, L0upBound_inLx, L0downBound_inLx, L0leftBound_inLx, L0rightBound_inLx, nTCells0_inLx)
fractional coverage of a given class of L0 fields in Lx field (Lx = L1 or L11)
- real(dp) function, dimension(size(nl0_cells_in_l1_cell, 1)), public [mo_upscaling_operators::upscale_arithmetic_mean](#) (nL0_cells_in_L1_cell, L1_upper_rowId_cell, L1_lower_rowId_cell, L1_left_colonId_cell, L1_right_colonId_cell, L0_cellId, mask0, nodata_value, L0_fineScale_data)
arithmetic mean
- real(dp) function, dimension(size(nl0_cells_in_l1_cell, 1)), public [mo_upscaling_operators::upscale_harmonic_mean](#) (nL0_cells_in_L1_cell, L1_upper_rowId_cell, L1_lower_rowId_cell, L1_left_colonId_cell, L1_right_colonId_cell, L0_cellId, mask0, nodata_value, L0_fineScale_data)
harmonic mean
- real(dp) function, dimension(size(l1_upper_rowId_cell, 1)), public [mo_upscaling_operators::upscale_geometric_mean](#) (L1_upper_rowId_cell, L1_lower_rowId_cell, L1_left_colonId_cell, L1_right_colonId_cell, mask0, nodata_value, L0_fineScale_data)
geometric mean

17.93 mo_utils.f90 File Reference

Data Types

- interface [mo_utils::equal](#)
Comparison of real values.
- interface [mo_utils::notequal](#)
- interface [mo_utils::greaterequal](#)
- interface [mo_utils::lesserequal](#)
- interface [mo_utils::eq](#)
- interface [mo_utils::ne](#)
- interface [mo_utils::ge](#)
- interface [mo_utils::le](#)
- interface [mo_utils::is_finite](#)
.true. if not IEEE Inf, IEEE NaN, nor IEEE Inf nor IEEE NaN, respectively.
- interface [mo_utils::is_nan](#)
- interface [mo_utils::is_normal](#)
- interface [mo_utils::locate](#)
Find closest values in a monotonic series, returns the indexes.
- interface [mo_utils::swap](#)
Swap to values or two elements in array.
- interface [mo_utils::special_value](#)
Special IEEE values.

Modules

- module [mo_utils](#)
General utilities for the CHS library.

Functions/Subroutines

- elemental pure logical function [mo_utils::equal_dp](#) (a, b)
- elemental pure logical function [mo_utils::equal_sp](#) (a, b)
- elemental pure logical function [mo_utils::greaterequal_dp](#) (a, b)
- elemental pure logical function [mo_utils::greaterequal_sp](#) (a, b)
- elemental pure logical function [mo_utils::lessequal_dp](#) (a, b)
- elemental pure logical function [mo_utils::lessequal_sp](#) (a, b)
- elemental pure logical function [mo_utils::notequal_dp](#) (a, b)
- elemental pure logical function [mo_utils::notequal_sp](#) (a, b)
- elemental pure logical function [mo_utils::is_finite_dp](#) (a)
- elemental pure logical function [mo_utils::is_finite_sp](#) (a)
- elemental pure logical function [mo_utils::is_nan_dp](#) (a)
- elemental pure logical function [mo_utils::is_nan_sp](#) (a)
- elemental pure logical function [mo_utils::is_normal_dp](#) (a)
- elemental pure logical function [mo_utils::is_normal_sp](#) (a)
- integer(i4) function [mo_utils::locate_0d_dp](#) (x, y)
- integer(i4) function [mo_utils::locate_0d_sp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [mo_utils::locate_1d_dp](#) (x, y)
- integer(i4) function, dimension(:), allocatable [mo_utils::locate_1d_sp](#) (x, y)
- elemental pure subroutine [mo_utils::swap_xy_dp](#) (x, y)
- elemental pure subroutine [mo_utils::swap_xy_sp](#) (x, y)
- elemental pure subroutine [mo_utils::swap_xy_i4](#) (x, y)
- subroutine [mo_utils::swap_vec_dp](#) (x, i1, i2)
- subroutine [mo_utils::swap_vec_sp](#) (x, i1, i2)
- subroutine [mo_utils::swap_vec_i4](#) (x, i1, i2)
- real(dp) function [mo_utils::special_value_dp](#) (x, ieee)
- real(sp) function [mo_utils::special_value_sp](#) (x, ieee)

17.94 mo_write_ascii.f90 File Reference

Modules

- module [mo_write_ascii](#)
Module to write ascii file output.

Functions/Subroutines

- subroutine, public [mo_write_ascii::write_configfile](#) ()
This modules writes the results of the configuration into an ASCII-file.
- subroutine, public [mo_write_ascii::write_optifile](#) (best_OF, best_paramSet, param_names)
Write briefly final optimization results.
- subroutine, public [mo_write_ascii::write_optinamelist](#) (processMatrix, parameters, maskpara, parameters_↵ name)
Write final, optimized parameter set in a namelist format.

17.95 mo_write_fluxes_states.f90 File Reference

Data Types

- interface [mo_write_fluxes_states::outputvariable](#)
- interface [mo_write_fluxes_states::outputvariable](#)
- interface [mo_write_fluxes_states::outputdataset](#)
- interface [mo_write_fluxes_states::outputdataset](#)

Modules

- module [mo_write_fluxes_states](#)

Creates NetCDF output for different fluxes and state variables of mHM.

Functions/Subroutines

- type(outputvariable) function [mo_write_fluxes_states::newoutputvariable](#) (nc, name, dtype, dims, ncells, mask, avg)
Initialize OutputVariable.
- subroutine [mo_write_fluxes_states::updatevariable](#) (self, data)
Update OutputVariable.
- subroutine [mo_write_fluxes_states::writevariabletimestep](#) (self, timestep)
Write timestep to file.
- type(outputdataset) function, public [mo_write_fluxes_states::newoutputdataset](#) (ibasin, mask1)
Initialize OutputDataset.
- subroutine [mo_write_fluxes_states::updatedataset](#) (self, sidx, eidx, L1_fSealed, L1_fNotSealed, L1_inter, L1_snowPack, L1_soilMoist, L1_soilMoistSat, L1_sealSTW, L1_unsatSTW, L1_satSTW, L1_neutrons, L1_pet, L1_aETSoil, L1_aETCanopy, L1_aETSealed, L1_total_runoff, L1_runoffSeal, L1_fastRunoff, L1_slowRunoff, L1_baseflow, L1_percol, L1_infilSoil, L1_preEffect)
Update all variables.
- subroutine [mo_write_fluxes_states::writetimestep](#) (self, timestep)
Write all accumulated data.
- subroutine [mo_write_fluxes_states::close](#) (self)
Close the file.
- type(ncdataset) function [mo_write_fluxes_states::createoutputfile](#) (ibasin)
Create and initialize output file.
- subroutine [mo_write_fluxes_states::writevariableattributes](#) (var, long_name, unit)
Write output variable attributes.
- subroutine [mo_write_fluxes_states::mapcoordinates](#) (ibasin, level, y, x)
Generate map coordinates.
- subroutine [mo_write_fluxes_states::geocoordinates](#) (ibasin, level, lat, lon)
Generate geographic coordinates.
- character(16) function [mo_write_fluxes_states::fluxesunit](#) (ibasin)
Generate a unit string.

17.96 mo_xor4096.f90 File Reference

Data Types

- interface [mo_xor4096::get_timeseed](#)
- interface [mo_xor4096::xor4096](#)
- interface [mo_xor4096::xor4096g](#)

Modules

- module [mo_xor4096](#)

Functions/Subroutines

- subroutine [mo_xor4096::get_timeseed_i4_0d](#) (seed)
- subroutine [mo_xor4096::get_timeseed_i4_1d](#) (seed)
- subroutine [mo_xor4096::get_timeseed_i8_0d](#) (seed)
- subroutine [mo_xor4096::get_timeseed_i8_1d](#) (seed)
- subroutine [mo_xor4096::xor4096s_0d](#) (seed, SingleIntegerRN, save_state)
- subroutine [mo_xor4096::xor4096s_1d](#) (seed, SingleIntegerRN, save_state)
- subroutine [mo_xor4096::xor4096f_0d](#) (seed, SingleRealRN, save_state)
- subroutine [mo_xor4096::xor4096f_1d](#) (seed, SingleRealRN, save_state)
- subroutine [mo_xor4096::xor4096l_0d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [mo_xor4096::xor4096l_1d](#) (seed, DoubleIntegerRN, save_state)
- subroutine [mo_xor4096::xor4096d_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [mo_xor4096::xor4096d_1d](#) (seed, DoubleRealRN, save_state)
- subroutine [mo_xor4096::xor4096gf_0d](#) (seed, SingleRealRN, save_state)
- subroutine [mo_xor4096::xor4096gf_1d](#) (seed, SingleRealRN, save_state)
- subroutine [mo_xor4096::xor4096gd_0d](#) (seed, DoubleRealRN, save_state)
- subroutine [mo_xor4096::xor4096gd_1d](#) (seed, DoubleRealRN, save_state)

Variables

- integer(i4), parameter, public [mo_xor4096::n_save_state](#) = 132_i4
Dimension of vector saving the state of a stream.

17.97 mrm_driver.f90 File Reference

Modules

- module [dummy](#)

Functions/Subroutines

- program [mrm_driver](#)

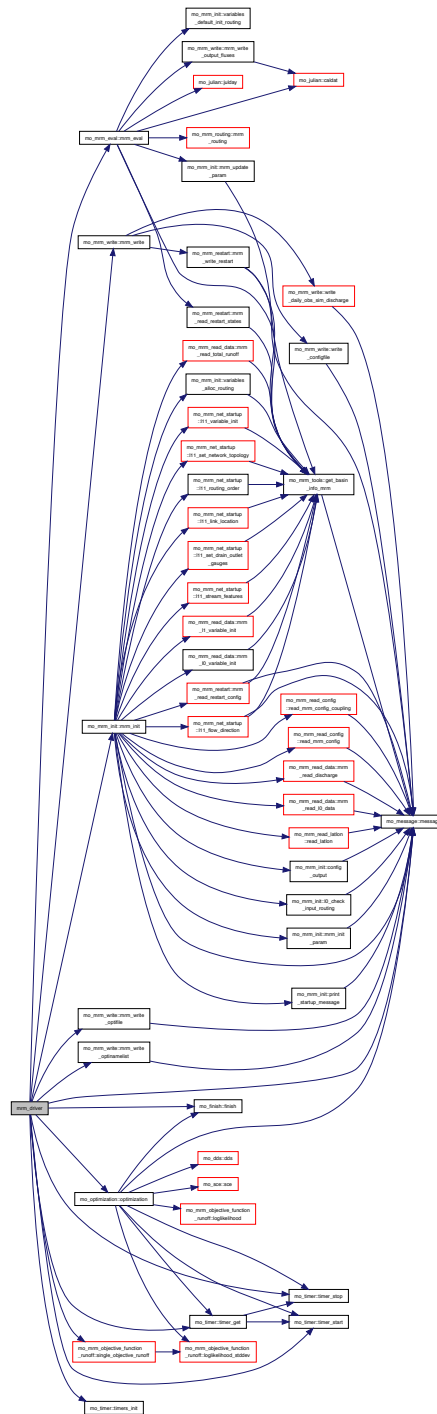
17.97.1 Function/Subroutine Documentation

17.97.1.1 mrm_driver()

```
program mrm_driver ( )
```

References [mo_mrm_global_variables::dirconfigout](#), [mo_kind::dp](#), [mo_finish::finish\(\)](#), [mo_common_variables::global_parameters](#), [mo_common_variables::global_parameters_name](#), [mo_kind::i4](#), [mo_message::message\(\)](#), [mo_mrm_eval::mrm_eval\(\)](#), [mo_mrm_init::mrm_init\(\)](#), [mo_mrm_write::mrm_write\(\)](#), [mo_mrm_write::mrm_write_optifile\(\)](#), [mo_mrm_write::mrm_write_optinamelist\(\)](#), [mo_optimization::optimization\(\)](#), [mo_common_variables::optimize](#), [mo_mrm_objective_function_runoff::single_objective_runoff\(\)](#), [mo_timer::timer_get\(\)](#), [mo_timer::timer_start\(\)](#), [mo_timer::timer_stop\(\)](#), and [mo_timer::timers_init\(\)](#).

17.98 RELEASES.md File Reference



Bibliography

- [1] E. Aarts and J. Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. Wiley, Chichester, 1990. [6](#)
- [2] S. Bergström. Development and application of a conceptual runoff model for scandinavian catchments. Technical Report 7, SMHI Reports RHO, Norrköping, 1976. [1](#)
- [3] K. Beven. Prophecy, reality and uncertainty in distributed hydrological modelling. *Adv. Water Resour.*, 16:41–51, 1993. [4](#)
- [4] G. Blöschl. Scaling in hydrology. *Hydrol. Process.*, 15(4):709–711, 2001. [2](#)
- [5] G. Blöschl, C. Reszler, and J. Komma. A spatially distributed flash flood forecasting model. *Environ. Model. Soft.*, 23(4):464–478, 2008. [1](#)
- [6] G. Blöschl and M. Sivapalan. Scale issues in hydrological modelling: A review. *Hydrol. Process.*, 9(3-4):251–290, 1995. [2](#)
- [7] V. T. Chow, D. R. Maidment, and L. W. Mays. *Applied Hydrology*. McGraw-Hill, 1988. [407](#)
- [8] L. Duckstein. Multiobjective optimization in structural design: The model choice problem. In E. Atrek, R. H. Gallagher, K. M. Ragsdell, and O. C. Zienkiewicz, editors, *New Directions in Optimum Structural Design*, pages 459–481. Eds. John Wiley and Sons Inc., New York, 1984. [6](#)
- [9] G. Hartmann and A. Bárdossy. Investigation of the transferability of hydrological models and a method to improve model calibration. *Adv. Geosciences*, 5:83–87, 2005. [6](#)
- [10] Y. Hundecha and A. Bárdossy. Modeling effect of land use changes on runoff generation of a river basin through parameter regionalization of a watershed model. *J. Hydrol.*, 292:281–295, 2004. [1](#)
- [11] X. Liang, D. P. Lettenmaier, E. F. Wood, and S. J. Burges. A simple hydrologically based model of land-surface water and energy fluxes for general-circulation models. *J. Geophys. Res.-Atmos.*, 99(D7):14415–14428, 1994. [1](#)
- [12] P. Pokhrel and H. V. Gupta. On the use of spatial-regularization strategies to improve calibration of distributed watershed models. *Water Resour. Res.*, 2009. In press. [4](#)
- [13] L. Samaniego and A. Bárdossy. Robust parametric models of runoff characteristics at the mesoscale. *Journal of Hydrology*, 303(1-4):136–151, 2005. doi: 10.1016/j.jhydrol.2004.08.022. [335](#)
- [14] L. Samaniego, R. Kumar, and S. Attinger. Multiscale parameter regionalization of a grid-based hydrologic model at the mesoscale. *Water Resour. Res.*, 46, 2010. [2](#), [5](#)
- [15] M. H. Tewelde and J. C. Smithers. Flood routing in ungauged catchments using muskingum methods. *Journal of Water South Africa*, 32(3):379–388, 2006. [407](#)
- [16] B. A. Tolson and C. A. Shoemaker. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resources Research*, 43(1):W01413, 2007. [6](#)

Index

1-main.dox, [977](#)
2-get_started.dox, [977](#)
3-data_preparation.dox, [977](#)
4-visualise_out.dox, [977](#)
5-calibration.dox, [977](#)
6-style_guide.dox, [977](#)
7-test_basin.dox, [977](#)
8-protocols_for_setup_new_mHM_basin.dox, [977](#)

absdev_dp
 mo_moment, [259](#)
 mo_moment::absdev, [779](#)
absdev_sp
 mo_moment, [259](#)
 mo_moment::absdev, [779](#)
add_inflow
 mo_mrm_routing, [405](#)
aerodynamical_resistance
 mo_multi_param_reg, [447](#)
all
 mo_mrm_write_fluxes_states::outputdataset, [904](#)
 mo_write_fluxes_states::outputdataset, [901](#)
alma_convention
 mo_common_variables, [89](#)
anneal_dp
 mo_anneal, [75](#)
 mo_anneal::anneal, [781](#)
append_char_3d
 mo_append, [80](#)
 mo_append::append, [783](#)
append_char_m_m
 mo_append, [80](#)
 mo_append::append, [783](#)
append_char_v_s
 mo_append, [81](#)
 mo_append::append, [783](#)
append_char_v_v
 mo_append, [81](#)
 mo_append::append, [783](#)
append_dp_3d
 mo_append, [81](#)
 mo_append::append, [783](#)
append_dp_m_m
 mo_append, [81](#)
 mo_append::append, [784](#)
append_dp_v_s
 mo_append, [81](#)
 mo_append::append, [784](#)
append_dp_v_v
 mo_append, [81](#)

 mo_append::append, [784](#)
append_i4_3d
 mo_append, [82](#)
append_i4_m_m
 mo_append, [82](#)
 mo_append::append, [784](#)
append_i4_v_s
 mo_append, [82](#)
 mo_append::append, [784](#)
append_i4_v_v
 mo_append, [82](#)
 mo_append::append, [784](#)
append_i8_3d
 mo_append, [82](#)
 mo_append::append, [785](#)
append_i8_m_m
 mo_append, [82](#)
 mo_append::append, [785](#)
append_i8_v_s
 mo_append, [82](#)
 mo_append::append, [785](#)
append_i8_v_v
 mo_append, [83](#)
 mo_append::append, [785](#)
append_lgt_3d
 mo_append, [83](#)
 mo_append::append, [785](#)
append_lgt_m_m
 mo_append, [83](#)
 mo_append::append, [785](#)
append_lgt_v_s
 mo_append, [83](#)
 mo_append::append, [785](#)
append_lgt_v_v
 mo_append, [83](#)
 mo_append::append, [786](#)
append_sp_3d
 mo_append, [83](#)
 mo_append::append, [786](#)
append_sp_m_m
 mo_append, [84](#)
 mo_append::append, [786](#)
append_sp_v_s
 mo_append, [84](#)
 mo_append::append, [786](#)
append_sp_v_v
 mo_append, [84](#)
 mo_append::append, [786](#)
approx_mon_int

- mo_neutrons, 566
- approx_mon_int_eps
 - mo_neutrons, 567
- approx_mon_int_steps
 - mo_neutrons, 568
- arth_dp
 - mo_corr, 103
 - mo_corr::arth, 787
- arth_i4
 - mo_corr, 103
 - mo_corr::arth, 787
- arth_sp
 - mo_corr, 103
 - mo_corr::arth, 787
- att
 - mo_ncwrite::variable, 968
- autocoeffk_1d_dp
 - mo_corr, 103
 - mo_corr::autocoeffk, 789
- autocoeffk_1d_sp
 - mo_corr, 103
 - mo_corr::autocoeffk, 789
- autocoeffk_dp
 - mo_corr, 104
 - mo_corr::autocoeffk, 789
- autocoeffk_sp
 - mo_corr, 104
 - mo_corr::autocoeffk, 789
- autocorr_1d_dp
 - mo_corr, 104
 - mo_corr::autocorr, 790
- autocorr_1d_sp
 - mo_corr, 104
 - mo_corr::autocorr, 790
- autocorr_dp
 - mo_corr, 104
 - mo_corr::autocorr, 790
- autocorr_sp
 - mo_corr, 104
 - mo_corr::autocorr, 790
- autocorrelation
 - mo_mrm_signatures, 414
- average
 - mo_mrm_write_fluxes_states::outputvariable, 913
 - mo_write_fluxes_states::outputvariable, 908
- average_counter
 - mo_mrm_write, 435
- average_dp
 - mo_moment, 260
 - mo_moment::average, 791
- average_sp
 - mo_moment, 260
 - mo_moment::average, 791
- avg
 - mo_mrm_write_fluxes_states::outputvariable, 913
 - mo_write_fluxes_states::outputvariable, 908
- baseflow_param
 - mo_multi_param_reg, 449
- basin
 - mo_global_variables, 151
 - mo_mrm_write_fluxes_states::outputdataset, 904
 - mo_write_fluxes_states::outputdataset, 901
- basin_avg_tws_obs
 - mo_global_variables, 151
- basin_avg_tws_sim
 - mo_global_variables, 151
- basin_mrm
 - mo_mrm_global_variables, 303
- basinid
 - mo_global_variables::twsstructure, 955
 - mo_mrm_global_variables::gaugingstation, 815
- before
 - mo_mrm_write_fluxes_states::outputvariable, 913
 - mo_write_fluxes_states::outputvariable, 908
- between
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputvariable, 909
- bias_dp_1d
 - mo_errormeasures, 114
 - mo_errormeasures::bias, 798
- bias_dp_2d
 - mo_errormeasures, 114
 - mo_errormeasures::bias, 798
- bias_dp_3d
 - mo_errormeasures, 114
 - mo_errormeasures::bias, 799
- bias_sp_1d
 - mo_errormeasures, 114
 - mo_errormeasures::bias, 799
- bias_sp_2d
 - mo_errormeasures, 115
 - mo_errormeasures::bias, 799
- bias_sp_3d
 - mo_errormeasures, 115
 - mo_errormeasures::bias, 799
- bulkdens_ormatter
 - mo_mhm_constants, 245
- bulksurface_resistance
 - mo_mpr_pet, 265
- c1_initstatesm
 - mo_mhm_constants, 245
- c2tstu
 - mo_global_variables, 151
- calculate_grid_properties
 - mo_init_states, 177
 - mo_mrm_tools, 423
- caldat
 - mo_julian, 185
- caldat360
 - mo_julian, 187
- caldat365
 - mo_julian, 188
- caldatjulian
 - mo_julian, 190
- calendar
 - mo_julian, 217

- calls
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputvariable, 909
- canopy_interc
 - mo_canopy_interc, 87
- canopy_intercept_param
 - mo_multi_param_reg, 450
- cce
 - mo_sce, 687
- celllength
 - mo_mrm_net_startup, 332
- cellsize
 - mo_global_variables::gridgeoref, 826
 - mo_mrm_global_variables::gridgeoref, 828
- central_moment_dp
 - mo_moment, 260
 - mo_moment::central_moment, 800
- central_moment_sp
 - mo_moment, 260
 - mo_moment::central_moment, 800
- central_moment_var_dp
 - mo_moment, 260
 - mo_moment::central_moment_var, 800
- central_moment_var_sp
 - mo_moment, 260
 - mo_moment::central_moment_var, 800
- check
 - mo_ncread, 464
 - mo_ncwrite, 484
 - mo_netcdf, 513
- check_consistency_lut_map
 - mo_read_wrapper, 673
- chkcst
 - mo_sce, 688
- chunk_config
 - mo_meteo_forcings, 227
- chunk_size
 - mo_meteo_forcings, 228
- circum
 - mo_template, 728
- classified_standard_score_dp
 - mo_standard_score, 712
 - mo_standard_score::classified_standard_score, 801
- classified_standard_score_sp
 - mo_standard_score, 712
 - mo_standard_score::classified_standard_score, 802
- clay
 - mo_global_variables::soiltype, 937
- clock_rate
 - mo_timer, 742
- close
 - mo_mrm_write_fluxes_states, 437
 - mo_mrm_write_fluxes_states::outputdataset, 904
 - mo_netcdf, 513
 - mo_netcdf::ncdataset, 864
 - mo_write_fluxes_states, 761
 - mo_write_fluxes_states::outputdataset, 900
- close_netcdf
 - mo_ncwrite, 485
- close_nml
 - mo_nml, 575
- comp
 - mo_sce, 688
- compress
 - mo_string_utils, 721
- config_output
 - mo_mrm_init, 320
- constants_init
 - mo_startup, 713
- contact
 - mo_global_variables, 151
 - mo_mrm_global_variables, 303
- contains
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputvariable, 909
- conventions
 - mo_global_variables, 152
 - mo_mrm_global_variables, 303
- corr_dp
 - mo_corr, 104
 - mo_corr::corr, 802
- corr_sp
 - mo_corr, 105
 - mo_corr::corr, 802
- correlation_dp
 - mo_moment, 261
 - mo_moment::correlation, 803
- correlation_sp
 - mo_moment, 261
 - mo_moment::correlation, 803
- cosmic
 - mo_neutrons, 568
- cosmic_alpha
 - mo_mhm_constants, 245
- cosmic_bd
 - mo_mhm_constants, 246
- cosmic_l1
 - mo_mhm_constants, 246
- cosmic_l2
 - mo_mhm_constants, 246
- cosmic_l3
 - mo_mhm_constants, 246
- cosmic_l4
 - mo_mhm_constants, 246
- cosmic_n
 - mo_mhm_constants, 246
- cosmic_vwclat
 - mo_mhm_constants, 246
- count
 - mo_mrm_write_fluxes_states::outputdataset, 904
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_ncwrite::variable, 968
 - mo_write_fluxes_states::outputdataset, 901
 - mo_write_fluxes_states::outputvariable, 909

- counter
 - mo_mrm_write_fluxes_states::outputdataset, 905
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputdataset, 901
 - mo_write_fluxes_states::outputvariable, 909
- covariance_dp
 - mo_moment, 261
 - mo_moment::covariance, 803
- covariance_sp
 - mo_moment, 261
 - mo_moment::covariance, 803
- cp0_dp
 - mo_constants, 94
- cp0_sp
 - mo_constants, 94
- cputime
 - mo_timer, 742
- create_netcdf
 - mo_ncwrite, 487
- created
 - mo_mrm_write_fluxes_states::outputdataset, 905
 - mo_write_fluxes_states::outputdataset, 901
- createoutputfile
 - mo_mrm_write_fluxes_states, 438
 - mo_write_fluxes_states, 762
- crosscoeffk_dp
 - mo_corr, 105
 - mo_corr::crosscoeffk, 804
- crosscoeffk_sp
 - mo_corr, 105
 - mo_corr::crosscoeffk, 804
- crosscorr_dp
 - mo_corr, 105
 - mo_corr::crosscorr, 804
- crosscorr_sp
 - mo_corr, 105
 - mo_corr::crosscorr, 805
- cycles1
 - mo_timer, 742
- cycles2
 - mo_timer, 743
- cycles_max
 - mo_timer, 743
- d_ctrper
 - mo_orderpack, 615
 - mo_orderpack::ctrper, 805
- d_fndnth
 - mo_orderpack, 615
 - mo_orderpack::fndnth, 813
- d_indmed
 - mo_orderpack, 615
 - mo_orderpack::indmed, 830
- d_indnth
 - mo_orderpack, 616
 - mo_orderpack::indnth, 830
- d_inspar
 - mo_orderpack, 616
 - mo_orderpack::inspar, 831
- d_inssor
 - mo_orderpack, 616
 - mo_orderpack::inssor, 832
- d_med
 - mo_orderpack, 616
- d_median
 - mo_orderpack, 617
 - mo_orderpack::omedian, 898
- d_mrgref
 - mo_orderpack, 617
 - mo_orderpack::mrgref, 858
- d_mrgrnk
 - mo_orderpack, 617
 - mo_orderpack::mrgrnk, 859
- d_mulcnt
 - mo_orderpack, 617
 - mo_orderpack::mulcnt, 861
- d_nearless
 - mo_orderpack, 617
 - mo_orderpack::nearless, 891
- d_rapknr
 - mo_orderpack, 617
 - mo_orderpack::rapknr, 925
- d_refpar
 - mo_orderpack, 618
 - mo_orderpack::refpar, 928
- d_refsor
 - mo_orderpack, 618
 - mo_orderpack::refsor, 929
 - mo_orderpack::sort, 941
- d_rinpar
 - mo_orderpack, 618
 - mo_orderpack::rinpar, 930
- d_rnkpar
 - mo_orderpack, 618
 - mo_orderpack::rnkpar, 932
- d_subsor
 - mo_orderpack, 618
- d_uniinv
 - mo_orderpack, 619
 - mo_orderpack::uniinv, 956
- d_unipar
 - mo_orderpack, 619
 - mo_orderpack::unipar, 956
- d_unirnk
 - mo_orderpack, 619
 - mo_orderpack::unirnk, 957
- d_unista
 - mo_orderpack, 619
 - mo_orderpack::unista, 958
- d_valmed
 - mo_orderpack, 619
 - mo_orderpack::valmed, 959
- d_valnth
 - mo_orderpack, 620
 - mo_orderpack::valnth, 959
- DEPENDENCIES.md, 977
- data

- mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputvariable, 909
- dataset
 - mo_netcdf::ncdataset, 872
- date2dec
 - mo_julian, 192
- date2dec360
 - mo_julian, 194
- date2dec365
 - mo_julian, 195
- date2decjulian
 - mo_julian, 197
- day2mon_average_dp
 - mo_temporal_aggregation, 730
 - mo_temporal_aggregation::day2mon_average, 806
- day_counter
 - mo_mrm_write, 436
- dayhours
 - mo_mhm_constants, 247
- daysecs
 - mo_mhm_constants, 247
- db
 - mo_global_variables::soiltype, 937
- dbm
 - mo_global_variables::soiltype, 937
- dchange_dp
 - mo_anneal, 76
- dds
 - mo_dds, 109
- dds_r
 - mo_common_variables, 89
- dec2date
 - mo_julian, 199
- dec2date360
 - mo_julian, 201
- dec2date365
 - mo_julian, 203
- dec2datejulian
 - mo_julian, 205
- deg2rad_dp
 - mo_constants, 95
- deg2rad_sp
 - mo_constants, 95
- deltah
 - mo_mrm_constants, 290
- dend
 - mo_common_variables::period, 924
- depth
 - mo_global_variables::soiltype, 937
- desilets_a0
 - mo_mhm_constants, 247
- desilets_a1
 - mo_mhm_constants, 247
- desilets_a2
 - mo_mhm_constants, 247
- desiletsn0
 - mo_neutrons, 570
- dimension
 - mo_netcdf::ncdimension, 889
- dimid
 - mo_ncwrite::dims, 807
- dimids
 - mo_ncwrite::variable, 968
- dimtypes
 - mo_ncwrite::variable, 968
- dirabsvappressure
 - mo_global_variables, 152
- dircommonfiles
 - mo_global_variables, 152
 - mo_mrm_global_variables, 303
- dirconfigout
 - mo_global_variables, 152
 - mo_mrm_global_variables, 304
- direvapotranspiration
 - mo_global_variables, 152
- dirgauges
 - mo_mrm_global_variables, 304
- dirgridded_lai
 - mo_global_variables, 152
- dirlcover
 - mo_global_variables, 152
 - mo_mrm_global_variables, 304
- dirmaxtemperature
 - mo_global_variables, 152
- dirmintemperature
 - mo_global_variables, 153
- dirmorpho
 - mo_global_variables, 153
 - mo_mrm_global_variables, 304
- dirnetradiation
 - mo_global_variables, 153
- dirneutrons
 - mo_global_variables, 153
- dirout
 - mo_global_variables, 153
 - mo_mrm_global_variables, 304
- dirprecipitation
 - mo_global_variables, 153
- dirreferenceet
 - mo_global_variables, 153
- dirrestartin
 - mo_global_variables, 153
 - mo_mrm_global_variables, 304
- dirrestartout
 - mo_global_variables, 154
 - mo_mrm_global_variables, 304
- dirsoil_moisture
 - mo_global_variables, 154
- dirtemperature
 - mo_global_variables, 154
- dirtotalrunoff
 - mo_mrm_global_variables, 305
- dirwindspeed
 - mo_global_variables, 154
- divide_string

- mo_string_utils, 722
- dnc
 - mo_ncwrite, 508
- dp
 - mo_kind, 218
- dp2str
 - mo_string_utils, 723
 - mo_string_utils::num2str, 896
- dpc
 - mo_kind, 218
- dstart
 - mo_common_variables::period, 924
- duffiedelta1
 - mo_mhm_constants, 247
- duffiedelta2
 - mo_mhm_constants, 247
- duffiedr
 - mo_mhm_constants, 247
- dummy, 75
- dump_netcdf_1d_dp
 - mo_ncwrite, 488
 - mo_ncwrite::dump_netcdf, 808
- dump_netcdf_1d_i4
 - mo_ncwrite, 488
 - mo_ncwrite::dump_netcdf, 808
- dump_netcdf_1d_sp
 - mo_ncwrite, 489
 - mo_ncwrite::dump_netcdf, 808
- dump_netcdf_2d_dp
 - mo_ncwrite, 489
 - mo_ncwrite::dump_netcdf, 809
- dump_netcdf_2d_i4
 - mo_ncwrite, 490
 - mo_ncwrite::dump_netcdf, 809
- dump_netcdf_2d_sp
 - mo_ncwrite, 490
 - mo_ncwrite::dump_netcdf, 809
- dump_netcdf_3d_dp
 - mo_ncwrite, 491
 - mo_ncwrite::dump_netcdf, 809
- dump_netcdf_3d_i4
 - mo_ncwrite, 491
 - mo_ncwrite::dump_netcdf, 810
- dump_netcdf_3d_sp
 - mo_ncwrite, 492
 - mo_ncwrite::dump_netcdf, 810
- dump_netcdf_4d_dp
 - mo_ncwrite, 492
 - mo_ncwrite::dump_netcdf, 810
- dump_netcdf_4d_i4
 - mo_ncwrite, 493
 - mo_ncwrite::dump_netcdf, 810
- dump_netcdf_4d_sp
 - mo_ncwrite, 493
 - mo_ncwrite::dump_netcdf, 810
- dump_netcdf_5d_dp
 - mo_ncwrite, 494
 - mo_ncwrite::dump_netcdf, 811
- dump_netcdf_5d_i4
 - mo_ncwrite, 494
 - mo_ncwrite::dump_netcdf, 811
- dump_netcdf_5d_sp
 - mo_ncwrite, 495
 - mo_ncwrite::dump_netcdf, 811
- eps_dp
 - mo_constants, 95
- eps_sp
 - mo_constants, 95
- equal_dp
 - mo_utils, 753
 - mo_utils::eq, 811
 - mo_utils::equal, 813
- equal_sp
 - mo_utils, 753
 - mo_utils::eq, 812
 - mo_utils::equal, 813
- equalncdimensions
 - mo_netcdf, 513
- equalstrings
 - mo_string_utils, 723
- euler
 - mo_constants, 95
- euler_d
 - mo_constants, 95
- eval
 - mo_mrm_objective_function_runoff, 346
- evalper
 - mo_global_variables, 154
 - mo_mrm_global_variables, 305
- evap_coeff
 - mo_global_variables, 154
- extract_basin_avg_tws
 - mo_objective_function, 580
- extract_runoff
 - mo_mrm_objective_function_runoff, 348
- extraterr_rad_approx
 - mo_pet, 633
- fday_pet
 - mo_global_variables, 154
- fday_prec
 - mo_global_variables, 154
- fday_temp
 - mo_global_variables, 155
- feddes_et_reduction
 - mo_soil_moisture, 702
- field_cap
 - mo_mpr_soilmoist, 280
- field_cap_c1
 - mo_mhm_constants, 248
- field_cap_c2
 - mo_mhm_constants, 248
- file
 - mo_netcdf::ncdataset, 872
- file_aspect
 - mo_file, 138

- file_config
 - mo_file, [138](#)
 - mo_mrm_file, [296](#)
- file_daily_discharge
 - mo_file, [138](#)
 - mo_mrm_file, [296](#)
- file_defoutput
 - mo_file, [139](#)
 - mo_mrm_file, [296](#)
- file_dem
 - mo_file, [139](#)
 - mo_mrm_file, [296](#)
- file_facc
 - mo_file, [139](#)
 - mo_mrm_file, [296](#)
- file_fdir
 - mo_file, [139](#)
 - mo_mrm_file, [297](#)
- file_gaugeloc
 - mo_mrm_file, [297](#)
- file_geolut
 - mo_file, [139](#)
- file_hydrogeoclass
 - mo_file, [139](#)
- file_lai_binary_end
 - mo_file, [139](#)
- file_lai_header
 - mo_file, [140](#)
- file_laiclass
 - mo_file, [140](#)
- file_lailut
 - mo_file, [140](#)
- file_main
 - mo_file, [140](#)
 - mo_mrm_file, [297](#)
- file_meteo_binary_end
 - mo_file, [140](#)
- file_meteo_header
 - mo_file, [140](#)
- file_mrm_output
 - mo_mrm_file, [297](#)
- file_namelist
 - mo_file, [140](#)
- file_namelist_mrm
 - mo_mrm_file, [297](#)
- file_namelist_param
 - mo_file, [141](#)
- file_namelist_param_mrm
 - mo_mrm_file, [297](#)
- file_opti
 - mo_file, [141](#)
 - mo_mrm_file, [297](#)
- file_opti_nml
 - mo_file, [141](#)
 - mo_mrm_file, [298](#)
- file_slope
 - mo_file, [141](#)
- file_soil_database
 - mo_file, [141](#)
- file_soil_database_1
 - mo_file, [141](#)
- file_soilclass
 - mo_file, [141](#)
- filelatlon
 - mo_global_variables, [155](#)
 - mo_mrm_global_variables, [305](#)
- filename
 - mo_netcdf::ncdataset, [872](#)
- filetwc
 - mo_global_variables, [155](#)
- finish
 - mo_finish, [146](#)
- flowdurationcurve
 - mo_mrm_signatures, [415](#)
- fluxesunit
 - mo_mrm_write_fluxes_states, [439](#)
 - mo_write_fluxes_states, [763](#)
- fname
 - mo_global_variables::twssstructure, [955](#)
 - mo_mrm_global_variables::gaugingstation, [816](#)
 - mo_netcdf::ncdataset, [872](#)
- fnight_pet
 - mo_global_variables, [155](#)
- fnight_prec
 - mo_global_variables, [155](#)
- fnight_temp
 - mo_global_variables, [155](#)
- four1_dp
 - mo_corr, [106](#)
 - mo_corr::four1, [814](#)
- four1_sp
 - mo_corr, [106](#)
 - mo_corr::four1, [814](#)
- fourrow_dp
 - mo_corr, [106](#)
 - mo_corr::fourrow, [814](#)
- fourrow_sp
 - mo_corr, [106](#)
 - mo_corr::fourrow, [815](#)
- fracsealed_cityarea
 - mo_global_variables, [155](#)
 - mo_mrm_global_variables, [305](#)
- g0_b
 - mo_ncwrite::variable, [969](#)
- g0_d
 - mo_ncwrite::variable, [969](#)
- g0_f
 - mo_ncwrite::variable, [969](#)
- g0_i
 - mo_ncwrite::variable, [969](#)
- g1_b
 - mo_ncwrite::variable, [969](#)
- g1_d
 - mo_ncwrite::variable, [969](#)
- g1_f
 - mo_ncwrite::variable, [969](#)

- g1_i
 - mo_ncwrite::variable, 969
- g2_b
 - mo_ncwrite::variable, 969
- g2_d
 - mo_ncwrite::variable, 970
- g2_f
 - mo_ncwrite::variable, 970
- g2_i
 - mo_ncwrite::variable, 970
- g3_b
 - mo_ncwrite::variable, 970
- g3_d
 - mo_ncwrite::variable, 970
- g3_f
 - mo_ncwrite::variable, 970
- g3_i
 - mo_ncwrite::variable, 970
- g4_b
 - mo_ncwrite::variable, 970
- g4_d
 - mo_ncwrite::variable, 970
- g4_f
 - mo_ncwrite::variable, 971
- g4_i
 - mo_ncwrite::variable, 971
- gatt
 - mo_ncwrite, 508
- gauge
 - mo_mrm_global_variables, 305
- gaugeid
 - mo_mrm_global_variables::gaugingstation, 816
- gaugeidlist
 - mo_mrm_global_variables::basininfo, 795
- gaugeindexlist
 - mo_mrm_global_variables::basininfo, 795
- gaugenodelist
 - mo_mrm_global_variables::basininfo, 795
- generate_neighborhood_weight_dp
 - mo_anneal, 77
 - mo_anneal::generate_neighborhood_weight, 817
- generate_soil_database
 - mo_soil_database, 699
- generatenewparameterset_dp
 - mo_mcmc, 221
- genuchten
 - mo_mpr_soilmoist, 281
- geocoordinates
 - mo_mrm_write_fluxes_states, 440
 - mo_write_fluxes_states, 765
- geounitkar
 - mo_global_variables, 155
- geounitlist
 - mo_global_variables, 155
- get_basin_info
 - mo_init_states, 178
- get_basin_info_mrm
 - mo_mrm_tools, 425
- get_distance_two_lat_lon_points
 - mo_mrm_net_startup, 332
- get_info
 - mo_ncread, 465
- get_ncdim
 - mo_ncread, 467
- get_ncdimatt
 - mo_ncread, 468
- get_ncvar_0d_dp
 - mo_ncread, 469
 - mo_ncread::get_ncvar, 818
- get_ncvar_0d_i1
 - mo_ncread, 469
 - mo_ncread::get_ncvar, 818
- get_ncvar_0d_i4
 - mo_ncread, 470
 - mo_ncread::get_ncvar, 818
- get_ncvar_0d_sp
 - mo_ncread, 470
 - mo_ncread::get_ncvar, 818
- get_ncvar_1d_dp
 - mo_ncread, 471
 - mo_ncread::get_ncvar, 818
- get_ncvar_1d_i1
 - mo_ncread, 471
 - mo_ncread::get_ncvar, 818
- get_ncvar_1d_i4
 - mo_ncread, 472
 - mo_ncread::get_ncvar, 819
- get_ncvar_1d_sp
 - mo_ncread, 472
 - mo_ncread::get_ncvar, 819
- get_ncvar_2d_dp
 - mo_ncread, 473
 - mo_ncread::get_ncvar, 819
- get_ncvar_2d_i1
 - mo_ncread, 473
 - mo_ncread::get_ncvar, 819
- get_ncvar_2d_i4
 - mo_ncread, 474
 - mo_ncread::get_ncvar, 820
- get_ncvar_2d_sp
 - mo_ncread, 474
 - mo_ncread::get_ncvar, 820
- get_ncvar_3d_dp
 - mo_ncread, 475
 - mo_ncread::get_ncvar, 820
- get_ncvar_3d_i1
 - mo_ncread, 475
 - mo_ncread::get_ncvar, 820
- get_ncvar_3d_i4
 - mo_ncread, 476
 - mo_ncread::get_ncvar, 820
- get_ncvar_3d_sp
 - mo_ncread, 476
 - mo_ncread::get_ncvar, 821
- get_ncvar_4d_dp
 - mo_ncread, 477

- mo_ncread::get_ncvar, 821
- get_ncvar_4d_i1
 - mo_ncread, 477
 - mo_ncread::get_ncvar, 821
- get_ncvar_4d_i4
 - mo_ncread, 478
 - mo_ncread::get_ncvar, 821
- get_ncvar_4d_sp
 - mo_ncread, 478
 - mo_ncread::get_ncvar, 821
- get_ncvar_5d_dp
 - mo_ncread, 479
 - mo_ncread::get_ncvar, 822
- get_ncvar_5d_i1
 - mo_ncread, 479
 - mo_ncread::get_ncvar, 822
- get_ncvar_5d_i4
 - mo_ncread, 480
 - mo_ncread::get_ncvar, 822
- get_ncvar_5d_sp
 - mo_ncread, 480
 - mo_ncread::get_ncvar, 822
- get_ncvaratt
 - mo_ncread, 481
- get_time
 - mo_read_forcing_nc, 650
- get_timeseed_i4_0d
 - mo_xor4096, 775
 - mo_xor4096::get_timeseed, 823
- get_timeseed_i4_1d
 - mo_xor4096, 775
 - mo_xor4096::get_timeseed, 823
- get_timeseed_i8_0d
 - mo_xor4096, 776
 - mo_xor4096::get_timeseed, 823
- get_timeseed_i8_1d
 - mo_xor4096, 776
 - mo_xor4096::get_timeseed, 823
- getattribute
 - mo_netcdf::ncdataset, 864
 - mo_netcdf::ncdimension, 876
- getdata
 - mo_netcdf::ncdimension, 877
- getdata1df32
 - mo_netcdf, 514
 - mo_netcdf::ncdimension, 877
- getdata1df64
 - mo_netcdf, 514
 - mo_netcdf::ncdimension, 877
- getdata1di16
 - mo_netcdf, 515
 - mo_netcdf::ncdimension, 877
- getdata1di32
 - mo_netcdf, 515
 - mo_netcdf::ncdimension, 877
- getdata1di8
 - mo_netcdf, 516
 - mo_netcdf::ncdimension, 877
- getdata2df32
 - mo_netcdf, 516
 - mo_netcdf::ncdimension, 878
- getdata2df64
 - mo_netcdf, 517
 - mo_netcdf::ncdimension, 878
- getdata2di16
 - mo_netcdf, 517
 - mo_netcdf::ncdimension, 878
- getdata2di32
 - mo_netcdf, 518
 - mo_netcdf::ncdimension, 878
- getdata2di8
 - mo_netcdf, 518
 - mo_netcdf::ncdimension, 878
- getdata3df32
 - mo_netcdf, 519
 - mo_netcdf::ncdimension, 878
- getdata3df64
 - mo_netcdf, 519
 - mo_netcdf::ncdimension, 878
- getdata3di16
 - mo_netcdf, 520
 - mo_netcdf::ncdimension, 878
- getdata3di32
 - mo_netcdf, 520
 - mo_netcdf::ncdimension, 878
- getdata3di8
 - mo_netcdf, 521
 - mo_netcdf::ncdimension, 879
- getdata4df32
 - mo_netcdf, 521
 - mo_netcdf::ncdimension, 879
- getdata4df64
 - mo_netcdf, 522
 - mo_netcdf::ncdimension, 879
- getdata4di16
 - mo_netcdf, 522
 - mo_netcdf::ncdimension, 879
- getdata4di32
 - mo_netcdf, 523
 - mo_netcdf::ncdimension, 879
- getdata4di8
 - mo_netcdf, 523
 - mo_netcdf::ncdimension, 879
- getdata5df32
 - mo_netcdf, 524
 - mo_netcdf::ncdimension, 879
- getdata5df64
 - mo_netcdf, 524
 - mo_netcdf::ncdimension, 879
- getdata5di16
 - mo_netcdf, 525
 - mo_netcdf::ncdimension, 879
- getdata5di32
 - mo_netcdf, 525
 - mo_netcdf::ncdimension, 880
- getdata5di8

- mo_netcdf, 526
- mo_netcdf::ncdimension, 880
- getdatascalarf32
 - mo_netcdf, 526
 - mo_netcdf::ncdimension, 880
- getdatascalarf64
 - mo_netcdf, 527
 - mo_netcdf::ncdimension, 880
- getdatascalarf16
 - mo_netcdf, 527
 - mo_netcdf::ncdimension, 880
- getdatascalarf32
 - mo_netcdf, 528
 - mo_netcdf::ncdimension, 880
- getdatascalarf8
 - mo_netcdf, 528
 - mo_netcdf::ncdimension, 880
- getdimension
 - mo_netcdf::ncdataset, 865
- getdimensionbyid
 - mo_netcdf, 529
 - mo_netcdf::ncdataset, 865
- getdimensionbyname
 - mo_netcdf, 529
 - mo_netcdf::ncdataset, 865
- getdimensionlength
 - mo_netcdf, 529
- getdimensionname
 - mo_netcdf, 530
- getdimensions
 - mo_netcdf::ncdimension, 880
- getdtype
 - mo_netcdf::ncdimension, 880
- getdtypefrominteger
 - mo_netcdf, 530
- getdtypefromstring
 - mo_netcdf, 531
- getfillvalue
 - mo_netcdf::ncdimension, 881
- getglobalattributechar
 - mo_netcdf, 531
 - mo_netcdf::ncdataset, 866
- getglobalattributef32
 - mo_netcdf, 531
 - mo_netcdf::ncdataset, 866
- getglobalattributef64
 - mo_netcdf, 532
 - mo_netcdf::ncdataset, 866
- getglobalattributef16
 - mo_netcdf, 532
 - mo_netcdf::ncdataset, 866
- getglobalattributef32
 - mo_netcdf, 533
 - mo_netcdf::ncdataset, 866
- getglobalattributei8
 - mo_netcdf, 533
 - mo_netcdf::ncdataset, 866
- getname
 - mo_netcdf::ncdimension, 881
- getnodimensions
 - mo_netcdf, 534
 - mo_netcdf::ncdimension, 881
- getnovariables
 - mo_netcdf, 534
 - mo_netcdf::ncdataset, 866
- getpnt
 - mo_sce, 689
- getreaddatashape
 - mo_netcdf, 534
- getshape
 - mo_netcdf::ncdimension, 881
- gettemperature_dp
 - mo_anneal, 77
 - mo_anneal::gettemperature, 825
- getunlimiteddimension
 - mo_netcdf, 535
 - mo_netcdf::ncdataset, 866
- getvariable
 - mo_netcdf::ncdataset, 867
- getvariableattributechar
 - mo_netcdf, 536
 - mo_netcdf::ncdimension, 881
- getvariableattributef32
 - mo_netcdf, 536
 - mo_netcdf::ncdimension, 881
- getvariableattributef64
 - mo_netcdf, 537
 - mo_netcdf::ncdimension, 882
- getvariableattributei16
 - mo_netcdf, 537
 - mo_netcdf::ncdimension, 882
- getvariableattributei32
 - mo_netcdf, 537
 - mo_netcdf::ncdimension, 882
- getvariableattributei8
 - mo_netcdf, 538
 - mo_netcdf::ncdimension, 882
- getvariablebyname
 - mo_netcdf, 538
 - mo_netcdf::ncdataset, 867
- getvariabledimensions
 - mo_netcdf, 539
- getvariabledtype
 - mo_netcdf, 539
- getvariablefillvaluef32
 - mo_netcdf, 539
 - mo_netcdf::ncdimension, 882
- getvariablefillvaluef64
 - mo_netcdf, 539
 - mo_netcdf::ncdimension, 882
- getvariablefillvaluei16
 - mo_netcdf, 540
 - mo_netcdf::ncdimension, 882
- getvariablefillvaluei32
 - mo_netcdf, 540
 - mo_netcdf::ncdimension, 882

- getvariablefillvaluei8
 - mo_netcdf, 540
 - mo_netcdf::ncdimension, 882
- getvariableids
 - mo_netcdf, 540
 - mo_netcdf::ncdataset, 867
- getvariablename
 - mo_netcdf, 540
- getvariables
 - mo_netcdf, 541
 - mo_netcdf::ncdataset, 867
- getvariablesshape
 - mo_netcdf, 541
- given_ts
 - mo_mrm_constants, 290
- global_parameters
 - mo_common_variables, 90
- global_parameters_name
 - mo_common_variables, 90
- gravity_dp
 - mo_constants, 96
- gravity_sp
 - mo_constants, 96
- greaterequal_dp
 - mo_utils, 753
 - mo_utils::ge, 816
 - mo_utils::greaterequal, 825
- greaterequal_sp
 - mo_utils, 753
 - mo_utils::ge, 816
 - mo_utils::greaterequal, 825
- h2odens
 - mo_mhm_constants, 248
- harsamconst
 - mo_mhm_constants, 248
- hasattribute
 - mo_netcdf, 541
 - mo_netcdf::ncdimension, 883
- hasdimension
 - mo_netcdf, 541
 - mo_netcdf::ncdataset, 868
- hasvariable
 - mo_netcdf, 541
 - mo_netcdf::ncdataset, 868
- history
 - mo_global_variables, 156
 - mo_mrm_global_variables, 305
- horizondepth_mhm
 - mo_global_variables, 156
- hour2day_average_dp
 - mo_temporal_aggregation, 731
 - mo_temporal_aggregation::hour2day_average, 829
- hoursecs
 - mo_mrm_constants, 290
- hydro_cond
 - mo_mpr_soilmoist, 283
- i1
 - mo_kind, 218
- i2
 - mo_kind, 218
- i4
 - mo_kind, 219
- i42str
 - mo_string_utils, 724
 - mo_string_utils::num2str, 896
- i4array2str
 - mo_string_utils, 724
 - mo_string_utils::numarray2str, 897
- i8
 - mo_kind, 219
- i82str
 - mo_string_utils, 724
 - mo_string_utils::num2str, 896
- i_ctrper
 - mo_orderpack, 620
 - mo_orderpack::ctrper, 805
- i_fndnth
 - mo_orderpack, 620
 - mo_orderpack::fndnth, 813
- i_indmed
 - mo_orderpack, 620
 - mo_orderpack::indmed, 830
- i_indnth
 - mo_orderpack, 620
 - mo_orderpack::indnth, 830
- i_inspar
 - mo_orderpack, 621
 - mo_orderpack::inspar, 831
- i_inssor
 - mo_orderpack, 621
 - mo_orderpack::inssor, 832
- i_med
 - mo_orderpack, 621
- i_median
 - mo_orderpack, 621
 - mo_orderpack::omedian, 898
- i_mrgref
 - mo_orderpack, 622
 - mo_orderpack::mrgref, 858
- i_mrgrnk
 - mo_orderpack, 622
 - mo_orderpack::mrgrnk, 859
- i_mulcnt
 - mo_orderpack, 622
 - mo_orderpack::mulcnt, 861
- i_nearless
 - mo_orderpack, 622
 - mo_orderpack::nearless, 891
- i_rapknr
 - mo_orderpack, 622
 - mo_orderpack::rapknr, 926
- i_refpar
 - mo_orderpack, 622
 - mo_orderpack::refpar, 928

- i_refsor
 - mo_orderpack, 622
 - mo_orderpack::refsor, 929
 - mo_orderpack::sort, 942
- i_rinpar
 - mo_orderpack, 623
 - mo_orderpack::rinpar, 930
- i_rnkpar
 - mo_orderpack, 623
 - mo_orderpack::rnkpar, 932
- i_subsor
 - mo_orderpack, 623
- i_uniinv
 - mo_orderpack, 624
 - mo_orderpack::uniinv, 956
- i_unipar
 - mo_orderpack, 624
 - mo_orderpack::unipar, 957
- i_unirnk
 - mo_orderpack, 624
 - mo_orderpack::unirnk, 957
- i_unista
 - mo_orderpack, 624
 - mo_orderpack::unista, 958
- i_valmed
 - mo_orderpack, 624
 - mo_orderpack::valmed, 959
- i_valnth
 - mo_orderpack, 624
 - mo_orderpack::valnth, 959
- ibasin
 - mo_mrm_write_fluxes_states::outputdataset, 905
 - mo_write_fluxes_states::outputdataset, 901
- id
 - mo_global_variables::soiltype, 937
 - mo_mrm_write_fluxes_states::outputdataset, 905
 - mo_netcdf::ncdataset, 872
 - mo_netcdf::ncdimension, 889
 - mo_write_fluxes_states::outputdataset, 901
- idont
 - mo_orderpack, 630
- iflag_cordinate_sys
 - mo_global_variables, 156
 - mo_mrm_global_variables, 305
- iflag_soildb
 - mo_global_variables, 156
- in_bound
 - mo_mrm_read_config, 387
 - mo_read_config, 647
- inflowgauge
 - mo_mrm_global_variables, 306
- inflowgaugeheadwater
 - mo_mrm_global_variables::basininfo, 795
- inflowgaugeidlist
 - mo_mrm_global_variables::basininfo, 795
- inflowgaugeindexlist
 - mo_mrm_global_variables::basininfo, 795
- inflowgaugenodelist
 - mo_mrm_global_variables::basininfo, 795
- initialise
 - mo_startup, 714
- initncdataset
 - mo_netcdf, 541
 - mo_netcdf::ncdataset, 868
- initncdimension
 - mo_netcdf, 542
- initncvariable
 - mo_netcdf, 542
 - mo_netcdf::ncdimension, 883
- inputformat_gridded_lai
 - mo_global_variables, 156
- inputformat_meteo_forcings
 - mo_global_variables, 156
- intgrandfast
 - mo_neutrons, 571
- iper_thres_runoff
 - mo_multi_param_reg, 452
- irow
 - mo_kind::sprs2_dp, 947
 - mo_kind::sprs2_sp, 948
- is_finite_dp
 - mo_utils, 753
 - mo_utils::is_finite, 833
- is_finite_sp
 - mo_utils, 753
 - mo_utils::is_finite, 833
- is_nan_dp
 - mo_utils, 754
 - mo_utils::is_nan, 833
- is_nan_sp
 - mo_utils, 754
 - mo_utils::is_nan, 833
- is_normal_dp
 - mo_utils, 754
 - mo_utils::is_normal, 834
- is_normal_sp
 - mo_utils, 754
 - mo_utils::is_normal, 834
- is_present
 - mo_global_variables::soiltype, 937
- is_read
 - mo_meteo_forcings, 230
- is_start
 - mo_mrm_global_variables, 306
- isdatasetunlimited
 - mo_netcdf, 542
- isunlimited
 - mo_netcdf::ncdataset, 869
 - mo_netcdf::ncdimension, 883
- isunlimitedddimension
 - mo_netcdf, 543
- isunlimitedvariable
 - mo_netcdf, 543
- itest
 - mo_template, 729
- jarvis_et_reduction

- mo_soil_moisture, 703
- jcol
 - mo_kind::sprs2_dp, 947
 - mo_kind::sprs2_sp, 949
- julday
 - mo_julian, 207
- julday360
 - mo_julian, 209
- julday365
 - mo_julian, 210
- juldayjulian
 - mo_julian, 211
- julend
 - mo_common_variables::period, 924
- julstart
 - mo_common_variables::period, 924
- karman
 - mo_mhm_constants, 248
- karstic_layer
 - mo_multi_param_reg, 453
- kge_dp_1d
 - mo_errormeasures, 115
 - mo_errormeasures::kge, 835
- kge_dp_2d
 - mo_errormeasures, 115
 - mo_errormeasures::kge, 835
- kge_dp_3d
 - mo_errormeasures, 115
 - mo_errormeasures::kge, 836
- kge_sp_1d
 - mo_errormeasures, 115
 - mo_errormeasures::kge, 836
- kge_sp_2d
 - mo_errormeasures, 116
 - mo_errormeasures::kge, 836
- kge_sp_3d
 - mo_errormeasures, 116
 - mo_errormeasures::kge, 836
- kgenocorr_dp_1d
 - mo_errormeasures, 116
 - mo_errormeasures::kgenocorr, 837
- kgenocorr_dp_2d
 - mo_errormeasures, 116
 - mo_errormeasures::kgenocorr, 837
- kgenocorr_dp_3d
 - mo_errormeasures, 116
 - mo_errormeasures::kgenocorr, 838
- kgenocorr_sp_1d
 - mo_errormeasures, 116
 - mo_errormeasures::kgenocorr, 838
- kgenocorr_sp_2d
 - mo_errormeasures, 117
 - mo_errormeasures::kgenocorr, 838
- kgenocorr_sp_3d
 - mo_errormeasures, 117
 - mo_errormeasures::kgenocorr, 838
- ks
 - mo_global_variables::soiltype, 937
- ks_c
 - mo_mhm_constants, 248
- kurtosis_dp
 - mo_moment, 261
 - mo_moment::kurtosis, 839
- kurtosis_sp
 - mo_moment, 261
 - mo_moment::kurtosis, 839
- l0_areacell
 - mo_global_variables, 156
 - mo_mrm_global_variables, 306
- l0_asp
 - mo_global_variables, 156
- l0_basin
 - mo_global_variables, 157
 - mo_mrm_global_variables, 306
- l0_cellcoor
 - mo_global_variables, 157
 - mo_mrm_global_variables, 306
- l0_check_input
 - mo_startup, 716
- l0_check_input_routing
 - mo_mrm_init, 321
- l0_coloutlet
 - mo_mrm_global_variables::basininfo, 796
- l0_dracell
 - mo_mrm_global_variables, 306
- l0_drasc
 - mo_mrm_global_variables, 306
- l0_elev
 - mo_global_variables, 157
- l0_elev_mrm
 - mo_mrm_global_variables, 306
- l0_elev_read
 - mo_mrm_global_variables, 307
- l0_facc
 - mo_mrm_global_variables, 307
- l0_fdir
 - mo_mrm_global_variables, 307
- l0_floodplain
 - mo_mrm_global_variables, 307
- l0_fractionalcover_in_lx
 - mo_upscaling_operators, 744
- l0_gaugeloc
 - mo_mrm_global_variables, 307
- l0_geounit
 - mo_global_variables, 157
- l0_gridded_lai
 - mo_global_variables, 157
- l0_id
 - mo_global_variables, 157
 - mo_mrm_global_variables, 307
- l0_iend
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 796
- l0_iendmask
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 796

- l0_inflowgaugeloc
 - mo_mrm_global_variables, 307
- l0_istart
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 796
- l0_istartmask
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 796
- l0_l11_id
 - mo_mrm_global_variables, 307
- l0_latitude
 - mo_global_variables, 157
 - mo_mrm_global_variables, 308
- l0_lcover
 - mo_global_variables, 158
- l0_lcover_lai
 - mo_global_variables, 158
- l0_lcover_mrm
 - mo_mrm_global_variables, 308
- l0_lcover_read
 - mo_mrm_global_variables, 308
- l0_longitude
 - mo_global_variables, 158
 - mo_mrm_global_variables, 308
- l0_mask
 - mo_global_variables, 158
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 796
- l0_mask_mrm
 - mo_mrm_global_variables, 308
- l0_ncells
 - mo_global_variables, 158
 - mo_mrm_global_variables, 308
- l0_noutlet
 - mo_mrm_global_variables::basininfo, 796
- l0_rowoutlet
 - mo_mrm_global_variables::basininfo, 796
- l0_slope
 - mo_global_variables, 158
- l0_slope_emp
 - mo_global_variables, 158
- l0_soilid
 - mo_global_variables, 158
- l0_streamnet
 - mo_mrm_global_variables, 308
- l0_variable_init
 - mo_startup, 717
- l110_iend
 - mo_mrm_global_variables::basininfo, 796
- l110_istart
 - mo_mrm_global_variables::basininfo, 797
- l11_afloodplain
 - mo_mrm_global_variables, 308
- l11_areacell
 - mo_mrm_global_variables, 309
- l11_c1
 - mo_mrm_global_variables, 309
- l11_c2
 - mo_mrm_global_variables, 309
- l11_cellcoor
 - mo_mrm_global_variables, 309
- l11_colout
 - mo_mrm_global_variables, 309
- l11_downbound_l0
 - mo_mrm_global_variables, 309
- l11_downbound_l1
 - mo_mrm_global_variables, 309
- l11_fcol
 - mo_mrm_global_variables, 310
- l11_fdir
 - mo_mrm_global_variables, 310
- l11_flow_direction
 - mo_mrm_net_startup, 333
- l11_fracpimp
 - mo_mrm_global_variables, 310
- l11_fraction_sealed_floodplain
 - mo_mrm_net_startup, 335
- l11_fromn
 - mo_mrm_global_variables, 310
- l11_frow
 - mo_mrm_global_variables, 310
- l11_id
 - mo_mrm_global_variables, 310
- l11_iend
 - mo_mrm_global_variables::basininfo, 797
- l11_iendmask
 - mo_mrm_global_variables::basininfo, 797
- l11_istart
 - mo_mrm_global_variables::basininfo, 797
- l11_istartmask
 - mo_mrm_global_variables::basininfo, 797
- l11_k
 - mo_mrm_global_variables, 310
- l11_l1_id
 - mo_mrm_global_variables, 311
- l11_label
 - mo_mrm_global_variables, 311
- l11_leftbound_l0
 - mo_mrm_global_variables, 311
- l11_leftbound_l1
 - mo_mrm_global_variables, 311
- l11_length
 - mo_mrm_global_variables, 311
- l11_link_location
 - mo_mrm_net_startup, 336
- l11_mask
 - mo_mrm_global_variables::basininfo, 797
- l11_ncells
 - mo_mrm_global_variables, 311
- l11_netperm
 - mo_mrm_global_variables, 311
- l11_noutlets
 - mo_mrm_global_variables, 312
- l11_qmod
 - mo_mrm_global_variables, 312
- l11_qout

- mo_mrm_global_variables, 312
- l11_qtin
 - mo_mrm_global_variables, 312
- l11_qtr
 - mo_mrm_global_variables, 312
- l11_rect_latitude
 - mo_mrm_global_variables, 312
- l11_rect_longitude
 - mo_mrm_global_variables, 312
- l11_rightbound_l0
 - mo_mrm_global_variables, 313
- l11_rightbound_l1
 - mo_mrm_global_variables, 313
- l11_rorder
 - mo_mrm_global_variables, 313
- l11_routing
 - mo_mrm_routing, 406
- l11_routing_order
 - mo_mrm_net_startup, 337
- l11_rowout
 - mo_mrm_global_variables, 313
- l11_runoff_acc
 - mo_mrm_routing, 408
- l11_set_drain_outlet_gauges
 - mo_mrm_net_startup, 338
- l11_set_network_topology
 - mo_mrm_net_startup, 340
- l11_sink
 - mo_mrm_global_variables, 313
- l11_slope
 - mo_mrm_global_variables, 313
- l11_stream_features
 - mo_mrm_net_startup, 341
- l11_tcol
 - mo_mrm_global_variables, 313
- l11_ton
 - mo_mrm_global_variables, 314
- l11_trow
 - mo_mrm_global_variables, 314
- l11_tsrout
 - mo_mrm_global_variables, 314
- l11_upbound_l0
 - mo_mrm_global_variables, 314
- l11_upbound_l1
 - mo_mrm_global_variables, 314
- l11_variable_init
 - mo_mrm_net_startup, 342
- l11_xi
 - mo_mrm_global_variables, 314
- l1_absvappress
 - mo_global_variables, 159
- l1_aeroresist
 - mo_global_variables, 159
- l1_aetcanopy
 - mo_global_variables, 159
- l1_aetsealed
 - mo_global_variables, 159
- l1_aetsoil
 - mo_global_variables, 159
- l1_alpha
 - mo_global_variables, 159
- l1_areacell
 - mo_global_variables, 159
 - mo_mrm_global_variables, 314
- l1_baseflow
 - mo_global_variables, 159
- l1_cellcoor
 - mo_global_variables, 160
- l1_degday
 - mo_global_variables, 160
- l1_degdayinc
 - mo_global_variables, 160
- l1_degdaymax
 - mo_global_variables, 160
- l1_degdaynopre
 - mo_global_variables, 160
- l1_downbound_l0
 - mo_global_variables, 160
- l1_et
 - mo_global_variables, 160
- l1_et_mask
 - mo_global_variables, 161
- l1_fasp
 - mo_global_variables, 161
- l1_fastrunoff
 - mo_global_variables, 161
- l1_fforest
 - mo_global_variables, 161
- l1_fperm
 - mo_global_variables, 161
- l1_froots
 - mo_global_variables, 161
- l1_fsealed
 - mo_global_variables, 162
- l1_harsamcoeff
 - mo_global_variables, 162
- l1_id
 - mo_global_variables, 162
 - mo_mrm_global_variables, 315
- l1_iend
 - mo_global_variables::basininfo, 792
 - mo_mrm_global_variables::basininfo, 797
- l1_iendmask
 - mo_global_variables::basininfo, 793
 - mo_mrm_global_variables::basininfo, 797
- l1_infilsoil
 - mo_global_variables, 162
- l1_inter
 - mo_global_variables, 162
- l1_istart
 - mo_global_variables::basininfo, 793
 - mo_mrm_global_variables::basininfo, 797
- l1_istartmask
 - mo_global_variables::basininfo, 793
 - mo_mrm_global_variables::basininfo, 798
- l1_jarvis_thresh_c1

- mo_global_variables, 162
- l1_karstloss
 - mo_global_variables, 162
- l1_kbaseflow
 - mo_global_variables, 163
- l1_kfastflow
 - mo_global_variables, 163
- l1_kperco
 - mo_global_variables, 163
- l1_kslowflow
 - mo_global_variables, 163
- l1_l11_id
 - mo_mrm_global_variables, 315
- l1_latitude
 - mo_global_variables, 163
- l1_leftbound_l0
 - mo_global_variables, 163
- l1_longitude
 - mo_global_variables, 163
- l1_mask
 - mo_global_variables::basininfo, 793
 - mo_mrm_global_variables::basininfo, 798
- l1_maxinter
 - mo_global_variables, 164
- l1_melt
 - mo_global_variables, 164
- l1_ncells
 - mo_global_variables, 164
 - mo_mrm_global_variables, 315
- l1_netrad
 - mo_global_variables, 164
- l1_neutrons
 - mo_global_variables, 164
- l1_neutronsdata
 - mo_global_variables, 164
- l1_neutronsdata_mask
 - mo_global_variables, 164
- l1_ntcells_l0
 - mo_global_variables, 164
- l1_percol
 - mo_global_variables, 165
- l1_pet
 - mo_global_variables, 165
- l1_pet_calc
 - mo_global_variables, 165
- l1_pet_weights
 - mo_global_variables, 165
- l1_petlaicorfactor
 - mo_global_variables, 165
- l1_pre
 - mo_global_variables, 165
- l1_pre_weights
 - mo_global_variables, 165
- l1_preeffect
 - mo_global_variables, 165
- l1_prietayalpha
 - mo_global_variables, 166
- l1_rain
 - mo_global_variables, 166
- l1_rect_latitude
 - mo_global_variables, 166
- l1_rect_longitude
 - mo_global_variables, 166
- l1_rightbound_l0
 - mo_global_variables, 166
- l1_runoffseal
 - mo_global_variables, 166
- l1_satstw
 - mo_global_variables, 166
- l1_sealedthresh
 - mo_global_variables, 167
- l1_sealstw
 - mo_global_variables, 167
- l1_slowrunoff
 - mo_global_variables, 167
- l1_sm
 - mo_global_variables, 167
- l1_sm_mask
 - mo_global_variables, 167
- l1_snow
 - mo_global_variables, 167
- l1_snowpack
 - mo_global_variables, 167
- l1_soilmoist
 - mo_global_variables, 168
- l1_soilmoistexp
 - mo_global_variables, 168
- l1_soilmoistfc
 - mo_global_variables, 168
- l1_soilmoistsat
 - mo_global_variables, 168
- l1_surfresist
 - mo_global_variables, 168
- l1_temp
 - mo_global_variables, 168
- l1_temp_weights
 - mo_global_variables, 168
- l1_tempthresh
 - mo_global_variables, 169
- l1_throughfall
 - mo_global_variables, 169
- l1_tmax
 - mo_global_variables, 169
- l1_tmin
 - mo_global_variables, 169
- l1_total_runoff
 - mo_global_variables, 169
 - mo_runoff, 681
- l1_total_runoff_in
 - mo_mrm_global_variables, 315
- l1_unsatstw
 - mo_global_variables, 169
- l1_unsatthresh
 - mo_global_variables, 169
- l1_upbound_l0
 - mo_global_variables, 170

- l1_variable_init
 - mo_startup, 718
- l1_wiltingpoint
 - mo_global_variables, 170
- l1_windspeed
 - mo_global_variables, 170
- l2_iend
 - mo_global_variables::basininfo, 793
- l2_iendmask
 - mo_global_variables::basininfo, 793
- l2_istart
 - mo_global_variables::basininfo, 793
- l2_istartmask
 - mo_global_variables::basininfo, 793
- l2_mask
 - mo_global_variables::basininfo, 793
- l2_variable_init
 - mo_startup, 719
- lai_factor_surfresi
 - mo_mhm_constants, 248
- lai_offset_surfresi
 - mo_mhm_constants, 249
- lailut
 - mo_global_variables, 170
- laiunitlist
 - mo_global_variables, 170
- lats
 - mo_global_variables, 170
- lcfilename
 - mo_global_variables, 170
 - mo_mrm_global_variables, 315
- lcyearid
 - mo_global_variables, 170
 - mo_mrm_global_variables, 315
- ld
 - mo_global_variables::soiltype, 937
- len
 - mo_kind::sprs2_dp, 947
 - mo_kind::sprs2_sp, 949
 - mo_ncwrite::dims, 807
- length_error
 - mo_nml, 579
- lesserequal_dp
 - mo_utils, 754
 - mo_utils::le, 839
 - mo_utils::lesserequal, 840
- lesserequal_sp
 - mo_utils, 754
 - mo_utils::le, 839
 - mo_utils::lesserequal, 840
- level0
 - mo_global_variables, 171
 - mo_mrm_global_variables, 315
- level1
 - mo_global_variables, 171
 - mo_mrm_global_variables, 316
- level11
 - mo_mrm_global_variables, 316
- level110
 - mo_mrm_global_variables, 316
- level2
 - mo_global_variables, 171
- lgt
 - mo_kind, 219
- limb_densities
 - mo_mrm_signatures, 416
- linfit_dp
 - mo_linfit, 220
 - mo_linfit::linfit, 841
- linfit_sp
 - mo_linfit, 220
 - mo_linfit::linfit, 841
- Innse_dp_1d
 - mo_errormeasures, 117
 - mo_errormeasures::Innse, 842
- Innse_dp_2d
 - mo_errormeasures, 117
 - mo_errormeasures::Innse, 842
- Innse_dp_3d
 - mo_errormeasures, 117
 - mo_errormeasures::Innse, 842
- Innse_sp_1d
 - mo_errormeasures, 117
 - mo_errormeasures::Innse, 842
- Innse_sp_2d
 - mo_errormeasures, 117
 - mo_errormeasures::Innse, 842
- Innse_sp_3d
 - mo_errormeasures, 118
 - mo_errormeasures::Innse, 842
- locate_0d_dp
 - mo_utils, 754
 - mo_utils::locate, 843
- locate_0d_sp
 - mo_utils, 755
 - mo_utils::locate, 843
- locate_1d_dp
 - mo_utils, 755
 - mo_utils::locate, 844
- locate_1d_sp
 - mo_utils, 755
 - mo_utils::locate, 844
- log2str
 - mo_string_utils, 724
 - mo_string_utils::num2str, 896
- loglikelihood
 - mo_mrm_objective_function_runoff, 350
- loglikelihood_evin2013_2
 - mo_mrm_objective_function_runoff, 353
- loglikelihood_stddev
 - mo_mrm_objective_function_runoff, 355
- loglikelihood_trend_no_autocorr
 - mo_mrm_objective_function_runoff, 357
- lons
 - mo_global_variables, 171
- lookupintegral

- mo_neutrons, 572
- mae_dp_1d
 - mo_errormeasures, 118
 - mo_errormeasures::mae, 844
- mae_dp_2d
 - mo_errormeasures, 118
 - mo_errormeasures::mae, 844
- mae_dp_3d
 - mo_errormeasures, 119
 - mo_errormeasures::mae, 845
- mae_sp_1d
 - mo_errormeasures, 119
 - mo_errormeasures::mae, 845
- mae_sp_2d
 - mo_errormeasures, 120
 - mo_errormeasures::mae, 845
- mae_sp_3d
 - mo_errormeasures, 120
 - mo_errormeasures::mae, 845
- majority_statistics
 - mo_upscaling_operators, 745
- mapcoordinates
 - mo_mrm_write_fluxes_states, 440
 - mo_write_fluxes_states, 766
- mask
 - mo_mrm_write_fluxes_states::outputvariable, 914
 - mo_write_fluxes_states::outputvariable, 909
- max_surfresist
 - mo_mhm_constants, 249
- max_timers
 - mo_timer, 743
- maxgeounit
 - mo_mhm_constants, 249
- maximummonthlyflow
 - mo_mrm_signatures, 417
- maxlen
 - mo_ncwrite, 508
- maxnlcovers
 - mo_mhm_constants, 249
 - mo_mrm_constants, 290
- maxnobasins
 - mo_mhm_constants, 249
 - mo_mrm_constants, 290
- maxnogauges
 - mo_mrm_constants, 290
- maxnooilhorizons
 - mo_mhm_constants, 249
- mcmc_dp
 - mo_mcmc, 222
 - mo_mcmc::mcmc, 849
- mcmc_error_params
 - mo_common_variables, 90
- mcmc_opti
 - mo_common_variables, 90
- mcmc_stddev_dp
 - mo_mcmc, 223
 - mo_mcmc::mcmc_stddev, 853
- mdds
 - mo_dds, 111
- mean_dp
 - mo_moment, 261
 - mo_moment::mean, 854
 - mo_template, 729
 - mo_template::mean, 855
- mean_sp
 - mo_moment, 262
 - mo_moment::mean, 854
 - mo_template, 729
 - mo_template::mean, 855
- median_dp
 - mo_percentile, 630
 - mo_percentile::median, 855
- median_sp
 - mo_percentile, 631
 - mo_percentile::median, 855
- mend
 - mo_common_variables::period, 924
- message
 - mo_message, 225
- message_text
 - mo_message, 226
- meteo_forcings_wrapper
 - mo_meteo_forcings, 231
- meteo_weights_wrapper
 - mo_meteo_forcings, 234
- mhm
 - mo_mhm, 238
- mhm_details
 - mo_global_variables, 171
 - mo_mrm_global_variables, 316
- mhm_driver
 - mhm_driver.f90, 977
- mhm_driver.f90, 977
 - mhm_driver, 977
- mhm_eval
 - mo_mhm_eval, 255
- mhm_papers.md, 980
- missing
 - mo_nml, 579
- mixed_central_moment_dp
 - mo_moment, 262
 - mo_moment::mixed_central_moment, 856
- mixed_central_moment_sp
 - mo_moment, 262
 - mo_moment::mixed_central_moment, 856
- mixed_central_moment_var_dp
 - mo_moment, 262
 - mo_moment::mixed_central_moment_var, 857
- mixed_central_moment_var_sp
 - mo_moment, 262
 - mo_moment::mixed_central_moment_var, 857
- mo_anneal, 75
 - anneal_dp, 75
 - dchange_dp, 76
 - generate_neighborhood_weight_dp, 77
 - gettemperature_dp, 77

- pargen_anneal_dp, 78
- pargen_dds_dp, 78
- mo_anneal.f90, 980
- mo_anneal::anneal, 779
 - anneal_dp, 781
- mo_anneal::generate_neighborhood_weight, 817
 - generate_neighborhood_weight_dp, 817
- mo_anneal::gettemperature, 823
 - gettemperature_dp, 825
- mo_append, 79
 - append_char_3d, 80
 - append_char_m_m, 80
 - append_char_v_s, 81
 - append_char_v_v, 81
 - append_dp_3d, 81
 - append_dp_m_m, 81
 - append_dp_v_s, 81
 - append_dp_v_v, 81
 - append_i4_3d, 82
 - append_i4_m_m, 82
 - append_i4_v_s, 82
 - append_i4_v_v, 82
 - append_i8_3d, 82
 - append_i8_m_m, 82
 - append_i8_v_s, 82
 - append_i8_v_v, 83
 - append_lgt_3d, 83
 - append_lgt_m_m, 83
 - append_lgt_v_s, 83
 - append_lgt_v_v, 83
 - append_sp_3d, 83
 - append_sp_m_m, 84
 - append_sp_v_s, 84
 - append_sp_v_v, 84
 - paste_char_m_m, 84
 - paste_char_m_s, 84
 - paste_char_m_v, 84
 - paste_dp_m_m, 84
 - paste_dp_m_s, 85
 - paste_dp_m_v, 85
 - paste_i4_m_m, 85
 - paste_i4_m_s, 85
 - paste_i4_m_v, 85
 - paste_i8_m_m, 85
 - paste_i8_m_s, 85
 - paste_i8_m_v, 86
 - paste_lgt_m_m, 86
 - paste_lgt_m_s, 86
 - paste_lgt_m_v, 86
 - paste_sp_m_m, 86
 - paste_sp_m_s, 86
 - paste_sp_m_v, 86
- mo_append.f90, 981
- mo_append::append, 782
 - append_char_3d, 783
 - append_char_m_m, 783
 - append_char_v_s, 783
 - append_char_v_v, 783
- append_dp_3d, 783
- append_dp_m_m, 784
- append_dp_v_s, 784
- append_dp_v_v, 784
- append_i4_m_m, 784
- append_i4_v_s, 784
- append_i4_v_v, 784
- append_i8_3d, 785
- append_i8_m_m, 785
- append_i8_v_s, 785
- append_i8_v_v, 785
- append_lgt_3d, 785
- append_lgt_m_m, 785
- append_lgt_v_s, 785
- append_lgt_v_v, 786
- append_sp_3d, 786
- append_sp_m_m, 786
- append_sp_v_s, 786
- append_sp_v_v, 786
- mo_append::paste, 916
 - paste_char_m_m, 917
 - paste_char_m_s, 917
 - paste_char_m_v, 918
 - paste_dp_m_m, 918
 - paste_dp_m_s, 918
 - paste_dp_m_v, 918
 - paste_i4_m_m, 918
 - paste_i4_m_s, 918
 - paste_i4_m_v, 918
 - paste_i8_m_m, 919
 - paste_i8_m_s, 919
 - paste_i8_m_v, 919
 - paste_lgt_m_m, 919
 - paste_lgt_m_s, 919
 - paste_lgt_m_v, 919
 - paste_sp_m_m, 920
 - paste_sp_m_s, 920
 - paste_sp_m_v, 920
- mo_canopy_interc, 87
 - canopy_interc, 87
- mo_canopy_interc.f90, 982
- mo_common_variables, 89
 - alma_convention, 89
 - dds_r, 89
 - global_parameters, 90
 - global_parameters_name, 90
 - mcmc_error_params, 90
 - mcmc_opti, 90
 - nerror_model, 90
 - niterations, 90
 - nprocesses, 90
 - opti_function, 90
 - opti_method, 91
 - optimize, 91
 - optimize_restart, 91
 - processmatrix, 91
 - sa_temp, 91
 - sce_ngs, 91

- sce_npg, 91
- sce_nps, 92
- seed, 92
- mo_common_variables.f90, 982
- mo_common_variables::period, 923
 - dend, 924
 - dstart, 924
 - julend, 924
 - julstart, 924
 - mend, 924
 - mstart, 924
 - nobs, 924
 - yend, 924
 - ystart, 925
- mo_constants, 92
 - cp0_dp, 94
 - cp0_sp, 94
 - deg2rad_dp, 95
 - deg2rad_sp, 95
 - eps_dp, 95
 - eps_sp, 95
 - euler, 95
 - euler_d, 95
 - gravity_dp, 96
 - gravity_sp, 96
 - nerr, 96
 - nin, 96
 - nnml, 96
 - nout, 96
 - p0_dp, 96
 - p0_sp, 96
 - pi, 97
 - pi_d, 97
 - pi_dp, 97
 - pi_sp, 97
 - pio2, 97
 - pio2_d, 97
 - pio2_dp, 97
 - pio2_sp, 98
 - psychro_dp, 98
 - psychro_sp, 98
 - rad2deg_dp, 98
 - rad2deg_sp, 98
 - radiusearth_dp, 98
 - radiusearth_sp, 98
 - rho0_dp, 99
 - rho0_sp, 99
 - secday_dp, 99
 - secday_sp, 99
 - sigma_dp, 99
 - sigma_sp, 99
 - solarconst_dp, 99
 - solarconst_sp, 100
 - specheatet_dp, 100
 - specheatet_sp, 100
 - sqrt2, 100
 - sqrt2_d, 100
 - sqrt2_dp, 100
 - sqrt2_sp, 100
 - t0_dp, 101
 - t0_sp, 101
 - twopi, 101
 - twopi_d, 101
 - twopi_dp, 101
 - twopi_sp, 101
 - twothird_dp, 101
 - twothird_sp, 102
- mo_constants.f90, 983
- mo_corr, 102
 - arth_dp, 103
 - arth_i4, 103
 - arth_sp, 103
 - autocoeffk_1d_dp, 103
 - autocoeffk_1d_sp, 103
 - autocoeffk_dp, 104
 - autocoeffk_sp, 104
 - autocorr_1d_dp, 104
 - autocorr_1d_sp, 104
 - autocorr_dp, 104
 - autocorr_sp, 104
 - corr_dp, 104
 - corr_sp, 105
 - crosscoeffk_dp, 105
 - crosscoeffk_sp, 105
 - crosscorr_dp, 105
 - crosscorr_sp, 105
 - four1_dp, 106
 - four1_sp, 106
 - fourrow_dp, 106
 - fourrow_sp, 106
 - npar2_arth, 108
 - npar_arth, 108
 - realft_dp, 106
 - realft_sp, 107
 - swap_1d_dpc, 107
 - swap_1d_spc, 107
 - zroots_unity_dp, 107
 - zroots_unity_sp, 108
- mo_corr.f90, 985
- mo_corr::arth, 786
 - arth_dp, 787
 - arth_i4, 787
 - arth_sp, 787
- mo_corr::autocoeffk, 789
 - autocoeffk_1d_dp, 789
 - autocoeffk_1d_sp, 789
 - autocoeffk_dp, 789
 - autocoeffk_sp, 789
- mo_corr::autocorr, 790
 - autocorr_1d_dp, 790
 - autocorr_1d_sp, 790
 - autocorr_dp, 790
 - autocorr_sp, 790
- mo_corr::corr, 802
 - corr_dp, 802
 - corr_sp, 802

- mo_corr::crosscoeffk, [804](#)
 - crosscoeffk_dp, [804](#)
 - crosscoeffk_sp, [804](#)
- mo_corr::crosscorr, [804](#)
 - crosscorr_dp, [804](#)
 - crosscorr_sp, [805](#)
- mo_corr::four1, [814](#)
 - four1_dp, [814](#)
 - four1_sp, [814](#)
- mo_corr::fourrow, [814](#)
 - fourrow_dp, [814](#)
 - fourrow_sp, [815](#)
- mo_corr::realft, [927](#)
 - realft_dp, [928](#)
 - realft_sp, [928](#)
- mo_corr::swap, [952](#)
 - swap_1d_dpc, [952](#)
 - swap_1d_spc, [953](#)
- mo_dds, [109](#)
 - dds, [109](#)
 - mdds, [111](#)
 - neigh_value, [112](#)
- mo_dds.f90, [986](#)
- mo_errormeasures, [113](#)
 - bias_dp_1d, [114](#)
 - bias_dp_2d, [114](#)
 - bias_dp_3d, [114](#)
 - bias_sp_1d, [114](#)
 - bias_sp_2d, [115](#)
 - bias_sp_3d, [115](#)
 - kge_dp_1d, [115](#)
 - kge_dp_2d, [115](#)
 - kge_dp_3d, [115](#)
 - kge_sp_1d, [115](#)
 - kge_sp_2d, [116](#)
 - kge_sp_3d, [116](#)
 - kgenocorr_dp_1d, [116](#)
 - kgenocorr_dp_2d, [116](#)
 - kgenocorr_dp_3d, [116](#)
 - kgenocorr_sp_1d, [116](#)
 - kgenocorr_sp_2d, [117](#)
 - kgenocorr_sp_3d, [117](#)
 - lnnse_dp_1d, [117](#)
 - lnnse_dp_2d, [117](#)
 - lnnse_dp_3d, [117](#)
 - lnnse_sp_1d, [117](#)
 - lnnse_sp_2d, [117](#)
 - lnnse_sp_3d, [118](#)
 - mae_dp_1d, [118](#)
 - mae_dp_2d, [118](#)
 - mae_dp_3d, [119](#)
 - mae_sp_1d, [119](#)
 - mae_sp_2d, [120](#)
 - mae_sp_3d, [120](#)
 - mse_dp_1d, [120](#)
 - mse_dp_2d, [121](#)
 - mse_dp_3d, [122](#)
 - mse_sp_1d, [122](#)
 - mse_sp_2d, [123](#)
 - mse_sp_3d, [124](#)
 - nse_dp_1d, [124](#)
 - nse_dp_2d, [125](#)
 - nse_dp_3d, [125](#)
 - nse_sp_1d, [125](#)
 - nse_sp_2d, [125](#)
 - nse_sp_3d, [125](#)
 - rmse_dp_1d, [125](#)
 - rmse_dp_2d, [126](#)
 - rmse_dp_3d, [126](#)
 - rmse_sp_1d, [126](#)
 - rmse_sp_2d, [127](#)
 - rmse_sp_3d, [127](#)
 - sae_dp_1d, [128](#)
 - sae_dp_2d, [128](#)
 - sae_dp_3d, [129](#)
 - sae_sp_1d, [130](#)
 - sae_sp_2d, [130](#)
 - sae_sp_3d, [131](#)
 - sse_dp_1d, [132](#)
 - sse_dp_2d, [132](#)
 - sse_dp_3d, [133](#)
 - sse_sp_1d, [134](#)
 - sse_sp_2d, [134](#)
 - sse_sp_3d, [135](#)
- mo_errormeasures.f90, [987](#)
- mo_errormeasures::bias, [798](#)
 - bias_dp_1d, [798](#)
 - bias_dp_2d, [798](#)
 - bias_dp_3d, [799](#)
 - bias_sp_1d, [799](#)
 - bias_sp_2d, [799](#)
 - bias_sp_3d, [799](#)
- mo_errormeasures::kge, [834](#)
 - kge_dp_1d, [835](#)
 - kge_dp_2d, [835](#)
 - kge_dp_3d, [836](#)
 - kge_sp_1d, [836](#)
 - kge_sp_2d, [836](#)
 - kge_sp_3d, [836](#)
- mo_errormeasures::kgenocorr, [836](#)
 - kgenocorr_dp_1d, [837](#)
 - kgenocorr_dp_2d, [837](#)
 - kgenocorr_dp_3d, [838](#)
 - kgenocorr_sp_1d, [838](#)
 - kgenocorr_sp_2d, [838](#)
 - kgenocorr_sp_3d, [838](#)
- mo_errormeasures::lnnse, [841](#)
 - lnnse_dp_1d, [842](#)
 - lnnse_dp_2d, [842](#)
 - lnnse_dp_3d, [842](#)
 - lnnse_sp_1d, [842](#)
 - lnnse_sp_2d, [842](#)
 - lnnse_sp_3d, [842](#)
- mo_errormeasures::mae, [844](#)
 - mae_dp_1d, [844](#)
 - mae_dp_2d, [844](#)

- mae_dp_3d, 845
- mae_sp_1d, 845
- mae_sp_2d, 845
- mae_sp_3d, 845
- mo_errormeasures::mse, 859
 - mse_dp_1d, 859
 - mse_dp_2d, 860
 - mse_dp_3d, 860
 - mse_sp_1d, 860
 - mse_sp_2d, 860
 - mse_sp_3d, 860
- mo_errormeasures::nse, 894
 - nse_dp_1d, 894
 - nse_dp_2d, 894
 - nse_dp_3d, 894
 - nse_sp_1d, 895
 - nse_sp_2d, 895
 - nse_sp_3d, 895
- mo_errormeasures::rmse, 930
 - rmse_dp_1d, 930
 - rmse_dp_2d, 931
 - rmse_dp_3d, 931
 - rmse_sp_1d, 931
 - rmse_sp_2d, 931
 - rmse_sp_3d, 931
- mo_errormeasures::sae, 932
 - sae_dp_1d, 933
 - sae_dp_2d, 933
 - sae_dp_3d, 933
 - sae_sp_1d, 933
 - sae_sp_2d, 933
 - sae_sp_3d, 933
- mo_errormeasures::sse, 949
 - sse_dp_1d, 949
 - sse_dp_2d, 949
 - sse_dp_3d, 950
 - sse_sp_1d, 950
 - sse_sp_2d, 950
 - sse_sp_3d, 950
- mo_file, 136
 - file_aspect, 138
 - file_config, 138
 - file_daily_discharge, 138
 - file_defoutput, 139
 - file_dem, 139
 - file_facc, 139
 - file_fdir, 139
 - file_geolut, 139
 - file_hydrogeoclass, 139
 - file_lai_binary_end, 139
 - file_lai_header, 140
 - file_laiclass, 140
 - file_lailut, 140
 - file_main, 140
 - file_meteo_binary_end, 140
 - file_meteo_header, 140
 - file_namelist, 140
 - file_namelist_param, 141
 - file_opti, 141
 - file_opti_nml, 141
 - file_slope, 141
 - file_soil_database, 141
 - file_soil_database_1, 141
 - file_soilclass, 141
 - uaspect, 142
 - uconfig, 142
 - udaily_discharge, 142
 - udefoutput, 142
 - udem, 142
 - udischarge, 142
 - ufacc, 142
 - ufdir, 143
 - ugeolut, 143
 - uhydrogeoclass, 143
 - ulai, 143
 - ulai_header, 143
 - ulaiclass, 143
 - ulailut, 143
 - ulcoverclass, 143
 - umeteo, 144
 - umeteo_header, 144
 - unamelist, 144
 - unamelist_param, 144
 - uopti, 144
 - uopti_nml, 144
 - uslope, 145
 - usoil_database, 145
 - usoilclass, 145
 - utws, 145
 - version, 145
 - version_date, 145
- mo_file.f90, 988
- mo_finish, 145
 - finish, 146
- mo_finish.f90, 991
- mo_global_variables, 147
 - basin, 151
 - basin_avg_tws_obs, 151
 - basin_avg_tws_sim, 151
 - c2tstu, 151
 - contact, 151
 - conventions, 152
 - dirabsvapppressure, 152
 - dircommonfiles, 152
 - dirconfigout, 152
 - direvapotranspiration, 152
 - dirgridded_lai, 152
 - dirlcover, 152
 - dirmaxtemperature, 152
 - dirminttemperature, 153
 - dirmorpho, 153
 - dirnetradiation, 153
 - dirneutrons, 153
 - dirout, 153
 - dirprecipitation, 153
 - dirreferenceet, 153

dirrestartin, 153
 dirrestartout, 154
 dirsoil_moisture, 154
 dirttemperature, 154
 dirwindspeed, 154
 evalper, 154
 evap_coeff, 154
 fday_pet, 154
 fday_prec, 154
 fday_temp, 155
 filelatlon, 155
 filelws, 155
 fnight_pet, 155
 fnight_prec, 155
 fnight_temp, 155
 fracsealed_cityarea, 155
 geounitkar, 155
 geounitlist, 155
 history, 156
 horizondepth_mhm, 156
 iflag_cordinate_sys, 156
 iflag_soildb, 156
 inputformat_gridded_lai, 156
 inputformat_meteo_forcings, 156
 IO_areacell, 156
 IO_asp, 156
 IO_basin, 157
 IO_cellcoor, 157
 IO_elev, 157
 IO_geounit, 157
 IO_gridded_lai, 157
 IO_id, 157
 IO_latitude, 157
 IO_lcover, 158
 IO_lcover_lai, 158
 IO_longitude, 158
 IO_mask, 158
 IO_ncells, 158
 IO_slope, 158
 IO_slope_emp, 158
 IO_soilid, 158
 l1_absvappress, 159
 l1_aeroresist, 159
 l1_aetcanopy, 159
 l1_aetsealed, 159
 l1_aetsoil, 159
 l1_alpha, 159
 l1_areacell, 159
 l1_baseflow, 159
 l1_cellcoor, 160
 l1_degday, 160
 l1_degdayinc, 160
 l1_degdaymax, 160
 l1_degdaynopre, 160
 l1_downbound_IO, 160
 l1_et, 160
 l1_et_mask, 161
 l1_fasp, 161
 l1_fastrunoff, 161
 l1_fforest, 161
 l1_fperm, 161
 l1_froots, 161
 l1_fsealed, 162
 l1_harsamcoeff, 162
 l1_id, 162
 l1_infilsoil, 162
 l1_inter, 162
 l1_jarvis_thresh_c1, 162
 l1_karstloss, 162
 l1_kbaseflow, 163
 l1_kfastflow, 163
 l1_kperco, 163
 l1_kslowflow, 163
 l1_latitude, 163
 l1_leftbound_IO, 163
 l1_longitude, 163
 l1_maxinter, 164
 l1_melt, 164
 l1_ncells, 164
 l1_netrad, 164
 l1_neutrons, 164
 l1_neutronsdata, 164
 l1_neutronsdata_mask, 164
 l1_ntcells_IO, 164
 l1_percol, 165
 l1_pet, 165
 l1_pet_calc, 165
 l1_pet_weights, 165
 l1_petlaicorfactor, 165
 l1_pre, 165
 l1_pre_weights, 165
 l1_preeffect, 165
 l1_prietayalpha, 166
 l1_rain, 166
 l1_rect_latitude, 166
 l1_rect_longitude, 166
 l1_rightbound_IO, 166
 l1_runoffseal, 166
 l1_satstw, 166
 l1_sealedthresh, 167
 l1_sealstw, 167
 l1_slowrunoff, 167
 l1_sm, 167
 l1_sm_mask, 167
 l1_snow, 167
 l1_snowpack, 167
 l1_soilmoist, 168
 l1_soilmoistexp, 168
 l1_soilmoistfc, 168
 l1_soilmoistsat, 168
 l1_surfresist, 168
 l1_temp, 168
 l1_temp_weights, 168
 l1_tempthresh, 169
 l1_throughfall, 169
 l1_tmax, 169

- l1_tmin, 169
- l1_total_runoff, 169
- l1_unsatstw, 169
- l1_unsatthresh, 169
- l1_upbound_l0, 170
- l1_wiltingpoint, 170
- l1_windspeed, 170
- lailut, 170
- laiunitlist, 170
- lats, 170
- lcfilename, 170
- lcyarid, 170
- level0, 171
- level1, 171
- level2, 171
- lons, 171
- mhm_details, 171
- nbasins, 171
- neutron_integral_afast, 171
- ngeounits, 172
- nlaiclass, 172
- nlcoverscene, 172
- nmeasperday_tws, 172
- nsoilhorizons_mhm, 172
- nsoilhorizons_sm_input, 172
- nsoiltypes, 172
- ntimesteps_l1_et, 172
- ntimesteps_l1_neutrons, 173
- ntimesteps_l1_sm, 173
- ntstepday, 173
- outputflxstate, 173
- perform_mpr, 173
- project_details, 173
- read_meteo_weights, 173
- read_restart, 173
- readper, 174
- resolutionhydrology, 174
- resolutionrouting, 174
- routingstates, 174
- setup_description, 174
- simper, 174
- simulation_type, 174
- soildb, 174
- tillagedepth, 175
- timestep, 175
- timestep_et_input, 175
- timestep_lai_input, 175
- timestep_model_inputs, 175
- timestep_model_outputs, 175
- timestep_neutrons_input, 175
- timestep_sm_input, 176
- warmingdays, 176
- warmper, 176
- write_restart, 176
- xcoor, 176
- ycoor, 176
- mo_global_variables.f90, 991
- mo_global_variables::basininfo, 791
- l0_iend, 792
- l0_iendmask, 792
- l0_istart, 792
- l0_istartmask, 792
- l0_mask, 792
- l1_iend, 792
- l1_iendmask, 793
- l1_istart, 793
- l1_istartmask, 793
- l1_mask, 793
- l2_iend, 793
- l2_iendmask, 793
- l2_istart, 793
- l2_istartmask, 793
- l2_mask, 793
- mo_global_variables::gridgeoref, 826
- cellsize, 826
- ncols, 826
- nodata_value, 826
- nrows, 826
- xllcorner, 827
- yllcorner, 827
- mo_global_variables::soiltype, 936
- clay, 937
- db, 937
- dbm, 937
- depth, 937
- id, 937
- is_present, 937
- ks, 937
- ld, 937
- nhorizons, 937
- ntillhorizons, 938
- rzdepth, 938
- sand, 938
- thetafc, 938
- thetafc_till, 938
- thetapw, 938
- thetapw_till, 938
- thetas, 938
- thetas_till, 938
- ud, 939
- wd, 939
- mo_global_variables::twsstructure, 955
- basinid, 955
- frame, 955
- tws, 955
- mo_init_states, 176
- calculate_grid_properties, 177
- get_basin_info, 178
- variables_alloc, 180
- variables_default_init, 181
- mo_init_states.f90, 995
- mo_julian, 183
- caldat, 185
- caldat360, 187
- caldat365, 188
- caldatjulian, 190

- calendar, 217
- date2dec, 192
- date2dec360, 194
- date2dec365, 195
- date2decjulian, 197
- dec2date, 199
- dec2date360, 201
- dec2date365, 203
- dec2datejulian, 205
- julday, 207
- julday360, 209
- julday365, 210
- juldayjulian, 211
- ndays, 212
- ndyin, 213
- selectcalendar, 214
- setcalendarinteger, 215
- setcalendarstring, 216
- mo_julian.f90, 995
- mo_julian::setcalendar, 934
 - setcalendarinteger, 934
 - setcalendarstring, 934
- mo_kind, 217
 - dp, 218
 - dpc, 218
 - i1, 218
 - i2, 218
 - i4, 219
 - i8, 219
 - lgt, 219
 - sp, 219
 - spc, 220
- mo_kind.f90, 996
- mo_kind::sprs2_dp, 946
 - irow, 947
 - jcol, 947
 - len, 947
 - n, 947
 - val, 948
- mo_kind::sprs2_sp, 948
 - irow, 948
 - jcol, 949
 - len, 949
 - n, 949
 - val, 949
- mo_linfit, 220
 - linfit_dp, 220
 - linfit_sp, 220
- mo_linfit.f90, 997
- mo_linfit::linfit, 840
 - linfit_dp, 841
 - linfit_sp, 841
- mo_mcmc, 221
 - generatenewparameterset_dp, 221
 - mcmc_dp, 222
 - mcmc_stddev_dp, 223
 - pargen_dp, 223
 - pargennorm_dp, 224
- mo_mcmc.f90, 997
- mo_mcmc::mcmc, 845
 - mcmc_dp, 849
- mo_mcmc::mcmc_stddev, 849
 - mcmc_stddev_dp, 853
- mo_message, 224
 - message, 225
 - message_text, 226
- mo_message.f90, 998
- mo_meteo_forcings, 226
 - chunk_config, 227
 - chunk_size, 228
 - is_read, 230
 - meteo_forcings_wrapper, 231
 - meteo_weights_wrapper, 234
 - prepare_meteo_forcings_data, 235
- mo_meteo_forcings.f90, 998
- mo_mhm, 237
 - mhm, 238
- mo_mhm.f90, 999
- mo_mhm_constants, 243
 - bulkdens_orgmatter, 245
 - c1_initstatesm, 245
 - cosmic_alpha, 245
 - cosmic_bd, 246
 - cosmic_l1, 246
 - cosmic_l2, 246
 - cosmic_l3, 246
 - cosmic_l4, 246
 - cosmic_n, 246
 - cosmic_vwclat, 246
 - dayhours, 247
 - daysecs, 247
 - desilets_a0, 247
 - desilets_a1, 247
 - desilets_a2, 247
 - duffiedelta1, 247
 - duffiedelta2, 247
 - duffiedr, 247
 - field_cap_c1, 248
 - field_cap_c2, 248
 - h2odens, 248
 - harsamconst, 248
 - karman, 248
 - ks_c, 248
 - lai_factor_surfresi, 248
 - lai_offset_surfresi, 249
 - max_surfresist, 249
 - maxgeounit, 249
 - maxnlcovers, 249
 - maxnobasins, 249
 - maxnosoilhorizons, 249
 - ncolpars, 249
 - nlcover_class, 249
 - nodata_dp, 250
 - nodata_i4, 250
 - noutflxstate, 250
 - p1_initstatefluxes, 250

- p2_initstatefluxes, 250
- p3_initstatefluxes, 250
- p4_initstatefluxes, 251
- p5_initstatefluxes, 251
- pwp_c, 251
- pwp_matpot_thetar, 251
- satpressureslope1, 251
- stboltzmann, 251
- tetens_c1, 251
- tetens_c2, 252
- tetens_c3, 252
- vgenuchten_sandtresh, 252
- vgenuchtenn_c1, 252
- vgenuchtenn_c10, 252
- vgenuchtenn_c11, 252
- vgenuchtenn_c12, 252
- vgenuchtenn_c13, 252
- vgenuchtenn_c14, 252
- vgenuchtenn_c15, 253
- vgenuchtenn_c16, 253
- vgenuchtenn_c17, 253
- vgenuchtenn_c18, 253
- vgenuchtenn_c2, 253
- vgenuchtenn_c3, 253
- vgenuchtenn_c4, 253
- vgenuchtenn_c5, 253
- vgenuchtenn_c6, 253
- vgenuchtenn_c7, 254
- vgenuchtenn_c8, 254
- vgenuchtenn_c9, 254
- windmeasheight, 254
- yeardays, 254
- yearmonths, 254
- yearmonths_i4, 254
- mo_mhm_constants.f90, 999
- mo_mhm_eval, 255
 - mhm_eval, 255
- mo_mhm_eval.f90, 1001
- mo_moment, 258
 - absdev_dp, 259
 - absdev_sp, 259
 - average_dp, 260
 - average_sp, 260
 - central_moment_dp, 260
 - central_moment_sp, 260
 - central_moment_var_dp, 260
 - central_moment_var_sp, 260
 - correlation_dp, 261
 - correlation_sp, 261
 - covariance_dp, 261
 - covariance_sp, 261
 - kurtosis_dp, 261
 - kurtosis_sp, 261
 - mean_dp, 261
 - mean_sp, 262
 - mixed_central_moment_dp, 262
 - mixed_central_moment_sp, 262
 - mixed_central_moment_var_dp, 262
 - mixed_central_moment_var_sp, 262
 - moment_dp, 263
 - moment_sp, 263
 - skewness_dp, 263
 - skewness_sp, 263
 - stddev_dp, 263
 - stddev_sp, 263
 - variance_dp, 264
 - variance_sp, 264
- mo_moment.f90, 1001
- mo_moment::absdev, 779
 - absdev_dp, 779
 - absdev_sp, 779
- mo_moment::average, 790
 - average_dp, 791
 - average_sp, 791
- mo_moment::central_moment, 799
 - central_moment_dp, 800
 - central_moment_sp, 800
- mo_moment::central_moment_var, 800
 - central_moment_var_dp, 800
 - central_moment_var_sp, 800
- mo_moment::correlation, 803
 - correlation_dp, 803
 - correlation_sp, 803
- mo_moment::covariance, 803
 - covariance_dp, 803
 - covariance_sp, 803
- mo_moment::kurtosis, 838
 - kurtosis_dp, 839
 - kurtosis_sp, 839
- mo_moment::mean, 853
 - mean_dp, 854
 - mean_sp, 854
- mo_moment::mixed_central_moment, 856
 - mixed_central_moment_dp, 856
 - mixed_central_moment_sp, 856
- mo_moment::mixed_central_moment_var, 856
 - mixed_central_moment_var_dp, 857
 - mixed_central_moment_var_sp, 857
- mo_moment::moment, 857
 - moment_dp, 857
 - moment_sp, 857
- mo_moment::skewness, 935
 - skewness_dp, 935
 - skewness_sp, 935
- mo_moment::stddev, 952
 - stddev_dp, 952
 - stddev_sp, 952
- mo_moment::variance, 972
 - variance_dp, 972
 - variance_sp, 972
- mo_mpr_pet, 264
 - bulksurface_resistance, 265
 - pet_correctbyasp, 266
 - pet_correctbylai, 268
 - priestley_taylor_alpha, 271
- mo_mpr_pet.f90, 1002

- mo_mpr_runoff, 272
 - mpr_runoff, 273
- mo_mpr_runoff.f90, 1003
- mo_mpr_smhorizons, 275
 - mpr_smhorizons, 276
- mo_mpr_smhorizons.f90, 1003
- mo_mpr_soilmoist, 279
 - field_cap, 280
 - genuchten, 281
 - hydro_cond, 283
 - mpr_sm, 284
 - pwp, 288
- mo_mpr_soilmoist.f90, 1003
- mo_mrm_constants, 289
 - deltah, 290
 - given_ts, 290
 - hoursecs, 290
 - maxnlcovers, 290
 - maxnobasins, 290
 - maxnogauges, 290
 - ncolpars, 290
 - nodata_dp, 291
 - nodata_i4, 291
 - noutflxstate, 291
 - nroutingstates, 291
 - p1_initstatefluxes, 291
 - rou_tspace_weight, 291
- mo_mrm_constants.f90, 1004
- mo_mrm_eval, 292
 - mrm_eval, 292
- mo_mrm_eval.f90, 1005
- mo_mrm_file, 294
 - file_config, 296
 - file_daily_discharge, 296
 - file_defoutput, 296
 - file_dem, 296
 - file_facc, 296
 - file_fdir, 297
 - file_gaugeloc, 297
 - file_main, 297
 - file_mrm_output, 297
 - file_namelist_mrm, 297
 - file_namelist_param_mrm, 297
 - file_opti, 297
 - file_opti_nml, 298
 - ncfile_discharge, 298
 - uconfig, 298
 - udaily_discharge, 298
 - udefoutput, 298
 - udem, 298
 - udischarge, 299
 - ufacc, 299
 - ufdir, 299
 - ugaugeloc, 299
 - ulcoverclass, 299
 - unamelist_mrm, 299
 - unamelist_param, 299
 - uopti, 300
 - uopti_nml, 300
 - version, 300
 - version_date, 300
- mo_mrm_file.f90, 1005
- mo_mrm_global_variables, 300
 - basin_mrm, 303
 - contact, 303
 - conventions, 303
 - dircommonfiles, 303
 - dirconfigout, 304
 - dirgauges, 304
 - dirlcover, 304
 - dirmorpho, 304
 - dirout, 304
 - dirrestartin, 304
 - dirrestartout, 304
 - dirtotalrunoff, 305
 - evalper, 305
 - filelatlon, 305
 - fracsealed_cityarea, 305
 - gauge, 305
 - history, 305
 - iflag_cordinate_sys, 305
 - inflowgauge, 306
 - is_start, 306
 - l0_areacell, 306
 - l0_basin, 306
 - l0_cellcoor, 306
 - l0_dracell, 306
 - l0_drasc, 306
 - l0_elev_mrm, 306
 - l0_elev_read, 307
 - l0_facc, 307
 - l0_fdir, 307
 - l0_floodplain, 307
 - l0_gaugeloc, 307
 - l0_id, 307
 - l0_inflowgaugeloc, 307
 - l0_l11_id, 307
 - l0_latitude, 308
 - l0_lcover_mrm, 308
 - l0_lcover_read, 308
 - l0_longitude, 308
 - l0_mask_mrm, 308
 - l0_ncells, 308
 - l0_streamnet, 308
 - l11_afloodplain, 308
 - l11_areacell, 309
 - l11_c1, 309
 - l11_c2, 309
 - l11_cellcoor, 309
 - l11_colout, 309
 - l11_downbound_l0, 309
 - l11_downbound_l1, 309
 - l11_fcol, 310
 - l11_fdir, 310
 - l11_fracfpimp, 310
 - l11_fromn, 310

- [l11_frow, 310](#)
- [l11_id, 310](#)
- [l11_k, 310](#)
- [l11_l1_id, 311](#)
- [l11_label, 311](#)
- [l11_leftbound_l0, 311](#)
- [l11_leftbound_l1, 311](#)
- [l11_length, 311](#)
- [l11_ncells, 311](#)
- [l11_netperm, 311](#)
- [l11_noutlets, 312](#)
- [l11_qmod, 312](#)
- [l11_qout, 312](#)
- [l11_qtin, 312](#)
- [l11_qtr, 312](#)
- [l11_rect_latitude, 312](#)
- [l11_rect_longitude, 312](#)
- [l11_rightbound_l0, 313](#)
- [l11_rightbound_l1, 313](#)
- [l11_rorder, 313](#)
- [l11_rowout, 313](#)
- [l11_sink, 313](#)
- [l11_slope, 313](#)
- [l11_tcol, 313](#)
- [l11_ton, 314](#)
- [l11_trow, 314](#)
- [l11_tsout, 314](#)
- [l11_upbound_l0, 314](#)
- [l11_upbound_l1, 314](#)
- [l11_xi, 314](#)
- [l1_areacell, 314](#)
- [l1_id, 315](#)
- [l1_l1_id, 315](#)
- [l1_ncells, 315](#)
- [l1_total_runoff_in, 315](#)
- [lcfilename, 315](#)
- [lcyarid, 315](#)
- [level0, 315](#)
- [level1, 316](#)
- [level11, 316](#)
- [level110, 316](#)
- [mhm_details, 316](#)
- [mrm_coupling_mode, 316](#)
- [mrm_runoff, 316](#)
- [nbasins, 316](#)
- [ngaigestotal, 317](#)
- [ninflowgaigestotal, 317](#)
- [nlcoverscene, 317](#)
- [nmeasperday, 317](#)
- [ntstepday, 317](#)
- [outputfixstate_mrm, 317](#)
- [perform_mpr, 317](#)
- [project_details, 318](#)
- [read_restart, 318](#)
- [readper, 318](#)
- [resolutionhydrology, 318](#)
- [resolutionrouting, 318](#)
- [setup_description, 318](#)
- [simper, 318](#)
- [simulation_type, 319](#)
- [timestep, 319](#)
- [timestep_model_inputs, 319](#)
- [timestep_model_outputs_mrm, 319](#)
- [warmingdays_mrm, 319](#)
- [warmper, 319](#)
- [write_restart, 319](#)
- [mo_mrm_global_variables.f90, 1006](#)
- [mo_mrm_global_variables::basininfo, 794](#)
 - [gaugeidlist, 795](#)
 - [gaugeindexlist, 795](#)
 - [gaugenodelist, 795](#)
 - [inflowgaugeheadwater, 795](#)
 - [inflowgaugeidlist, 795](#)
 - [inflowgaugeindexlist, 795](#)
 - [inflowgaugenodelist, 795](#)
 - [l0_coloutlet, 796](#)
 - [l0_iend, 796](#)
 - [l0_iendmask, 796](#)
 - [l0_istart, 796](#)
 - [l0_istartmask, 796](#)
 - [l0_mask, 796](#)
 - [l0_noutlet, 796](#)
 - [l0_rowoutlet, 796](#)
 - [l110_iend, 796](#)
 - [l110_istart, 797](#)
 - [l11_iend, 797](#)
 - [l11_iendmask, 797](#)
 - [l11_istart, 797](#)
 - [l11_istartmask, 797](#)
 - [l11_mask, 797](#)
 - [l1_iend, 797](#)
 - [l1_iendmask, 797](#)
 - [l1_istart, 797](#)
 - [l1_istartmask, 798](#)
 - [l1_mask, 798](#)
 - [ngauges, 798](#)
 - [ninflowgauges, 798](#)
- [mo_mrm_global_variables::gaugingstation, 815](#)
 - [basinid, 815](#)
 - [fname, 816](#)
 - [gaugeid, 816](#)
 - [q, 816](#)
- [mo_mrm_global_variables::gridgeoref, 827](#)
 - [cellsize, 828](#)
 - [ncols, 828](#)
 - [nodata_value, 828](#)
 - [nrows, 828](#)
 - [xllcorner, 828](#)
 - [yllcorner, 828](#)
- [mo_mrm_init, 320](#)
 - [config_output, 320](#)
 - [l0_check_input_routing, 321](#)
 - [mrm_init, 322](#)
 - [mrm_init_param, 324](#)
 - [mrm_update_param, 325](#)
 - [print_startup_message, 326](#)

- variables_alloc_routing, 327
- variables_default_init_routing, 327
- mo_mrm_init.f90, 1009
- mo_mrm_mpr, 328
 - reg_rout, 329
- mo_mrm_mpr.f90, 1009
- mo_mrm_net_startup, 330
 - celllength, 332
 - get_distance_two_lat_lon_points, 332
 - l11_flow_direction, 333
 - l11_fraction_sealed_floodplain, 335
 - l11_link_location, 336
 - l11_routing_order, 337
 - l11_set_drain_outlet_gauges, 338
 - l11_set_network_topology, 340
 - l11_stream_features, 341
 - l11_variable_init, 342
 - movedownonecell, 343
 - moveup, 344
- mo_mrm_net_startup.f90, 1009
- mo_mrm_objective_function_runoff, 344
 - eval, 346
 - extract_runoff, 348
 - loglikelihood, 350
 - loglikelihood_evin2013_2, 353
 - loglikelihood_stddev, 355
 - loglikelihood_trend_no_autocorr, 357
 - multi_objective_ae_fdc_lsv_nse_djf, 359
 - multi_objective_lnnse_highflow_lnnse_lowflow, 361
 - multi_objective_lnnse_highflow_lnnse_lowflow_2, 363
 - multi_objective_nse_lnnse, 366
 - multi_objective_runoff, 368
 - objective_equal_nse_lnnse, 369
 - objective_kge, 372
 - objective_lnnse, 374
 - objective_multiple_gauges_kge_power6, 376
 - objective_nse, 379
 - objective_power6_nse_lnnse, 381
 - objective_sse, 383
 - parameter_regularization, 385
 - single_objective_runoff, 385
- mo_mrm_objective_function_runoff.f90, 1010
- mo_mrm_read_config, 387
 - in_bound, 387
 - read_mrm_config, 388
 - read_mrm_config_coupling, 389
 - read_mrm_routing_params, 390
- mo_mrm_read_config.f90, 1011
- mo_mrm_read_data, 391
 - mrm_l0_variable_init, 392
 - mrm_l1_variable_init, 393
 - mrm_read_discharge, 394
 - mrm_read_l0_data, 395
 - mrm_read_total_runoff, 396
 - rotate_fdir_variable, 397
- mo_mrm_read_data.f90, 1012
- mo_mrm_read_latlon, 398
 - read_latlon, 398
- mo_mrm_read_latlon.f90, 1012
- mo_mrm_restart, 399
 - mrm_read_restart_config, 400
 - mrm_read_restart_states, 401
 - mrm_write_restart, 403
- mo_mrm_restart.f90, 1012
- mo_mrm_routing, 404
 - add_inflow, 405
 - l11_routing, 406
 - l11_runoff_acc, 408
 - mrm_routing, 409
- mo_mrm_routing.f90, 1013
- mo_mrm_signatures, 413
 - autocorrelation, 414
 - flowdurationcurve, 415
 - limb_densities, 416
 - maximummonthlyflow, 417
 - moments, 418
 - peakdistribution, 420
 - runoffratio, 421
 - zeroflowratio, 422
- mo_mrm_signatures.f90, 1013
- mo_mrm_tools, 423
 - calculate_grid_properties, 423
 - get_basin_info_mrm, 425
- mo_mrm_tools.f90, 1014
- mo_mrm_write, 427
 - average_counter, 435
 - day_counter, 436
 - month_counter, 436
 - mrm_write, 428
 - mrm_write_optifile, 429
 - mrm_write_optinamelist, 430
 - mrm_write_output_fluxes, 431
 - nc, 436
 - write_configfile, 433
 - write_daily_obs_sim_discharge, 434
 - year_counter, 436
- mo_mrm_write.f90, 1014
- mo_mrm_write_fluxes_states, 436
 - close, 437
 - createoutputfile, 438
 - fluxesunit, 439
 - geocoordinates, 440
 - mapcoordinates, 440
 - newoutputdataset, 441
 - newoutputvariable, 442
 - updatedataset, 443
 - updatevariable, 443
 - writetimestep, 444
 - writevariableattributes, 444
 - writevariabletimestep, 445
- mo_mrm_write_fluxes_states.f90, 1015
- mo_mrm_write_fluxes_states::outputdataset, 903
 - all, 904
 - basin, 904

- close, 904
- count, 904
- counter, 905
- created, 905
- ibasin, 905
- id, 905
- nc, 905
- ncdataset, 905
- steps, 905
- store, 905
- time, 905
- to, 906
- updateddataset, 904
- variables, 906
- vars, 906
- write, 906
- writetimestep, 904
- written, 906
- mo_mrm_write_fluxes_states::outputvariable, 912
 - average, 913
 - avg, 913
 - before, 913
 - between, 914
 - calls, 914
 - contains, 914
 - count, 914
 - counter, 914
 - data, 914
 - mask, 914
 - nc, 914
 - ncdataset, 915
 - number, 915
 - of, 915
 - reconstruct, 915
 - store, 915
 - the, 915
 - to, 915
 - updatevariable, 913, 916
 - variable, 916
 - which, 916
 - writes, 916
 - writevariabletimestep, 913
 - writing, 916
- mo_multi_param_reg, 446
 - aerodynamical_resistance, 447
 - baseflow_param, 449
 - canopy_intercept_param, 450
 - iper_thres_runoff, 452
 - karstic_layer, 453
 - mpr, 456
 - snow_acc_melt_param, 461
- mo_multi_param_reg.f90, 1016
- mo_ncread, 463
 - check, 464
 - get_info, 465
 - get_ncdim, 467
 - get_ncdimatt, 468
 - get_ncvar_0d_dp, 469
 - get_ncvar_0d_i1, 469
 - get_ncvar_0d_i4, 470
 - get_ncvar_0d_sp, 470
 - get_ncvar_1d_dp, 471
 - get_ncvar_1d_i1, 471
 - get_ncvar_1d_i4, 472
 - get_ncvar_1d_sp, 472
 - get_ncvar_2d_dp, 473
 - get_ncvar_2d_i1, 473
 - get_ncvar_2d_i4, 474
 - get_ncvar_2d_sp, 474
 - get_ncvar_3d_dp, 475
 - get_ncvar_3d_i1, 475
 - get_ncvar_3d_i4, 476
 - get_ncvar_3d_sp, 476
 - get_ncvar_4d_dp, 477
 - get_ncvar_4d_i1, 477
 - get_ncvar_4d_i4, 478
 - get_ncvar_4d_sp, 478
 - get_ncvar_5d_dp, 479
 - get_ncvar_5d_i1, 479
 - get_ncvar_5d_i4, 480
 - get_ncvar_5d_sp, 480
 - get_ncvaratt, 481
 - ncclose, 482
 - ncopen, 482
- mo_ncread.f90, 1016
- mo_ncread::get_ncvar, 817
 - get_ncvar_0d_dp, 818
 - get_ncvar_0d_i1, 818
 - get_ncvar_0d_i4, 818
 - get_ncvar_0d_sp, 818
 - get_ncvar_1d_dp, 818
 - get_ncvar_1d_i1, 818
 - get_ncvar_1d_i4, 819
 - get_ncvar_1d_sp, 819
 - get_ncvar_2d_dp, 819
 - get_ncvar_2d_i1, 819
 - get_ncvar_2d_i4, 820
 - get_ncvar_2d_sp, 820
 - get_ncvar_3d_dp, 820
 - get_ncvar_3d_i1, 820
 - get_ncvar_3d_i4, 820
 - get_ncvar_3d_sp, 821
 - get_ncvar_4d_dp, 821
 - get_ncvar_4d_i1, 821
 - get_ncvar_4d_i4, 821
 - get_ncvar_4d_sp, 821
 - get_ncvar_5d_dp, 822
 - get_ncvar_5d_i1, 822
 - get_ncvar_5d_i4, 822
 - get_ncvar_5d_sp, 822
- mo_ncwrite, 483
 - check, 484
 - close_netcdf, 485
 - create_netcdf, 487
 - dnc, 508
 - dump_netcdf_1d_dp, 488

- dump_netcdf_1d_i4, [488](#)
- dump_netcdf_1d_sp, [489](#)
- dump_netcdf_2d_dp, [489](#)
- dump_netcdf_2d_i4, [490](#)
- dump_netcdf_2d_sp, [490](#)
- dump_netcdf_3d_dp, [491](#)
- dump_netcdf_3d_i4, [491](#)
- dump_netcdf_3d_sp, [492](#)
- dump_netcdf_4d_dp, [492](#)
- dump_netcdf_4d_i4, [493](#)
- dump_netcdf_4d_sp, [493](#)
- dump_netcdf_5d_dp, [494](#)
- dump_netcdf_5d_i4, [494](#)
- dump_netcdf_5d_sp, [495](#)
- gatt, [508](#)
- maxlen, [508](#)
- nattdim, [509](#)
- ndims, [509](#)
- ngatt, [509](#)
- nmaxatt, [509](#)
- nmaxdim, [509](#)
- nvars, [509](#)
- open_netcdf, [495](#)
- v, [509](#)
- var2nc_1d_dp, [497](#)
- var2nc_1d_i4, [498](#)
- var2nc_1d_sp, [499](#)
- var2nc_2d_dp, [499](#)
- var2nc_2d_i4, [500](#)
- var2nc_2d_sp, [501](#)
- var2nc_3d_dp, [501](#)
- var2nc_3d_i4, [502](#)
- var2nc_3d_sp, [503](#)
- var2nc_4d_dp, [503](#)
- var2nc_4d_i4, [504](#)
- var2nc_4d_sp, [505](#)
- var2nc_5d_dp, [505](#)
- var2nc_5d_i4, [506](#)
- var2nc_5d_sp, [507](#)
- write_dynamic_netcdf, [507](#)
- write_static_netcdf, [508](#)
- mo_ncwrite.f90, [1017](#)
- mo_ncwrite::attribute, [788](#)
 - name, [788](#)
 - nvalues, [788](#)
 - values, [788](#)
 - xtype, [788](#)
- mo_ncwrite::dims, [807](#)
 - dimid, [807](#)
 - len, [807](#)
 - name, [807](#)
- mo_ncwrite::dump_netcdf, [808](#)
 - dump_netcdf_1d_dp, [808](#)
 - dump_netcdf_1d_i4, [808](#)
 - dump_netcdf_1d_sp, [808](#)
 - dump_netcdf_2d_dp, [809](#)
 - dump_netcdf_2d_i4, [809](#)
 - dump_netcdf_2d_sp, [809](#)
 - dump_netcdf_3d_dp, [809](#)
 - dump_netcdf_3d_i4, [810](#)
 - dump_netcdf_3d_sp, [810](#)
 - dump_netcdf_4d_dp, [810](#)
 - dump_netcdf_4d_i4, [810](#)
 - dump_netcdf_4d_sp, [810](#)
 - dump_netcdf_5d_dp, [811](#)
 - dump_netcdf_5d_i4, [811](#)
 - dump_netcdf_5d_sp, [811](#)
- mo_ncwrite::var2nc, [960](#)
 - var2nc_1d_dp, [961](#)
 - var2nc_1d_i4, [962](#)
 - var2nc_1d_sp, [962](#)
 - var2nc_2d_dp, [962](#)
 - var2nc_2d_i4, [963](#)
 - var2nc_2d_sp, [963](#)
 - var2nc_3d_dp, [963](#)
 - var2nc_3d_i4, [964](#)
 - var2nc_3d_sp, [964](#)
 - var2nc_4d_dp, [964](#)
 - var2nc_4d_i4, [965](#)
 - var2nc_4d_sp, [965](#)
 - var2nc_5d_dp, [965](#)
 - var2nc_5d_i4, [966](#)
 - var2nc_5d_sp, [966](#)
- mo_ncwrite::variable, [967](#)
 - att, [968](#)
 - count, [968](#)
 - dimids, [968](#)
 - dimtypes, [968](#)
 - g0_b, [969](#)
 - g0_d, [969](#)
 - g0_f, [969](#)
 - g0_i, [969](#)
 - g1_b, [969](#)
 - g1_d, [969](#)
 - g1_f, [969](#)
 - g1_i, [969](#)
 - g2_b, [969](#)
 - g2_d, [970](#)
 - g2_f, [970](#)
 - g2_i, [970](#)
 - g3_b, [970](#)
 - g3_d, [970](#)
 - g3_f, [970](#)
 - g3_i, [970](#)
 - g4_b, [970](#)
 - g4_d, [970](#)
 - g4_f, [971](#)
 - g4_i, [971](#)
 - name, [971](#)
 - natt, [971](#)
 - ndims, [971](#)
 - nlvls, [971](#)
 - nsubs, [971](#)
 - start, [971](#)
 - unlimited, [971](#)
 - varid, [972](#)

- wflag, 972
- xtype, 972
- mo_netcdf, 510
 - check, 513
 - close, 513
 - equalncdimensions, 513
 - getdata1df32, 514
 - getdata1df64, 514
 - getdata1di16, 515
 - getdata1di32, 515
 - getdata1di8, 516
 - getdata2df32, 516
 - getdata2df64, 517
 - getdata2di16, 517
 - getdata2di32, 518
 - getdata2di8, 518
 - getdata3df32, 519
 - getdata3df64, 519
 - getdata3di16, 520
 - getdata3di32, 520
 - getdata3di8, 521
 - getdata4df32, 521
 - getdata4df64, 522
 - getdata4di16, 522
 - getdata4di32, 523
 - getdata4di8, 523
 - getdata5df32, 524
 - getdata5df64, 524
 - getdata5di16, 525
 - getdata5di32, 525
 - getdata5di8, 526
 - getdatascalarf32, 526
 - getdatascalarf64, 527
 - getdatascalar16, 527
 - getdatascalar32, 528
 - getdatascalar8, 528
 - getdimensionbyid, 529
 - getdimensionbyname, 529
 - getdimensionlength, 529
 - getdimensionname, 530
 - getdtypefrominteger, 530
 - getdtypefromstring, 531
 - getglobalattributechar, 531
 - getglobalattributef32, 531
 - getglobalattributef64, 532
 - getglobalattributei16, 532
 - getglobalattributei32, 533
 - getglobalattributei8, 533
 - getnodimensions, 534
 - getnovariables, 534
 - getreaddatashape, 534
 - getunlimiteddimension, 535
 - getvariableattributechar, 536
 - getvariableattributef32, 536
 - getvariableattributef64, 537
 - getvariableattributei16, 537
 - getvariableattributei32, 537
 - getvariableattributei8, 538
 - getvariablebyname, 538
 - getvariabledimensions, 539
 - getvariabledtype, 539
 - getvariablefillvaluef32, 539
 - getvariablefillvaluef64, 539
 - getvariablefillvaluei16, 540
 - getvariablefillvaluei32, 540
 - getvariablefillvaluei8, 540
 - getvariableids, 540
 - getvariablename, 540
 - getvariables, 541
 - getvariablesshape, 541
 - hasattribute, 541
 - hasdimension, 541
 - hasvariable, 541
 - initncdataset, 541
 - initncdimension, 542
 - initncvariable, 542
 - isdatasetunlimited, 542
 - isunlimiteddimension, 543
 - isunlimitedvariable, 543
 - newncdataset, 543
 - newncdimension, 543
 - newncvariable, 543
 - setdata1df32, 543
 - setdata1df64, 544
 - setdata1di16, 544
 - setdata1di32, 545
 - setdata1di8, 545
 - setdata2df32, 546
 - setdata2df64, 546
 - setdata2di16, 547
 - setdata2di32, 547
 - setdata2di8, 548
 - setdata3df32, 548
 - setdata3df64, 549
 - setdata3di16, 549
 - setdata3di32, 550
 - setdata3di8, 550
 - setdata4df32, 551
 - setdata4df64, 551
 - setdata4di16, 552
 - setdata4di32, 552
 - setdata4di8, 553
 - setdata5df32, 553
 - setdata5df64, 554
 - setdata5di16, 554
 - setdata5di32, 555
 - setdata5di8, 555
 - setdatascalarf32, 556
 - setdatascalarf64, 556
 - setdatascalar16, 557
 - setdatascalar32, 557
 - setdatascalar8, 557
 - setdimension, 558
 - setglobalattributechar, 558
 - setglobalattributef32, 559
 - setglobalattributef64, 559

- setglobalattributei16, 559
- setglobalattributei32, 560
- setglobalattributei8, 560
- setvariableattributechar, 561
- setvariableattributef32, 561
- setvariableattributef64, 561
- setvariableattributei16, 562
- setvariableattributei32, 562
- setvariableattributei8, 563
- setvariablefillvaluef32, 563
- setvariablefillvaluef64, 563
- setvariablefillvaluei16, 563
- setvariablefillvaluei32, 563
- setvariablefillvaluei8, 563
- setvariablewithids, 564
- setvariablewithnames, 565
- setvariablewithtypes, 565
- mo_netcdf.f90, 1019
- mo_netcdf::ncdataset, 862
 - close, 864
 - dataset, 872
 - file, 872
 - filename, 872
 - fname, 872
 - getattribute, 864
 - getdimension, 865
 - getdimensionbyid, 865
 - getdimensionbyname, 865
 - getglobalattributechar, 866
 - getglobalattributef32, 866
 - getglobalattributef64, 866
 - getglobalattributei16, 866
 - getglobalattributei32, 866
 - getglobalattributei8, 866
 - getnovariables, 866
 - getunlimiteddimension, 866
 - getvariable, 867
 - getvariablebyname, 867
 - getvariableids, 867
 - getvariables, 867
 - hasdimension, 868
 - hasvariable, 868
 - id, 872
 - initncdataset, 868
 - isunlimited, 869
 - mode, 872
 - netcdf, 872
 - of, 872
 - open, 872
 - opened, 873
 - setattribute, 869
 - setdimension, 869
 - setglobalattributechar, 870
 - setglobalattributef32, 870
 - setglobalattributef64, 870
 - setglobalattributei16, 870
 - setglobalattributei32, 870
 - setglobalattributei8, 870
 - setvariable, 870
 - setvariablewithids, 871
 - setvariablewithnames, 871
 - setvariablewithtypes, 871
 - the, 873
- mo_netcdf::ncdimension, 873
 - dimension, 889
 - getattribute, 876
 - getdata, 877
 - getdata1df32, 877
 - getdata1df64, 877
 - getdata1di16, 877
 - getdata1di32, 877
 - getdata1di8, 877
 - getdata2df32, 878
 - getdata2df64, 878
 - getdata2di16, 878
 - getdata2di32, 878
 - getdata2di8, 878
 - getdata3df32, 878
 - getdata3df64, 878
 - getdata3di16, 878
 - getdata3di32, 878
 - getdata3di8, 879
 - getdata4df32, 879
 - getdata4df64, 879
 - getdata4di16, 879
 - getdata4di32, 879
 - getdata4di8, 879
 - getdata5df32, 879
 - getdata5df64, 879
 - getdata5di16, 879
 - getdata5di32, 880
 - getdata5di8, 880
 - getdatascalarf32, 880
 - getdatascalarf64, 880
 - getdatascalar16, 880
 - getdatascalar32, 880
 - getdatascalar8, 880
 - getdimensions, 880
 - getdtype, 880
 - getfillvalue, 881
 - getname, 881
 - getnodimensions, 881
 - getshape, 881
 - getvariableattributechar, 881
 - getvariableattributef32, 881
 - getvariableattributef64, 882
 - getvariableattributei16, 882
 - getvariableattributei32, 882
 - getvariableattributei8, 882
 - getvariablefillvaluef32, 882
 - getvariablefillvaluef64, 882
 - getvariablefillvaluei16, 882
 - getvariablefillvaluei32, 882
 - getvariablefillvaluei8, 882
 - hasattribute, 883
 - id, 889

- initncvariable, 883
- isunlimited, 883
- netcdf, 889
- parent, 889
- s, 890
- setattribute, 883
- setdata, 884
- setdata1df32, 884
- setdata1df64, 884
- setdata1di16, 884
- setdata1di32, 884
- setdata1di8, 884
- setdata2df32, 885
- setdata2df64, 885
- setdata2di16, 885
- setdata2di32, 885
- setdata2di8, 885
- setdata3df32, 885
- setdata3df64, 885
- setdata3di16, 885
- setdata3di32, 885
- setdata3di8, 886
- setdata4df32, 886
- setdata4df64, 886
- setdata4di16, 886
- setdata4di32, 886
- setdata4di8, 886
- setdata5df32, 886
- setdata5df64, 886
- setdata5di16, 886
- setdata5di32, 887
- setdata5di8, 887
- setdatascalarf32, 887
- setdatascalarf64, 887
- setdatascalar16, 887
- setdatascalar32, 887
- setdatascalar8, 887
- setfillvalue, 887
- setvariableattributechar, 888
- setvariableattributef32, 888
- setvariableattributef64, 888
- setvariableattributei16, 888
- setvariableattributei32, 888
- setvariableattributei8, 888
- setvariablefillvaluef32, 888
- setvariablefillvaluef64, 889
- setvariablefillvaluei16, 889
- setvariablefillvaluei32, 889
- setvariablefillvaluei8, 889
- the, 890
- mo_netcdf::ncvariable, 890
- mo_neutrons, 566
 - approx_mon_int, 566
 - approx_mon_int_eps, 567
 - approx_mon_int_steps, 568
 - cosmic, 568
 - desiletsn0, 570
 - intgrandfast, 571
 - lookupintegral, 572
 - oldintegration, 573
 - tabularintegralafast, 573
- mo_neutrons.f90, 1021
- mo_nml, 574
 - close_nml, 575
 - length_error, 579
 - missing, 579
 - nunitnml, 579
 - open_nml, 576
 - position_nml, 577
 - positioned, 579
 - read_error, 579
- mo_nml.f90, 1022
- mo_objective_function, 579
 - extract_basin_avg_tws, 580
 - objective, 582
 - objective_et_kge_catchment_avg, 584
 - objective_kge_q_et, 587
 - objective_kge_q_rmse_et, 589
 - objective_kge_q_rmse_tws, 591
 - objective_kge_q_sm_corr, 593
 - objective_neutrons_kge_catchment_avg, 595
 - objective_sm_corr, 598
 - objective_sm_kge_catchment_avg, 601
 - objective_sm_pd, 604
 - objective_sm_sse_standard_score, 607
- mo_objective_function.f90, 1023
- mo_optimization, 610
 - optimization, 610
- mo_optimization.f90, 1023
- mo_orderpack, 613
 - d_ctrper, 615
 - d_fndnth, 615
 - d_indmed, 615
 - d_indnth, 616
 - d_inspar, 616
 - d_inssor, 616
 - d_med, 616
 - d_median, 617
 - d_mrgref, 617
 - d_mrgnrk, 617
 - d_mulcnt, 617
 - d_nearless, 617
 - d_rapknr, 617
 - d_refpar, 618
 - d_refsor, 618
 - d_rinpar, 618
 - d_rnkpar, 618
 - d_subsor, 618
 - d_uniinv, 619
 - d_unipar, 619
 - d_unirnk, 619
 - d_unista, 619
 - d_valmed, 619
 - d_valnth, 620
 - i_ctrper, 620
 - i_fndnth, 620

- i_indmed, 620
- i_indnth, 620
- i_inspar, 621
- i_inssor, 621
- i_med, 621
- i_median, 621
- i_mrgref, 622
- i_mrgrnk, 622
- i_mulcnt, 622
- i_nearless, 622
- i_rapknr, 622
- i_refpar, 622
- i_refsor, 622
- i_rinpar, 623
- i_rnkpar, 623
- i_subsor, 623
- i_uniinv, 624
- i_unipar, 624
- i_unirnk, 624
- i_unista, 624
- i_valmed, 624
- i_valnth, 624
- idont, 630
- r_ctrper, 625
- r_fndnth, 625
- r_indmed, 625
- r_indnth, 625
- r_inspar, 625
- r_inssor, 626
- r_med, 626
- r_median, 626
- r_mrgref, 626
- r_mrgrnk, 627
- r_mulcnt, 627
- r_nearless, 627
- r_rapknr, 627
- r_refpar, 627
- r_refsor, 627
- r_rinpar, 628
- r_rnkpar, 628
- r_subsor, 628
- r_uniinv, 628
- r_unipar, 629
- r_unirnk, 629
- r_unista, 629
- r_valmed, 629
- r_valnth, 629
- sort_index_dp, 629
- sort_index_i4, 629
- sort_index_sp, 630
- mo_orderpack.f90, 1024
- mo_orderpack::ctrper, 805
 - d_ctrper, 805
 - i_ctrper, 805
 - r_ctrper, 805
- mo_orderpack::fndnth, 813
 - d_fndnth, 813
 - i_fndnth, 813
 - r_fndnth, 813
- mo_orderpack::indmed, 829
 - d_indmed, 830
 - i_indmed, 830
 - r_indmed, 830
- mo_orderpack::indnth, 830
 - d_indnth, 830
 - i_indnth, 830
 - r_indnth, 831
- mo_orderpack::inspar, 831
 - d_inspar, 831
 - i_inspar, 831
 - r_inspar, 831
- mo_orderpack::inssor, 832
 - d_inssor, 832
 - i_inssor, 832
 - r_inssor, 832
- mo_orderpack::mrgref, 858
 - d_mrgref, 858
 - i_mrgref, 858
 - r_mrgref, 858
- mo_orderpack::mrgrnk, 859
 - d_mrgrnk, 859
 - i_mrgrnk, 859
 - r_mrgrnk, 859
- mo_orderpack::mulcnt, 861
 - d_mulcnt, 861
 - i_mulcnt, 861
 - r_mulcnt, 861
- mo_orderpack::nearless, 891
 - d_nearless, 891
 - i_nearless, 891
 - r_nearless, 891
- mo_orderpack::omedian, 898
 - d_median, 898
 - i_median, 898
 - r_median, 898
- mo_orderpack::rapknr, 925
 - d_rapknr, 925
 - i_rapknr, 926
 - r_rapknr, 926
- mo_orderpack::refpar, 928
 - d_refpar, 928
 - i_refpar, 928
 - r_refpar, 929
- mo_orderpack::refsor, 929
 - d_refsor, 929
 - i_refsor, 929
 - r_refsor, 929
- mo_orderpack::rinpar, 929
 - d_rinpar, 930
 - i_rinpar, 930
 - r_rinpar, 930
- mo_orderpack::rnkpar, 932
 - d_rnkpar, 932
 - i_rnkpar, 932
 - r_rnkpar, 932
- mo_orderpack::sort, 939

- d_refsor, 941
- i_refsor, 942
- r_refsor, 942
- mo_orderpack::sort_index, 942
 - sort_index_dp, 942
 - sort_index_i4, 942
 - sort_index_sp, 942
- mo_orderpack::uniinv, 956
 - d_uniinv, 956
 - i_uniinv, 956
 - r_uniinv, 956
- mo_orderpack::unipar, 956
 - d_unipar, 956
 - i_unipar, 957
 - r_unipar, 957
- mo_orderpack::unirnk, 957
 - d_unirnk, 957
 - i_unirnk, 957
 - r_unirnk, 958
- mo_orderpack::unista, 958
 - d_unista, 958
 - i_unista, 958
 - r_unista, 958
- mo_orderpack::valmed, 959
 - d_valmed, 959
 - i_valmed, 959
 - r_valmed, 959
- mo_orderpack::valnth, 959
 - d_valnth, 959
 - i_valnth, 959
 - r_valnth, 960
- mo_percentile, 630
 - median_dp, 630
 - median_sp, 631
 - n_element_dp, 631
 - n_element_sp, 631
 - percentile_0d_dp, 631
 - percentile_0d_sp, 631
 - percentile_1d_dp, 632
 - percentile_1d_sp, 632
 - qmedian_dp, 632
 - qmedian_sp, 632
- mo_percentile.f90, 1026
- mo_percentile::median, 855
 - median_dp, 855
 - median_sp, 855
- mo_percentile::n_element, 861
 - n_element_dp, 861
 - n_element_sp, 862
- mo_percentile::percentile, 922
 - percentile_0d_dp, 922
 - percentile_0d_sp, 922
 - percentile_1d_dp, 922
 - percentile_1d_sp, 923
- mo_percentile::qmedian, 925
 - qmedian_dp, 925
 - qmedian_sp, 925
- mo_pet, 632
 - extraterr_rad_approx, 633
 - pet_hargreaves, 635
 - pet_penman, 637
 - pet_priestly, 639
 - sat_vap_pressure, 640
 - slope_satpressure, 642
- mo_pet.f90, 1026
- mo_prepare_gridded_lai, 644
 - prepare_gridded_daily_lai_data, 645
 - prepare_gridded_mean_monthly_lai_data, 646
- mo_prepare_gridded_lai.f90, 1027
- mo_read_config, 647
 - in_bound, 647
 - read_config, 648
- mo_read_config.f90, 1027
- mo_read_forcing_nc, 649
 - get_time, 650
 - read_forcing_nc, 651
 - read_weights_nc, 654
- mo_read_forcing_nc.f90, 1027
- mo_read_latlon, 656
 - read_latlon, 656
- mo_read_latlon.f90, 1028
- mo_read_lut, 658
 - read_geoformation_lut, 658
 - read_lai_lut, 659
- mo_read_lut.f90, 1028
- mo_read_meteo, 660
 - read_meteo_bin, 661
- mo_read_meteo.f90, 1028
- mo_read_optional_data, 662
 - read_basin_avg_tws, 663
 - read_evapotranspiration, 664
 - read_neutrons, 665
 - read_soil_moisture, 666
- mo_read_optional_data.f90, 1029
- mo_read_spatial_data, 667
 - read_header_ascii, 668
 - read_spatial_data_ascii_dp, 669
 - read_spatial_data_ascii_i4, 670
- mo_read_spatial_data.f90, 1029
- mo_read_spatial_data::read_spatial_data_ascii, 926
 - read_spatial_data_ascii_dp, 927
 - read_spatial_data_ascii_i4, 927
- mo_read_timeseries, 670
 - read_timeseries, 671
- mo_read_timeseries.f90, 1029
- mo_read_wrapper, 672
 - check_consistency_lut_map, 673
 - read_data, 674
- mo_read_wrapper.f90, 1030
- mo_restart, 675
 - read_restart_config, 676
 - read_restart_states, 677
 - write_restart_files, 679
- mo_restart.f90, 1030
- mo_runoff, 681
 - l1_total_runoff, 681

- runoff_sat_zone, 682
- runoff_unsat_zone, 684
- mo_runoff.f90, 1030
- mo_sce, 686
 - cce, 687
 - chkcst, 688
 - comp, 688
 - getpnt, 689
 - parstt, 689
 - sce, 690
 - sort_matrix, 694
- mo_sce.f90, 1031
 - set_optional, 1031
 - write_best_final, 1032
 - write_best_intermediate, 1032
 - write_population, 1032
 - write_termination_case, 1033
- mo_set_netcdf_outputs, 695
 - set_netcdf, 695
- mo_set_netcdf_outputs.f90, 1033
- mo_snow_accum_melt, 696
 - snow_accum_melt, 696
- mo_snow_accum_melt.f90, 1033
- mo_soil_database, 698
 - generate_soil_database, 699
 - read_soil_lut, 700
- mo_soil_database.f90, 1034
- mo_soil_moisture, 701
 - feddes_et_reduction, 702
 - jarvis_et_reduction, 703
 - soil_moisture, 705
- mo_soil_moisture.f90, 1034
- mo_spatial_agg_disagg_forcing, 708
 - spatial_aggregation_3d, 709
 - spatial_aggregation_4d, 709
 - spatial_disaggregation_3d, 709
 - spatial_disaggregation_4d, 710
- mo_spatial_agg_disagg_forcing.f90, 1035
- mo_spatial_agg_disagg_forcing::spatial_aggregation, 943
 - spatial_aggregation_3d, 943
 - spatial_aggregation_4d, 943
- mo_spatial_agg_disagg_forcing::spatial_disaggregation, 944
 - spatial_disaggregation_3d, 944
 - spatial_disaggregation_4d, 945
- mo_spatialsimilarity, 710
 - nndv_dp, 711
 - nndv_sp, 711
 - pd_dp, 711
 - pd_sp, 711
- mo_spatialsimilarity.f90, 1035
- mo_spatialsimilarity::nndv, 891
 - nndv_dp, 893
 - nndv_sp, 893
- mo_spatialsimilarity::pd, 920
 - pd_dp, 921
 - pd_sp, 922
- mo_standard_score, 711
 - classified_standard_score_dp, 712
 - classified_standard_score_sp, 712
 - standard_score_dp, 712
 - standard_score_sp, 712
- mo_standard_score.f90, 1036
- mo_standard_score::classified_standard_score, 801
 - classified_standard_score_dp, 801
 - classified_standard_score_sp, 802
- mo_standard_score::standard_score, 950
 - standard_score_dp, 951
 - standard_score_sp, 951
- mo_startup, 713
 - constants_init, 713
 - initialise, 714
 - l0_check_input, 716
 - l0_variable_init, 717
 - l1_variable_init, 718
 - l2_variable_init, 719
- mo_startup.f90, 1036
- mo_string_utils, 720
 - compress, 721
 - divide_string, 722
 - dp2str, 723
 - equalstrings, 723
 - i42str, 724
 - i4array2str, 724
 - i82str, 724
 - log2str, 724
 - nonnull, 724
 - separator, 727
 - sp2str, 725
 - splitstring, 725
 - startswith, 725
 - str2num, 726
 - tolower, 726
 - toupper, 727
- mo_string_utils.f90, 1037
- mo_string_utils::num2str, 895
 - dp2str, 896
 - i42str, 896
 - i82str, 896
 - log2str, 896
 - sp2str, 897
- mo_string_utils::numarray2str, 897
 - i4array2str, 897
- mo_template, 728
 - circum, 728
 - itest, 729
 - mean_dp, 729
 - mean_sp, 729
 - pi_dp, 729
 - pi_sp, 730
- mo_template.f90, 1037
- mo_template::mean, 854
 - mean_dp, 855
 - mean_sp, 855
- mo_temporal_aggregation, 730

- day2mon_average_dp, 730
- hour2day_average_dp, 731
- mo_temporal_aggregation.f90, 1038
- mo_temporal_aggregation::day2mon_average, 806
 - day2mon_average_dp, 806
- mo_temporal_aggregation::hour2day_average, 828
 - hour2day_average_dp, 829
- mo_temporal_disagg_forcing, 731
 - temporal_disagg_forcing, 732
- mo_temporal_disagg_forcing.f90, 1038
- mo_timer, 734
 - clock_rate, 742
 - cputime, 742
 - cycles1, 742
 - cycles2, 743
 - cycles_max, 743
 - max_timers, 743
 - status, 743
 - timer_check, 735
 - timer_clear, 736
 - timer_get, 736
 - timer_print, 738
 - timer_start, 739
 - timer_stop, 740
 - timers_init, 741
- mo_timer.f90, 1039
- mo_upscaling_operators, 743
 - l0_fractionalcover_in_lx, 744
 - majority_statistics, 745
 - upscale_arithmetic_mean, 746
 - upscale_geometric_mean, 749
 - upscale_harmonic_mean, 750
- mo_upscaling_operators.f90, 1039
- mo_utils, 751
 - equal_dp, 753
 - equal_sp, 753
 - greaterequal_dp, 753
 - greaterequal_sp, 753
 - is_finite_dp, 753
 - is_finite_sp, 753
 - is_nan_dp, 754
 - is_nan_sp, 754
 - is_normal_dp, 754
 - is_normal_sp, 754
 - lesserequal_dp, 754
 - lesserequal_sp, 754
 - locate_0d_dp, 754
 - locate_0d_sp, 755
 - locate_1d_dp, 755
 - locate_1d_sp, 755
 - notequal_dp, 755
 - notequal_sp, 755
 - special_value_dp, 755
 - special_value_sp, 756
 - swap_vec_dp, 756
 - swap_vec_i4, 756
 - swap_vec_sp, 756
 - swap_xy_dp, 757
 - swap_xy_i4, 757
 - swap_xy_sp, 757
- mo_utils.f90, 1040
- mo_utils::eq, 811
 - equal_dp, 811
 - equal_sp, 812
- mo_utils::equal, 812
 - equal_dp, 813
 - equal_sp, 813
- mo_utils::ge, 816
 - greaterequal_dp, 816
 - greaterequal_sp, 816
- mo_utils::greaterequal, 825
 - greaterequal_dp, 825
 - greaterequal_sp, 825
- mo_utils::is_finite, 832
 - is_finite_dp, 833
 - is_finite_sp, 833
- mo_utils::is_nan, 833
 - is_nan_dp, 833
 - is_nan_sp, 833
- mo_utils::is_normal, 834
 - is_normal_dp, 834
 - is_normal_sp, 834
- mo_utils::le, 839
 - lesserequal_dp, 839
 - lesserequal_sp, 839
- mo_utils::lesserequal, 840
 - lesserequal_dp, 840
 - lesserequal_sp, 840
- mo_utils::locate, 843
 - locate_0d_dp, 843
 - locate_0d_sp, 843
 - locate_1d_dp, 844
 - locate_1d_sp, 844
- mo_utils::ne, 890
 - notequal_dp, 890
 - notequal_sp, 890
- mo_utils::notequal, 893
 - notequal_dp, 893
 - notequal_sp, 894
- mo_utils::special_value, 945
 - special_value_dp, 946
 - special_value_sp, 946
- mo_utils::swap, 953
 - swap_vec_dp, 954
 - swap_vec_i4, 954
 - swap_vec_sp, 954
 - swap_xy_dp, 954
 - swap_xy_i4, 954
 - swap_xy_sp, 954
- mo_write_ascii, 757
 - write_configfile, 758
 - write_optifile, 759
 - write_optinamelist, 759
- mo_write_ascii.f90, 1041
- mo_write_fluxes_states, 760
 - close, 761

- createoutputfile, [762](#)
- fluxesunit, [763](#)
- geocoordinates, [765](#)
- mapcoordinates, [766](#)
- newoutputdataset, [767](#)
- newoutputvariable, [769](#)
- updatedataset, [770](#)
- updatevariable, [771](#)
- writetimestep, [772](#)
- writevariableattributes, [773](#)
- writevariabletimestep, [774](#)
- mo_write_fluxes_states.f90, [1041](#)
- mo_write_fluxes_states::outputdataset, [899](#)
 - all, [901](#)
 - basin, [901](#)
 - close, [900](#)
 - count, [901](#)
 - counter, [901](#)
 - created, [901](#)
 - ibasin, [901](#)
 - id, [901](#)
 - nc, [901](#)
 - ncdataset, [901](#)
 - steps, [902](#)
 - store, [902](#)
 - time, [902](#)
 - to, [902](#)
 - updatedataset, [900](#)
 - variables, [902](#)
 - vars, [902](#)
 - write, [902](#)
 - writetimestep, [900](#)
 - written, [902](#)
- mo_write_fluxes_states::outputvariable, [907](#)
 - average, [908](#)
 - avg, [908](#)
 - before, [908](#)
 - between, [909](#)
 - calls, [909](#)
 - contains, [909](#)
 - count, [909](#)
 - counter, [909](#)
 - data, [909](#)
 - mask, [909](#)
 - nc, [909](#)
 - ncdataset, [910](#)
 - number, [910](#)
 - of, [910](#)
 - reconstruct, [910](#)
 - store, [910](#)
 - the, [910](#)
 - to, [910](#)
 - updatevariable, [908](#), [911](#)
 - variable, [911](#)
 - which, [911](#)
 - writes, [911](#)
 - writevariabletimestep, [908](#)
 - writing, [911](#)
- mo_xor4096, [775](#)
 - get_timeseed_i4_0d, [775](#)
 - get_timeseed_i4_1d, [775](#)
 - get_timeseed_i8_0d, [776](#)
 - get_timeseed_i8_1d, [776](#)
 - n_save_state, [778](#)
 - xor4096d_0d, [776](#)
 - xor4096d_1d, [776](#)
 - xor4096f_0d, [776](#)
 - xor4096f_1d, [776](#)
 - xor4096gd_0d, [777](#)
 - xor4096gd_1d, [777](#)
 - xor4096gf_0d, [777](#)
 - xor4096gf_1d, [777](#)
 - xor4096l_0d, [777](#)
 - xor4096l_1d, [778](#)
 - xor4096s_0d, [778](#)
 - xor4096s_1d, [778](#)
- mo_xor4096.f90, [1042](#)
- mo_xor4096::get_timeseed, [823](#)
 - get_timeseed_i4_0d, [823](#)
 - get_timeseed_i4_1d, [823](#)
 - get_timeseed_i8_0d, [823](#)
 - get_timeseed_i8_1d, [823](#)
- mo_xor4096::xor4096, [973](#)
 - xor4096d_0d, [973](#)
 - xor4096d_1d, [973](#)
 - xor4096f_0d, [973](#)
 - xor4096f_1d, [973](#)
 - xor4096l_0d, [973](#)
 - xor4096l_1d, [974](#)
 - xor4096s_0d, [974](#)
 - xor4096s_1d, [974](#)
- mo_xor4096::xor4096g, [974](#)
 - xor4096gd_0d, [974](#)
 - xor4096gd_1d, [975](#)
 - xor4096gf_0d, [975](#)
 - xor4096gf_1d, [975](#)
- mode
 - mo_netcdf::ncdataset, [872](#)
- moment_dp
 - mo_moment, [263](#)
 - mo_moment::moment, [857](#)
- moment_sp
 - mo_moment, [263](#)
 - mo_moment::moment, [857](#)
- moments
 - mo_mrm_signatures, [418](#)
- month_counter
 - mo_mrm_write, [436](#)
- movedownonecell
 - mo_mrm_net_startup, [343](#)
- moveup
 - mo_mrm_net_startup, [344](#)
- mpr
 - mo_multi_param_reg, [456](#)
- mpr_runoff
 - mo_mpr_runoff, [273](#)

- mpr_sm
 - mo_mpr_soilmoist, [284](#)
- mpr_smhorizons
 - mo_mpr_smhorizons, [276](#)
- mrm_coupling_mode
 - mo_mrm_global_variables, [316](#)
- mrm_driver
 - mrm_driver.f90, [1043](#)
- mrm_driver.f90, [1043](#)
 - mrm_driver, [1043](#)
- mrm_eval
 - mo_mrm_eval, [292](#)
- mrm_init
 - mo_mrm_init, [322](#)
- mrm_init_param
 - mo_mrm_init, [324](#)
- mrm_l0_variable_init
 - mo_mrm_read_data, [392](#)
- mrm_l1_variable_init
 - mo_mrm_read_data, [393](#)
- mrm_read_discharge
 - mo_mrm_read_data, [394](#)
- mrm_read_l0_data
 - mo_mrm_read_data, [395](#)
- mrm_read_restart_config
 - mo_mrm_restart, [400](#)
- mrm_read_restart_states
 - mo_mrm_restart, [401](#)
- mrm_read_total_runoff
 - mo_mrm_read_data, [396](#)
- mrm_routing
 - mo_mrm_routing, [409](#)
- mrm_runoff
 - mo_mrm_global_variables, [316](#)
- mrm_update_param
 - mo_mrm_init, [325](#)
- mrm_write
 - mo_mrm_write, [428](#)
- mrm_write_optifile
 - mo_mrm_write, [429](#)
- mrm_write_optinamelist
 - mo_mrm_write, [430](#)
- mrm_write_output_fluxes
 - mo_mrm_write, [431](#)
- mrm_write_restart
 - mo_mrm_restart, [403](#)
- mse_dp_1d
 - mo_errormeasures, [120](#)
 - mo_errormeasures::mse, [859](#)
- mse_dp_2d
 - mo_errormeasures, [121](#)
 - mo_errormeasures::mse, [860](#)
- mse_dp_3d
 - mo_errormeasures, [122](#)
 - mo_errormeasures::mse, [860](#)
- mse_sp_1d
 - mo_errormeasures, [122](#)
 - mo_errormeasures::mse, [860](#)
- mse_sp_2d
 - mo_errormeasures, [123](#)
 - mo_errormeasures::mse, [860](#)
- mse_sp_3d
 - mo_errormeasures, [124](#)
 - mo_errormeasures::mse, [860](#)
- mstart
 - mo_common_variables::period, [924](#)
- multi_objective_ae_fdc_lsv_nse_djf
 - mo_mrm_objective_function_runoff, [359](#)
- multi_objective_lnnse_highflow_lnnse_lowflow
 - mo_mrm_objective_function_runoff, [361](#)
- multi_objective_lnnse_highflow_lnnse_lowflow_2
 - mo_mrm_objective_function_runoff, [363](#)
- multi_objective_nse_lnnse
 - mo_mrm_objective_function_runoff, [366](#)
- multi_objective_runoff
 - mo_mrm_objective_function_runoff, [368](#)
- n
 - mo_kind::sprs2_dp, [947](#)
 - mo_kind::sprs2_sp, [949](#)
- n_element_dp
 - mo_percentile, [631](#)
 - mo_percentile::n_element, [861](#)
- n_element_sp
 - mo_percentile, [631](#)
 - mo_percentile::n_element, [862](#)
- n_save_state
 - mo_xor4096, [778](#)
- name
 - mo_ncwrite::attribute, [788](#)
 - mo_ncwrite::dims, [807](#)
 - mo_ncwrite::variable, [971](#)
- natt
 - mo_ncwrite::variable, [971](#)
- nattdim
 - mo_ncwrite, [509](#)
- nbasins
 - mo_global_variables, [171](#)
 - mo_mrm_global_variables, [316](#)
- nc
 - mo_mrm_write, [436](#)
 - mo_mrm_write_fluxes_states::outputdataset, [905](#)
 - mo_mrm_write_fluxes_states::outputvariable, [914](#)
 - mo_write_fluxes_states::outputdataset, [901](#)
 - mo_write_fluxes_states::outputvariable, [909](#)
- ncclose
 - mo_ncread, [482](#)
- ncdataset
 - mo_mrm_write_fluxes_states::outputdataset, [905](#)
 - mo_mrm_write_fluxes_states::outputvariable, [915](#)
 - mo_write_fluxes_states::outputdataset, [901](#)
 - mo_write_fluxes_states::outputvariable, [910](#)
- ncfile_discharge
 - mo_mrm_file, [298](#)
- ncolpars
 - mo_mhm_constants, [249](#)
 - mo_mrm_constants, [290](#)

- ncols
 - mo_global_variables::gridgeoref, 826
 - mo_mrm_global_variables::gridgeoref, 828
- ncopen
 - mo_ncread, 482
- ndays
 - mo_julian, 212
- ndims
 - mo_ncwrite, 509
 - mo_ncwrite::variable, 971
- ndyin
 - mo_julian, 213
- neigh_value
 - mo_dds, 112
- nerr
 - mo_constants, 96
- nerror_model
 - mo_common_variables, 90
- netcdf
 - mo_netcdf::ncdataset, 872
 - mo_netcdf::ncdimension, 889
- neutron_integral_afast
 - mo_global_variables, 171
- newncdataset
 - mo_netcdf, 543
- newncdimension
 - mo_netcdf, 543
- newncvariable
 - mo_netcdf, 543
- newoutputdataset
 - mo_mrm_write_fluxes_states, 441
 - mo_write_fluxes_states, 767
- newoutputvariable
 - mo_mrm_write_fluxes_states, 442
 - mo_write_fluxes_states, 769
- ngatt
 - mo_ncwrite, 509
- ngauges
 - mo_mrm_global_variables::basininfo, 798
- ngaugestotal
 - mo_mrm_global_variables, 317
- ngeounits
 - mo_global_variables, 172
- nhorizons
 - mo_global_variables::soiltype, 937
- nin
 - mo_constants, 96
- ninflowgauges
 - mo_mrm_global_variables::basininfo, 798
- ninflowgaugestotal
 - mo_mrm_global_variables, 317
- niterations
 - mo_common_variables, 90
- nlaiclass
 - mo_global_variables, 172
- nlcover_class
 - mo_mhm_constants, 249
- nlcoverscene
 - mo_global_variables, 172
 - mo_mrm_global_variables, 317
- nlvls
 - mo_ncwrite::variable, 971
- nmaxatt
 - mo_ncwrite, 509
- nmaxdim
 - mo_ncwrite, 509
- nmeasperday
 - mo_mrm_global_variables, 317
- nmeasperday_tws
 - mo_global_variables, 172
- nndv_dp
 - mo_spatialsimilarity, 711
 - mo_spatialsimilarity::nndv, 893
- nndv_sp
 - mo_spatialsimilarity, 711
 - mo_spatialsimilarity::nndv, 893
- nnml
 - mo_constants, 96
- nobs
 - mo_common_variables::period, 924
- nodata_dp
 - mo_mhm_constants, 250
 - mo_mrm_constants, 291
- nodata_i4
 - mo_mhm_constants, 250
 - mo_mrm_constants, 291
- nodata_value
 - mo_global_variables::gridgeoref, 826
 - mo_mrm_global_variables::gridgeoref, 828
- nonnull
 - mo_string_utils, 724
- notequal_dp
 - mo_utils, 755
 - mo_utils::ne, 890
 - mo_utils::notequal, 893
- notequal_sp
 - mo_utils, 755
 - mo_utils::ne, 890
 - mo_utils::notequal, 894
- nout
 - mo_constants, 96
- noutflxstate
 - mo_mhm_constants, 250
 - mo_mrm_constants, 291
- npar2_arth
 - mo_corr, 108
- npar_arth
 - mo_corr, 108
- nprocesses
 - mo_common_variables, 90
- nroutingstates
 - mo_mrm_constants, 291
- nrows
 - mo_global_variables::gridgeoref, 826
 - mo_mrm_global_variables::gridgeoref, 828
- nse_dp_1d

- mo_errormeasures, 124
 - mo_errormeasures::nse, 894
- nse_dp_2d
 - mo_errormeasures, 125
 - mo_errormeasures::nse, 894
- nse_dp_3d
 - mo_errormeasures, 125
 - mo_errormeasures::nse, 894
- nse_sp_1d
 - mo_errormeasures, 125
 - mo_errormeasures::nse, 895
- nse_sp_2d
 - mo_errormeasures, 125
 - mo_errormeasures::nse, 895
- nse_sp_3d
 - mo_errormeasures, 125
 - mo_errormeasures::nse, 895
- nsoilhorizons_mhm
 - mo_global_variables, 172
- nsoilhorizons_sm_input
 - mo_global_variables, 172
- nsoiltypes
 - mo_global_variables, 172
- nsubs
 - mo_ncwrite::variable, 971
- ntillhorizons
 - mo_global_variables::soiltype, 938
- ntimesteps_l1_et
 - mo_global_variables, 172
- ntimesteps_l1_neutrons
 - mo_global_variables, 173
- ntimesteps_l1_sm
 - mo_global_variables, 173
- ntstepday
 - mo_global_variables, 173
 - mo_mrm_global_variables, 317
- number
 - mo_mrm_write_fluxes_states::outputvariable, 915
 - mo_write_fluxes_states::outputvariable, 910
- nunitnml
 - mo_nml, 579
- nvalues
 - mo_ncwrite::attribute, 788
- nvars
 - mo_ncwrite, 509
- objective
 - mo_objective_function, 582
- objective_equal_nse_lnnse
 - mo_mrm_objective_function_runoff, 369
- objective_et_kge_catchment_avg
 - mo_objective_function, 584
- objective_kge
 - mo_mrm_objective_function_runoff, 372
- objective_kge_q_et
 - mo_objective_function, 587
- objective_kge_q_rmse_et
 - mo_objective_function, 589
- objective_kge_q_rmse_tws
 - mo_objective_function, 591
- objective_kge_q_sm_corr
 - mo_objective_function, 593
- objective_lnnse
 - mo_mrm_objective_function_runoff, 374
- objective_multiple_gauges_kge_power6
 - mo_mrm_objective_function_runoff, 376
- objective_neutrons_kge_catchment_avg
 - mo_objective_function, 595
- objective_nse
 - mo_mrm_objective_function_runoff, 379
- objective_power6_nse_lnnse
 - mo_mrm_objective_function_runoff, 381
- objective_sm_corr
 - mo_objective_function, 598
- objective_sm_kge_catchment_avg
 - mo_objective_function, 601
- objective_sm_pd
 - mo_objective_function, 604
- objective_sm_sse_standard_score
 - mo_objective_function, 607
- objective_sse
 - mo_mrm_objective_function_runoff, 383
- of
 - mo_mrm_write_fluxes_states::outputvariable, 915
 - mo_netcdf::ncdataset, 872
 - mo_write_fluxes_states::outputvariable, 910
- oldintegration
 - mo_neutrons, 573
- open
 - mo_netcdf::ncdataset, 872
- open_netcdf
 - mo_ncwrite, 495
- open_nml
 - mo_nml, 576
- opened
 - mo_netcdf::ncdataset, 873
- opti_function
 - mo_common_variables, 90
- opti_method
 - mo_common_variables, 91
- optimization
 - mo_optimization, 610
- optimize
 - mo_common_variables, 91
- optimize_restart
 - mo_common_variables, 91
- outputflxstate
 - mo_global_variables, 173
- outputflxstate_mrm
 - mo_mrm_global_variables, 317
- p0_dp
 - mo_constants, 96
- p0_sp
 - mo_constants, 96
- p1_initstatefluxes
 - mo_mhm_constants, 250
 - mo_mrm_constants, 291

- p2_initstatefluxes
 - mo_mhm_constants, [250](#)
- p3_initstatefluxes
 - mo_mhm_constants, [250](#)
- p4_initstatefluxes
 - mo_mhm_constants, [251](#)
- p5_initstatefluxes
 - mo_mhm_constants, [251](#)
- parameter_regularization
 - mo_mrm_objective_function_runoff, [385](#)
- parent
 - mo_netcdf::ncdimension, [889](#)
- pargen_anneal_dp
 - mo_anneal, [78](#)
- pargen_dds_dp
 - mo_anneal, [78](#)
- pargen_dp
 - mo_mcmc, [223](#)
- pargennorm_dp
 - mo_mcmc, [224](#)
- parstt
 - mo_sce, [689](#)
- paste_char_m_m
 - mo_append, [84](#)
 - mo_append::paste, [917](#)
- paste_char_m_s
 - mo_append, [84](#)
 - mo_append::paste, [917](#)
- paste_char_m_v
 - mo_append, [84](#)
 - mo_append::paste, [918](#)
- paste_dp_m_m
 - mo_append, [84](#)
 - mo_append::paste, [918](#)
- paste_dp_m_s
 - mo_append, [85](#)
 - mo_append::paste, [918](#)
- paste_dp_m_v
 - mo_append, [85](#)
 - mo_append::paste, [918](#)
- paste_i4_m_m
 - mo_append, [85](#)
 - mo_append::paste, [918](#)
- paste_i4_m_s
 - mo_append, [85](#)
 - mo_append::paste, [918](#)
- paste_i4_m_v
 - mo_append, [85](#)
 - mo_append::paste, [918](#)
- paste_i8_m_m
 - mo_append, [85](#)
 - mo_append::paste, [919](#)
- paste_i8_m_s
 - mo_append, [85](#)
 - mo_append::paste, [919](#)
- paste_i8_m_v
 - mo_append, [86](#)
 - mo_append::paste, [919](#)
- paste_lgt_m_m
 - mo_append, [86](#)
 - mo_append::paste, [919](#)
- paste_lgt_m_s
 - mo_append, [86](#)
 - mo_append::paste, [919](#)
- paste_lgt_m_v
 - mo_append, [86](#)
 - mo_append::paste, [919](#)
- paste_sp_m_m
 - mo_append, [86](#)
 - mo_append::paste, [920](#)
- paste_sp_m_s
 - mo_append, [86](#)
 - mo_append::paste, [920](#)
- paste_sp_m_v
 - mo_append, [86](#)
 - mo_append::paste, [920](#)
- pd_dp
 - mo_spatialsimilarity, [711](#)
 - mo_spatialsimilarity::pd, [921](#)
- pd_sp
 - mo_spatialsimilarity, [711](#)
 - mo_spatialsimilarity::pd, [922](#)
- peakdistribution
 - mo_mrm_signatures, [420](#)
- percentile_0d_dp
 - mo_percentile, [631](#)
 - mo_percentile::percentile, [922](#)
- percentile_0d_sp
 - mo_percentile, [631](#)
 - mo_percentile::percentile, [922](#)
- percentile_1d_dp
 - mo_percentile, [632](#)
 - mo_percentile::percentile, [922](#)
- percentile_1d_sp
 - mo_percentile, [632](#)
 - mo_percentile::percentile, [923](#)
- perform_mpr
 - mo_global_variables, [173](#)
 - mo_mrm_global_variables, [317](#)
- pet_correctbyasp
 - mo_mpr_pet, [266](#)
- pet_correctbylai
 - mo_mpr_pet, [268](#)
- pet_hargreaves
 - mo_pet, [635](#)
- pet_penman
 - mo_pet, [637](#)
- pet_priestly
 - mo_pet, [639](#)
- pi
 - mo_constants, [97](#)
- pi_d
 - mo_constants, [97](#)
- pi_dp
 - mo_constants, [97](#)
 - mo_template, [729](#)

- pi_sp
 - mo_constants, 97
 - mo_template, 730
- pio2
 - mo_constants, 97
- pio2_d
 - mo_constants, 97
- pio2_dp
 - mo_constants, 97
- pio2_sp
 - mo_constants, 98
- position_nml
 - mo_nml, 577
- positioned
 - mo_nml, 579
- prepare_gridded_daily_lai_data
 - mo_prepare_gridded_lai, 645
- prepare_gridded_mean_monthly_lai_data
 - mo_prepare_gridded_lai, 646
- prepare_meteo_forcings_data
 - mo_meteo_forcings, 235
- priestley_taylor_alpha
 - mo_mpr_pet, 271
- print_startup_message
 - mo_mrm_init, 326
- processmatrix
 - mo_common_variables, 91
- project_details
 - mo_global_variables, 173
 - mo_mrm_global_variables, 318
- psychro_dp
 - mo_constants, 98
- psychro_sp
 - mo_constants, 98
- pwp
 - mo_mpr_soilmoist, 288
- pwp_c
 - mo_mhm_constants, 251
- pwp_matpot_thetar
 - mo_mhm_constants, 251
- q
 - mo_mrm_global_variables::gaugingstation, 816
- qmedian_dp
 - mo_percentile, 632
 - mo_percentile::qmedian, 925
- qmedian_sp
 - mo_percentile, 632
 - mo_percentile::qmedian, 925
- r_ctrper
 - mo_orderpack, 625
 - mo_orderpack::ctrper, 805
- r_fndnth
 - mo_orderpack, 625
 - mo_orderpack::fndnth, 813
- r_indmed
 - mo_orderpack, 625
 - mo_orderpack::indmed, 830
- r_indnth
 - mo_orderpack, 625
 - mo_orderpack::indnth, 831
- r_inspar
 - mo_orderpack, 625
 - mo_orderpack::inspar, 831
- r_inssor
 - mo_orderpack, 626
 - mo_orderpack::inssor, 832
- r_med
 - mo_orderpack, 626
- r_median
 - mo_orderpack, 626
 - mo_orderpack::omedian, 898
- r_mrgref
 - mo_orderpack, 626
 - mo_orderpack::mrgref, 858
- r_mrgrnk
 - mo_orderpack, 627
 - mo_orderpack::mrgrnk, 859
- r_mulcnt
 - mo_orderpack, 627
 - mo_orderpack::mulcnt, 861
- r_nearless
 - mo_orderpack, 627
 - mo_orderpack::nearless, 891
- r_rapknr
 - mo_orderpack, 627
 - mo_orderpack::rapknr, 926
- r_refpar
 - mo_orderpack, 627
 - mo_orderpack::refpar, 929
- r_refsor
 - mo_orderpack, 627
 - mo_orderpack::refsor, 929
 - mo_orderpack::sort, 942
- r_rinpar
 - mo_orderpack, 628
 - mo_orderpack::rinpar, 930
- r_rnkpar
 - mo_orderpack, 628
 - mo_orderpack::rnkpar, 932
- r_subsor
 - mo_orderpack, 628
- r_uniinv
 - mo_orderpack, 628
 - mo_orderpack::uniinv, 956
- r_unipar
 - mo_orderpack, 629
 - mo_orderpack::unipar, 957
- r_unirnk
 - mo_orderpack, 629
 - mo_orderpack::unirnk, 958
- r_unista
 - mo_orderpack, 629
 - mo_orderpack::unista, 958
- r_valmed
 - mo_orderpack, 629

- mo_orderpack::valmed, 959
- r_valnth
 - mo_orderpack, 629
 - mo_orderpack::valnth, 960
- RELEASES.md, 1044
- rad2deg_dp
 - mo_constants, 98
- rad2deg_sp
 - mo_constants, 98
- radiusearth_dp
 - mo_constants, 98
- radiusearth_sp
 - mo_constants, 98
- read_basin_avg_tws
 - mo_read_optional_data, 663
- read_config
 - mo_read_config, 648
- read_data
 - mo_read_wrapper, 674
- read_error
 - mo_nml, 579
- read_evapotranspiration
 - mo_read_optional_data, 664
- read_forcing_nc
 - mo_read_forcing_nc, 651
- read_geoformation_lut
 - mo_read_lut, 658
- read_header_ascii
 - mo_read_spatial_data, 668
- read_lai_lut
 - mo_read_lut, 659
- read_latlon
 - mo_mrm_read_latlon, 398
 - mo_read_latlon, 656
- read_meteo_bin
 - mo_read_meteo, 661
- read_meteo_weights
 - mo_global_variables, 173
- read_mrm_config
 - mo_mrm_read_config, 388
- read_mrm_config_coupling
 - mo_mrm_read_config, 389
- read_mrm_routing_params
 - mo_mrm_read_config, 390
- read_neutrons
 - mo_read_optional_data, 665
- read_restart
 - mo_global_variables, 173
 - mo_mrm_global_variables, 318
- read_restart_config
 - mo_restart, 676
- read_restart_states
 - mo_restart, 677
- read_soil_lut
 - mo_soil_database, 700
- read_soil_moisture
 - mo_read_optional_data, 666
- read_spatial_data_ascii_dp
 - mo_read_spatial_data, 669
 - mo_read_spatial_data::read_spatial_data_ascii, 927
- read_spatial_data_ascii_i4
 - mo_read_spatial_data, 670
 - mo_read_spatial_data::read_spatial_data_ascii, 927
- read_timeseries
 - mo_read_timeseries, 671
- read_weights_nc
 - mo_read_forcing_nc, 654
- readper
 - mo_global_variables, 174
 - mo_mrm_global_variables, 318
- realft_dp
 - mo_corr, 106
 - mo_corr::realft, 928
- realft_sp
 - mo_corr, 107
 - mo_corr::realft, 928
- reconstruct
 - mo_mrm_write_fluxes_states::outputvariable, 915
 - mo_write_fluxes_states::outputvariable, 910
- reg_rout
 - mo_mrm_mpr, 329
- resolutionhydrology
 - mo_global_variables, 174
 - mo_mrm_global_variables, 318
- resolutionrouting
 - mo_global_variables, 174
 - mo_mrm_global_variables, 318
- rho0_dp
 - mo_constants, 99
- rho0_sp
 - mo_constants, 99
- rmse_dp_1d
 - mo_errormeasures, 125
 - mo_errormeasures::rmse, 930
- rmse_dp_2d
 - mo_errormeasures, 126
 - mo_errormeasures::rmse, 931
- rmse_dp_3d
 - mo_errormeasures, 126
 - mo_errormeasures::rmse, 931
- rmse_sp_1d
 - mo_errormeasures, 126
 - mo_errormeasures::rmse, 931
- rmse_sp_2d
 - mo_errormeasures, 127
 - mo_errormeasures::rmse, 931
- rmse_sp_3d
 - mo_errormeasures, 127
 - mo_errormeasures::rmse, 931
- rotate_fdir_variable
 - mo_mrm_read_data, 397
- rout_space_weight
 - mo_mrm_constants, 291
- routingstates

- mo_global_variables, 174
- runoff_sat_zone
 - mo_runoff, 682
- runoff_unsat_zone
 - mo_runoff, 684
- runoffratio
 - mo_mrm_signatures, 421
- rzdepth
 - mo_global_variables::soiltype, 938
- s
 - mo_netcdf::ncdimension, 890
- sa_temp
 - mo_common_variables, 91
- sae_dp_1d
 - mo_errormeasures, 128
 - mo_errormeasures::sae, 933
- sae_dp_2d
 - mo_errormeasures, 128
 - mo_errormeasures::sae, 933
- sae_dp_3d
 - mo_errormeasures, 129
 - mo_errormeasures::sae, 933
- sae_sp_1d
 - mo_errormeasures, 130
 - mo_errormeasures::sae, 933
- sae_sp_2d
 - mo_errormeasures, 130
 - mo_errormeasures::sae, 933
- sae_sp_3d
 - mo_errormeasures, 131
 - mo_errormeasures::sae, 933
- sand
 - mo_global_variables::soiltype, 938
- sat_vap_pressure
 - mo_pet, 640
- satpressureslope1
 - mo_mhm_constants, 251
- sce
 - mo_sce, 690
- sce_ngs
 - mo_common_variables, 91
- sce_npg
 - mo_common_variables, 91
- sce_nps
 - mo_common_variables, 92
- secday_dp
 - mo_constants, 99
- secday_sp
 - mo_constants, 99
- seed
 - mo_common_variables, 92
- selectcalendar
 - mo_julian, 214
- separator
 - mo_string_utils, 727
- set_netcdf
 - mo_set_netcdf_outputs, 695
- set_optional
 - mo_sce.f90, 1031
- setattribute
 - mo_netcdf::ncdataset, 869
 - mo_netcdf::ncdimension, 883
- setcalendarinteger
 - mo_julian, 215
 - mo_julian::setcalendar, 934
- setcalendarstring
 - mo_julian, 216
 - mo_julian::setcalendar, 934
- setdata
 - mo_netcdf::ncdimension, 884
- setdata1df32
 - mo_netcdf, 543
 - mo_netcdf::ncdimension, 884
- setdata1df64
 - mo_netcdf, 544
 - mo_netcdf::ncdimension, 884
- setdata1di16
 - mo_netcdf, 544
 - mo_netcdf::ncdimension, 884
- setdata1di32
 - mo_netcdf, 545
 - mo_netcdf::ncdimension, 884
- setdata1di8
 - mo_netcdf, 545
 - mo_netcdf::ncdimension, 884
- setdata2df32
 - mo_netcdf, 546
 - mo_netcdf::ncdimension, 885
- setdata2df64
 - mo_netcdf, 546
 - mo_netcdf::ncdimension, 885
- setdata2di16
 - mo_netcdf, 547
 - mo_netcdf::ncdimension, 885
- setdata2di32
 - mo_netcdf, 547
 - mo_netcdf::ncdimension, 885
- setdata2di8
 - mo_netcdf, 548
 - mo_netcdf::ncdimension, 885
- setdata3df32
 - mo_netcdf, 548
 - mo_netcdf::ncdimension, 885
- setdata3df64
 - mo_netcdf, 549
 - mo_netcdf::ncdimension, 885
- setdata3di16
 - mo_netcdf, 549
 - mo_netcdf::ncdimension, 885
- setdata3di32
 - mo_netcdf, 550
 - mo_netcdf::ncdimension, 885
- setdata3di8
 - mo_netcdf, 550
 - mo_netcdf::ncdimension, 886
- setdata4df32

- mo_netcdf, 551
- mo_netcdf::ncdimension, 886
- setdata4df64
 - mo_netcdf, 551
 - mo_netcdf::ncdimension, 886
- setdata4di16
 - mo_netcdf, 552
 - mo_netcdf::ncdimension, 886
- setdata4di32
 - mo_netcdf, 552
 - mo_netcdf::ncdimension, 886
- setdata4di8
 - mo_netcdf, 553
 - mo_netcdf::ncdimension, 886
- setdata5df32
 - mo_netcdf, 553
 - mo_netcdf::ncdimension, 886
- setdata5df64
 - mo_netcdf, 554
 - mo_netcdf::ncdimension, 886
- setdata5di16
 - mo_netcdf, 554
 - mo_netcdf::ncdimension, 886
- setdata5di32
 - mo_netcdf, 555
 - mo_netcdf::ncdimension, 887
- setdata5di8
 - mo_netcdf, 555
 - mo_netcdf::ncdimension, 887
- setdatascalarf32
 - mo_netcdf, 556
 - mo_netcdf::ncdimension, 887
- setdatascalarf64
 - mo_netcdf, 556
 - mo_netcdf::ncdimension, 887
- setdatascalarf16
 - mo_netcdf, 557
 - mo_netcdf::ncdimension, 887
- setdatascalarf32
 - mo_netcdf, 557
 - mo_netcdf::ncdimension, 887
- setdatascalarf8
 - mo_netcdf, 557
 - mo_netcdf::ncdimension, 887
- setdimension
 - mo_netcdf, 558
 - mo_netcdf::ncdataset, 869
- setfillvalue
 - mo_netcdf::ncdimension, 887
- setglobalattributechar
 - mo_netcdf, 558
 - mo_netcdf::ncdataset, 870
- setglobalattributef32
 - mo_netcdf, 559
 - mo_netcdf::ncdataset, 870
- setglobalattributef64
 - mo_netcdf, 559
 - mo_netcdf::ncdataset, 870
- setglobalattributei16
 - mo_netcdf, 559
 - mo_netcdf::ncdataset, 870
- setglobalattributei32
 - mo_netcdf, 560
 - mo_netcdf::ncdataset, 870
- setglobalattributei8
 - mo_netcdf, 560
 - mo_netcdf::ncdataset, 870
- setup_description
 - mo_global_variables, 174
 - mo_mrm_global_variables, 318
- setvariable
 - mo_netcdf::ncdataset, 870
- setvariableattributechar
 - mo_netcdf, 561
 - mo_netcdf::ncdimension, 888
- setvariableattributef32
 - mo_netcdf, 561
 - mo_netcdf::ncdimension, 888
- setvariableattributef64
 - mo_netcdf, 561
 - mo_netcdf::ncdimension, 888
- setvariableattributei16
 - mo_netcdf, 562
 - mo_netcdf::ncdimension, 888
- setvariableattributei32
 - mo_netcdf, 562
 - mo_netcdf::ncdimension, 888
- setvariableattributei8
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 888
- setvariablefillvaluef32
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 888
- setvariablefillvaluef64
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 889
- setvariablefillvaluei16
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 889
- setvariablefillvaluei32
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 889
- setvariablefillvaluei8
 - mo_netcdf, 563
 - mo_netcdf::ncdimension, 889
- setvariablewithids
 - mo_netcdf, 564
 - mo_netcdf::ncdataset, 871
- setvariablewithnames
 - mo_netcdf, 565
 - mo_netcdf::ncdataset, 871
- setvariablewithtypes
 - mo_netcdf, 565
 - mo_netcdf::ncdataset, 871
- sigma_dp
 - mo_constants, 99

- sigma_sp
 - mo_constants, 99
- simper
 - mo_global_variables, 174
 - mo_mrm_global_variables, 318
- simulation_type
 - mo_global_variables, 174
 - mo_mrm_global_variables, 319
- single_objective_runoff
 - mo_mrm_objective_function_runoff, 385
- skewness_dp
 - mo_moment, 263
 - mo_moment::skewness, 935
- skewness_sp
 - mo_moment, 263
 - mo_moment::skewness, 935
- slope_satpressure
 - mo_pet, 642
- snow_acc_melt_param
 - mo_multi_param_reg, 461
- snow_accum_melt
 - mo_snow_accum_melt, 696
- soil_moisture
 - mo_soil_moisture, 705
- soildb
 - mo_global_variables, 174
- solarconst_dp
 - mo_constants, 99
- solarconst_sp
 - mo_constants, 100
- sort_index_dp
 - mo_orderpack, 629
 - mo_orderpack::sort_index, 942
- sort_index_i4
 - mo_orderpack, 629
 - mo_orderpack::sort_index, 942
- sort_index_sp
 - mo_orderpack, 630
 - mo_orderpack::sort_index, 942
- sort_matrix
 - mo_sce, 694
- sp
 - mo_kind, 219
- sp2str
 - mo_string_utils, 725
 - mo_string_utils::num2str, 897
- spatial_aggregation_3d
 - mo_spatial_agg_disagg_forcing, 709
 - mo_spatial_agg_disagg_forcing::spatial_aggregation, 943
- spatial_aggregation_4d
 - mo_spatial_agg_disagg_forcing, 709
 - mo_spatial_agg_disagg_forcing::spatial_aggregation, 943
- spatial_disaggregation_3d
 - mo_spatial_agg_disagg_forcing, 709
 - mo_spatial_agg_disagg_forcing::spatial_disaggregation, 944
- spatial_disaggregation_4d
 - mo_spatial_agg_disagg_forcing, 710
 - mo_spatial_agg_disagg_forcing::spatial_disaggregation, 945
- spc
 - mo_kind, 220
- specheatet_dp
 - mo_constants, 100
- specheatet_sp
 - mo_constants, 100
- special_value_dp
 - mo_utils, 755
 - mo_utils::special_value, 946
- special_value_sp
 - mo_utils, 756
 - mo_utils::special_value, 946
- splitstring
 - mo_string_utils, 725
- sqrt2
 - mo_constants, 100
- sqrt2_d
 - mo_constants, 100
- sqrt2_dp
 - mo_constants, 100
- sqrt2_sp
 - mo_constants, 100
- sse_dp_1d
 - mo_errormeasures, 132
 - mo_errormeasures::sse, 949
- sse_dp_2d
 - mo_errormeasures, 132
 - mo_errormeasures::sse, 949
- sse_dp_3d
 - mo_errormeasures, 133
 - mo_errormeasures::sse, 950
- sse_sp_1d
 - mo_errormeasures, 134
 - mo_errormeasures::sse, 950
- sse_sp_2d
 - mo_errormeasures, 134
 - mo_errormeasures::sse, 950
- sse_sp_3d
 - mo_errormeasures, 135
 - mo_errormeasures::sse, 950
- standard_score_dp
 - mo_standard_score, 712
 - mo_standard_score::standard_score, 951
- standard_score_sp
 - mo_standard_score, 712
 - mo_standard_score::standard_score, 951
- start
 - mo_ncwrite::variable, 971
 - mo_string_utils, 725
- status
 - mo_timer, 743
- stboltzmann
 - mo_mhm_constants, 251

- stddev_dp
 - mo_moment, [263](#)
 - mo_moment::stddev, [952](#)
- stddev_sp
 - mo_moment, [263](#)
 - mo_moment::stddev, [952](#)
- steps
 - mo_mrm_write_fluxes_states::outputdataset, [905](#)
 - mo_write_fluxes_states::outputdataset, [902](#)
- store
 - mo_mrm_write_fluxes_states::outputdataset, [905](#)
 - mo_mrm_write_fluxes_states::outputvariable, [915](#)
 - mo_write_fluxes_states::outputdataset, [902](#)
 - mo_write_fluxes_states::outputvariable, [910](#)
- str2num
 - mo_string_utils, [726](#)
- swap_1d_dpc
 - mo_corr, [107](#)
 - mo_corr::swap, [952](#)
- swap_1d_spc
 - mo_corr, [107](#)
 - mo_corr::swap, [953](#)
- swap_vec_dp
 - mo_utils, [756](#)
 - mo_utils::swap, [954](#)
- swap_vec_i4
 - mo_utils, [756](#)
 - mo_utils::swap, [954](#)
- swap_vec_sp
 - mo_utils, [756](#)
 - mo_utils::swap, [954](#)
- swap_xy_dp
 - mo_utils, [757](#)
 - mo_utils::swap, [954](#)
- swap_xy_i4
 - mo_utils, [757](#)
 - mo_utils::swap, [954](#)
- swap_xy_sp
 - mo_utils, [757](#)
 - mo_utils::swap, [954](#)
- t0_dp
 - mo_constants, [101](#)
- t0_sp
 - mo_constants, [101](#)
- tabularintegralfast
 - mo_neutrons, [573](#)
- temporal_disagg_forcing
 - mo_temporal_disagg_forcing, [732](#)
- tetens_c1
 - mo_mhm_constants, [251](#)
- tetens_c2
 - mo_mhm_constants, [252](#)
- tetens_c3
 - mo_mhm_constants, [252](#)
- the
 - mo_mrm_write_fluxes_states::outputvariable, [915](#)
 - mo_netcdf::ncdataset, [873](#)
 - mo_netcdf::ncdimension, [890](#)
 - mo_write_fluxes_states::outputvariable, [910](#)
- thetafc
 - mo_global_variables::soiltype, [938](#)
- thetafc_till
 - mo_global_variables::soiltype, [938](#)
- thetapw
 - mo_global_variables::soiltype, [938](#)
- thetapw_till
 - mo_global_variables::soiltype, [938](#)
- thetas
 - mo_global_variables::soiltype, [938](#)
- thetas_till
 - mo_global_variables::soiltype, [938](#)
- tillagedepth
 - mo_global_variables, [175](#)
- time
 - mo_mrm_write_fluxes_states::outputdataset, [905](#)
 - mo_write_fluxes_states::outputdataset, [902](#)
- timer_check
 - mo_timer, [735](#)
- timer_clear
 - mo_timer, [736](#)
- timer_get
 - mo_timer, [736](#)
- timer_print
 - mo_timer, [738](#)
- timer_start
 - mo_timer, [739](#)
- timer_stop
 - mo_timer, [740](#)
- timers_init
 - mo_timer, [741](#)
- timestep
 - mo_global_variables, [175](#)
 - mo_mrm_global_variables, [319](#)
- timestep_et_input
 - mo_global_variables, [175](#)
- timestep_lai_input
 - mo_global_variables, [175](#)
- timestep_model_inputs
 - mo_global_variables, [175](#)
 - mo_mrm_global_variables, [319](#)
- timestep_model_outputs
 - mo_global_variables, [175](#)
- timestep_model_outputs_mrm
 - mo_mrm_global_variables, [319](#)
- timestep_neutrons_input
 - mo_global_variables, [175](#)
- timestep_sm_input
 - mo_global_variables, [176](#)
- to
 - mo_mrm_write_fluxes_states::outputdataset, [906](#)
 - mo_mrm_write_fluxes_states::outputvariable, [915](#)
 - mo_write_fluxes_states::outputdataset, [902](#)
 - mo_write_fluxes_states::outputvariable, [910](#)
- tolower
 - mo_string_utils, [726](#)
- toupper

- mo_string_utils, 727
- twopi
 - mo_constants, 101
- twopi_d
 - mo_constants, 101
- twopi_dp
 - mo_constants, 101
- twopi_sp
 - mo_constants, 101
- twothird_dp
 - mo_constants, 101
- twothird_sp
 - mo_constants, 102
- twos
 - mo_global_variables::twosstructure, 955
- uaspect
 - mo_file, 142
- uconfig
 - mo_file, 142
 - mo_mrm_file, 298
- ud
 - mo_global_variables::soiltype, 939
- udaily_discharge
 - mo_file, 142
 - mo_mrm_file, 298
- undefoutput
 - mo_file, 142
 - mo_mrm_file, 298
- udem
 - mo_file, 142
 - mo_mrm_file, 298
- udischarge
 - mo_file, 142
 - mo_mrm_file, 299
- ufacc
 - mo_file, 142
 - mo_mrm_file, 299
- ufdir
 - mo_file, 143
 - mo_mrm_file, 299
- ugaugeloc
 - mo_mrm_file, 299
- ugeolut
 - mo_file, 143
- uhydrogeoclass
 - mo_file, 143
- ulai
 - mo_file, 143
- ulai_header
 - mo_file, 143
- ulaiclass
 - mo_file, 143
- ulailut
 - mo_file, 143
- ulcoverclass
 - mo_file, 143
 - mo_mrm_file, 299
- umeteo
 - mo_file, 144
- umeteo_header
 - mo_file, 144
- unamelist
 - mo_file, 144
- unamelist_mrm
 - mo_mrm_file, 299
- unamelist_param
 - mo_file, 144
 - mo_mrm_file, 299
- unlimited
 - mo_ncwrite::variable, 971
- uopti
 - mo_file, 144
 - mo_mrm_file, 300
- uopti_nml
 - mo_file, 144
 - mo_mrm_file, 300
- updateddataset
 - mo_mrm_write_fluxes_states, 443
 - mo_mrm_write_fluxes_states::outputdataset, 904
 - mo_write_fluxes_states, 770
 - mo_write_fluxes_states::outputdataset, 900
- updatevariable
 - mo_mrm_write_fluxes_states, 443
 - mo_mrm_write_fluxes_states::outputvariable, 913, 916
 - mo_write_fluxes_states, 771
 - mo_write_fluxes_states::outputvariable, 908, 911
- upscale_arithmetic_mean
 - mo_upscaling_operators, 746
- upscale_geometric_mean
 - mo_upscaling_operators, 749
- upscale_harmonic_mean
 - mo_upscaling_operators, 750
- uslope
 - mo_file, 145
- usoil_database
 - mo_file, 145
- usoilclass
 - mo_file, 145
- utws
 - mo_file, 145
- v
 - mo_ncwrite, 509
- val
 - mo_kind::sprs2_dp, 948
 - mo_kind::sprs2_sp, 949
- values
 - mo_ncwrite::attribute, 788
- var2nc_1d_dp
 - mo_ncwrite, 497
 - mo_ncwrite::var2nc, 961
- var2nc_1d_i4
 - mo_ncwrite, 498
 - mo_ncwrite::var2nc, 962
- var2nc_1d_sp
 - mo_ncwrite, 499

- mo_ncwrite::var2nc, 962
- var2nc_2d_dp
 - mo_ncwrite, 499
 - mo_ncwrite::var2nc, 962
- var2nc_2d_i4
 - mo_ncwrite, 500
 - mo_ncwrite::var2nc, 963
- var2nc_2d_sp
 - mo_ncwrite, 501
 - mo_ncwrite::var2nc, 963
- var2nc_3d_dp
 - mo_ncwrite, 501
 - mo_ncwrite::var2nc, 963
- var2nc_3d_i4
 - mo_ncwrite, 502
 - mo_ncwrite::var2nc, 964
- var2nc_3d_sp
 - mo_ncwrite, 503
 - mo_ncwrite::var2nc, 964
- var2nc_4d_dp
 - mo_ncwrite, 503
 - mo_ncwrite::var2nc, 964
- var2nc_4d_i4
 - mo_ncwrite, 504
 - mo_ncwrite::var2nc, 965
- var2nc_4d_sp
 - mo_ncwrite, 505
 - mo_ncwrite::var2nc, 965
- var2nc_5d_dp
 - mo_ncwrite, 505
 - mo_ncwrite::var2nc, 965
- var2nc_5d_i4
 - mo_ncwrite, 506
 - mo_ncwrite::var2nc, 966
- var2nc_5d_sp
 - mo_ncwrite, 507
 - mo_ncwrite::var2nc, 966
- variable
 - mo_mrm_write_fluxes_states::outputvariable, 916
 - mo_write_fluxes_states::outputvariable, 911
- variables
 - mo_mrm_write_fluxes_states::outputdataset, 906
 - mo_write_fluxes_states::outputdataset, 902
- variables_alloc
 - mo_init_states, 180
- variables_alloc_routing
 - mo_mrm_init, 327
- variables_default_init
 - mo_init_states, 181
- variables_default_init_routing
 - mo_mrm_init, 327
- variance_dp
 - mo_moment, 264
 - mo_moment::variance, 972
- variance_sp
 - mo_moment, 264
 - mo_moment::variance, 972
- varid
 - mo_ncwrite::variable, 972
- vars
 - mo_mrm_write_fluxes_states::outputdataset, 906
 - mo_write_fluxes_states::outputdataset, 902
- version
 - mo_file, 145
 - mo_mrm_file, 300
- version_date
 - mo_file, 145
 - mo_mrm_file, 300
- vgenuchten_sandtresh
 - mo_mhm_constants, 252
- vgenuchtenn_c1
 - mo_mhm_constants, 252
- vgenuchtenn_c10
 - mo_mhm_constants, 252
- vgenuchtenn_c11
 - mo_mhm_constants, 252
- vgenuchtenn_c12
 - mo_mhm_constants, 252
- vgenuchtenn_c13
 - mo_mhm_constants, 252
- vgenuchtenn_c14
 - mo_mhm_constants, 252
- vgenuchtenn_c15
 - mo_mhm_constants, 253
- vgenuchtenn_c16
 - mo_mhm_constants, 253
- vgenuchtenn_c17
 - mo_mhm_constants, 253
- vgenuchtenn_c18
 - mo_mhm_constants, 253
- vgenuchtenn_c2
 - mo_mhm_constants, 253
- vgenuchtenn_c3
 - mo_mhm_constants, 253
- vgenuchtenn_c4
 - mo_mhm_constants, 253
- vgenuchtenn_c5
 - mo_mhm_constants, 253
- vgenuchtenn_c6
 - mo_mhm_constants, 253
- vgenuchtenn_c7
 - mo_mhm_constants, 254
- vgenuchtenn_c8
 - mo_mhm_constants, 254
- vgenuchtenn_c9
 - mo_mhm_constants, 254
- warmingdays
 - mo_global_variables, 176
- warmingdays_mrm
 - mo_mrm_global_variables, 319
- warmper
 - mo_global_variables, 176
 - mo_mrm_global_variables, 319
- wd
 - mo_global_variables::soiltype, 939
- wflag

- mo_ncwrite::variable, 972
- which
 - mo_mrm_write_fluxes_states::outputvariable, 916
 - mo_write_fluxes_states::outputvariable, 911
- windmeasheight
 - mo_mhm_constants, 254
- write
 - mo_mrm_write_fluxes_states::outputdataset, 906
 - mo_write_fluxes_states::outputdataset, 902
- write_best_final
 - mo_sce.f90, 1032
- write_best_intermediate
 - mo_sce.f90, 1032
- write_configfile
 - mo_mrm_write, 433
 - mo_write_ascii, 758
- write_daily_obs_sim_discharge
 - mo_mrm_write, 434
- write_dynamic_netcdf
 - mo_ncwrite, 507
- write_optifile
 - mo_write_ascii, 759
- write_optinamelist
 - mo_write_ascii, 759
- write_population
 - mo_sce.f90, 1032
- write_restart
 - mo_global_variables, 176
 - mo_mrm_global_variables, 319
- write_restart_files
 - mo_restart, 679
- write_static_netcdf
 - mo_ncwrite, 508
- write_termination_case
 - mo_sce.f90, 1033
- writes
 - mo_mrm_write_fluxes_states::outputvariable, 916
 - mo_write_fluxes_states::outputvariable, 911
- writetimestep
 - mo_mrm_write_fluxes_states, 444
 - mo_mrm_write_fluxes_states::outputdataset, 904
 - mo_write_fluxes_states, 772
 - mo_write_fluxes_states::outputdataset, 900
- writevariableattributes
 - mo_mrm_write_fluxes_states, 444
 - mo_write_fluxes_states, 773
- writevariabletimestep
 - mo_mrm_write_fluxes_states, 445
 - mo_mrm_write_fluxes_states::outputvariable, 913
 - mo_write_fluxes_states, 774
 - mo_write_fluxes_states::outputvariable, 908
- writing
 - mo_mrm_write_fluxes_states::outputvariable, 916
 - mo_write_fluxes_states::outputvariable, 911
- written
 - mo_mrm_write_fluxes_states::outputdataset, 906
 - mo_write_fluxes_states::outputdataset, 902
- xcoord
 - mo_global_variables, 176
- xllcorner
 - mo_global_variables::gridgeoref, 827
 - mo_mrm_global_variables::gridgeoref, 828
- xor4096d_0d
 - mo_xor4096, 776
 - mo_xor4096::xor4096, 973
- xor4096d_1d
 - mo_xor4096, 776
 - mo_xor4096::xor4096, 973
- xor4096f_0d
 - mo_xor4096, 776
 - mo_xor4096::xor4096, 973
- xor4096f_1d
 - mo_xor4096, 776
 - mo_xor4096::xor4096, 973
- xor4096gd_0d
 - mo_xor4096, 777
 - mo_xor4096::xor4096g, 974
- xor4096gd_1d
 - mo_xor4096, 777
 - mo_xor4096::xor4096g, 975
- xor4096gf_0d
 - mo_xor4096, 777
 - mo_xor4096::xor4096g, 975
- xor4096gf_1d
 - mo_xor4096, 777
 - mo_xor4096::xor4096g, 975
- xor4096l_0d
 - mo_xor4096, 777
 - mo_xor4096::xor4096, 973
- xor4096l_1d
 - mo_xor4096, 778
 - mo_xor4096::xor4096, 974
- xor4096s_0d
 - mo_xor4096, 778
 - mo_xor4096::xor4096, 974
- xor4096s_1d
 - mo_xor4096, 778
 - mo_xor4096::xor4096, 974
- xtype
 - mo_ncwrite::attribute, 788
 - mo_ncwrite::variable, 972
- ycoord
 - mo_global_variables, 176
- year_counter
 - mo_mrm_write, 436
- yeardays
 - mo_mhm_constants, 254
- yearmonths
 - mo_mhm_constants, 254
- yearmonths_i4
 - mo_mhm_constants, 254
- yend
 - mo_common_variables::period, 924
- yllcorner
 - mo_global_variables::gridgeoref, 827
 - mo_mrm_global_variables::gridgeoref, 828

ystart
 mo_common_variables::period, [925](#)

zeroflowratio
 mo_mrm_signatures, [422](#)

zroots_unity_dp
 mo_corr, [107](#)

zroots_unity_sp
 mo_corr, [108](#)