

COMMUNITY LAND MODEL VERSION 3.0 (CLM3.0) DEVELOPER'S GUIDE

June 2004

Prepared by
Forrest Hoffman
Mariana Vertenstein
Peter Thornton
Keith Oleson
Samuel Levis

DOCUMENT AVAILABILITY

Reports produced after January 1, 1996, are generally available free via the U.S. Department of Energy (DOE) Information Bridge:

Web Site: <http://www.osti.gov/bridge>

Reports produced before January 1, 1996, may be purchased by members of the public from the following source:

National Technical Information Service
5285 Port Royal Road
Springfield, VA 22161
Telephone: 703-605-6000 (1-800-553-6847)
TDD: 703-487-4639
Fax: 703-605-6900
E-mail: info@ntis.fedworld.gov
Web site: <http://www.ntis.gov/support/ordernowabout.htm>

Reports are available to DOE employees, DOE contractors, Energy Technology Data Exchange (ETDE), and International Nuclear Information System (INIS) representatives from the following sources:

Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831
Telephone: 865-576-8401
Fax: 865-576-5728
E-mail: reports@adonis.osti.gov
Web site: <http://www.osti.gov/contact.html>

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

COMMUNITY LAND MODEL VERSION 3.0 (CLM3.0) DEVELOPER'S GUIDE

Forrest Hoffman
Mariana Vertenstein
Peter Thornton
Keith Oleson
Samuel Levis

Date Published: June 2004

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831-6008
managed by
UT-Battelle, LLC
for the
U.S. DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725
and
NATIONAL CENTER FOR ATMOSPHERIC RESEARCH
Climate and Global Dynamics Division
Boulder, Colorado 80307-3000
operated by the
University Corporation for Atmospheric Research
sponsored by the
NATIONAL SCIENCE FOUNDATION

Contents

1	Introduction	1
2	Code Structure	3
2.1	Source Code Organization	3
2.2	Calling Tree	3
2.3	Operating Modes	6
2.4	Initialization	7
3	Data Structures	9
3.1	Hierarchy of Grid Scales	9
3.1.1	Adding Quantities to the CLM Data Structures	11
3.2	Decomposition and Clumps	11
3.3	Filters	12
3.4	Loop Composition	12
3.5	Averaging from a Subgrid Level to an Encompassing Subgrid Level	14
4	Input and Output Files	15
4.1	Surface Data Input	15
4.2	History Dataset Fields	15
4.2.1	History Output on the Model Grid	16
4.2.2	History Output on the RTM Grid	17
4.3	Restart Datasets	18
4.4	Initial Datasets	20
5	Error Conditions	25
5.1	Energy and water balance errors	25
A	ProTeX Source Code Documentation	27
A.1	Module BalanceCheckMod (File: BalanceCheckMod.F90)	28
A.1.1	BalanceCheck	28
A.2	Module BareGroundFluxesMod (File: BareGroundFluxesMod.F90)	31
A.2.1	BareGroundFluxes	31
A.3	Module Biogeophysics1Mod (File: Biogeophysics1Mod.F90)	35
A.3.1	Biogeophysics1	35
A.4	Module Biogeophysics2Mod (File: Biogeophysics2Mod.F90)	39
A.4.1	Biogeophysics2	39
A.5	Module BiogeophysicsLakeMod (File: BiogeophysicsLakeMod.F90)	43
A.5.1	BiogeophysicsLake	43
A.6	Module CanopyFluxesMod (File: CanopyFluxesMod.F90)	47

A.6.1	CanopyFluxes	47
A.6.2	Stomata	50
A.7	Module DGVMAllocationMod (File: DGVMAllocationMod.F90)	53
A.7.1	Allocation	53
A.8	Module DGVMEcosystemDynMod (File: DGVMEcosystemDynMod.F90)	57
A.8.1	DGVMEcosystemDynini	57
A.8.2	DGVMEcosystemDyn	58
A.8.3	DGVMRrespiration	59
A.8.4	Phenology	60
A.8.5	FireSeason	61
A.8.6	LitterSOM	62
A.9	Module DGVMEstablishmentMod (File: DGVMEstablishmentMod.F90)	65
A.9.1	Establishment	65
A.10	Module FireMod (File: DGVMFireMod.F90)	67
A.10.1	Fire	67
A.11	Module KillMod (File: DGVMKillMod.F90)	69
A.11.1	Kill	69
A.12	Module LightMod (File: DGVMLightMod.F90)	71
A.12.1	Light	71
A.13	Module DGVMMod (File: DGVMMod.F90)	73
A.13.1	lpj	73
A.13.2	lpjreset1	74
A.13.3	lpjreset2	75
A.13.4	histDGVM	76
A.13.5	resetTimeConstDGVM	77
A.13.6	resetWeightsDGVM	77
A.13.7	gatherWeightsDGVM	78
A.13.8	set_dgvm_filename	79
A.13.9	BuildNatVegFilter	80
A.14	Module MortalityMod (File: DGVM MortalityMod.F90)	81
A.14.1	Mortality	81
A.15	Module ReproductionMod (File: DGVMReproductionMod.F90)	83
A.15.1	Reproduction	83
A.16	Module restDGVMMod (File: DGVMRestMod.F90)	85
A.16.1	restart_dgvm	85
A.17	Module TurnoverMod (File: DGVMTurnoverMod.F90)	87
A.17.1	Turnover	87
A.18	Module DriverInitMod (File: DriverInitMod.F90)	89
A.18.1	DriverInit	89
A.19	Module FracWetMod (File: FracWetMod.F90)	91
A.19.1	FracWet	91
A.20	Module FrictionVelocityMod (File: FrictionVelocityMod.F90)	93
A.20.1	FrictionVelocity	93
A.20.2	StabilityFunc	94
A.20.3	StabilityFunc2	95
A.20.4	MoninObukIni	95
A.21	Module Hydrology1Mod (File: Hydrology1Mod.F90)	97
A.21.1	Hydrology1	97
A.22	Module Hydrology2Mod (File: Hydrology2Mod.F90)	101
A.22.1	Hydrology2	101

A.23 Module HydrologyLakeMod (File: HydrologyLakeMod.F90)	103
A.23.1 HydrologyLake	103
A.24 Module QSatMod (File: QSatMod.F90)	105
A.24.1 QSat	105
A.25 Module RtmMod (File: RtmMod.F90)	107
A.25.1 Rtmgridini	107
A.25.2 Rtmlandini	108
A.25.3 Rtmfluxini()	109
A.25.4 Rtmriverflux	109
A.25.5 UpdateInput	110
A.25.6 Rtm	111
A.25.7 UpdateGlobal	111
A.25.8 restart_rtm	112
A.26 Module RunoffMod (File: RunoffMod.F90)	113
A.26.1 set_roflnd	113
A.26.2 set_rofocn	114
A.26.3 set_proc_rof_bounds	114
A.26.4 UpdateRunoff	115
A.26.5 get_proc_rof_total	116
A.26.6 get_proc_rof_bounds	116
A.26.7 get_proc_rof_global	117
A.27 Module STATICEcosysDynMod (File: STATICEcosysDynMod.F90)	119
A.27.1 EcosystemDynini	119
A.27.2 EcosystemDyn	120
A.27.3 interpMonthlyVeg	120
A.27.4 readMonthlyVegetation	121
A.28 Module SnowHydrologyMod (File: SnowHydrologyMod.F90)	123
A.28.1 SnowWater	123
A.28.2 SnowCompaction	124
A.28.3 CombineSnowLayers	125
A.28.4 DivideSnowLayers	126
A.28.5 Combo	127
A.28.6 BuildSnowFilter	127
A.29 Module SoilHydrologyMod (File: SoilHydrologyMod.F90)	129
A.29.1 SurfaceRunoff	129
A.29.2 Infiltration	130
A.29.3 SoilWater	131
A.29.4 Drainage	133
A.30 Module SoilTemperatureMod (File: SoilTemperatureMod.F90)	135
A.30.1 SoilTemperature	135
A.30.2 SoilThermProp	136
A.30.3 PhaseChange	138
A.31 Module SurfaceAlbedoMod (File: SurfaceAlbedoMod.F90)	141
A.31.1 SurfaceAlbedo	141
A.31.2 SnowAlbedo	142
A.31.3 SoilAlbedo	143
A.31.4 TwoStream	144
A.31.5 SnowAge	145
A.32 Module SurfaceRadiationMod (File: SurfaceRadiationMod.F90)	147
A.32.1 SurfaceRadiation	147

A.33 Module TridiagonalMod (File: TridiagonalMod.F90)	149
A.33.1 Tridiagonal	149
A.34 Module VOCEmissionMod (File: VOCEmissionMod.F90)	151
A.34.1 VOCEmission	151
A.35 Module abortutils (File: abortutils.F90)	153
A.35.1 endrun	153
A.36 Module accFldsMod (File: accFldsMod.F90)	155
A.36.1 initAccFlds()	155
A.36.2 updateAccFlds	156
A.36.3 initAccClmtype	157
A.37 Module accumulMod (File: accumulMod.F90)	159
A.37.1 init_accum_field	159
A.37.2 print_accum_fields	160
A.37.3 extract_accum_field_sl	161
A.37.4 extract_accum_field_ml	161
A.37.5 update_accum_field_sl	162
A.37.6 update_accum_field_ml	162
A.37.7 restart_accum	163
A.38 Module areaMod (File: areaMod.F90)	165
A.38.1 areaini	165
A.38.2 areaave	167
A.38.3 areamap	168
A.38.4 areaovr	169
A.38.5 cellarea_regional	170
A.38.6 cellarea_global	171
A.38.7 celledge_regional	171
A.38.8 celledge_global	172
A.38.9 areaini_point	173
A.38.10 areamap_point	175
A.38.11 areaovr_point	176
A.38.12 mkmxovr	177
A.39 Module atm_lndMod (File: atm_lndMod.F90)	181
A.39.1 atm_lnd_ini	181
A.39.2 atm_lnd_drv	182
A.40 Module atmdrvMod (File: atmdrvMod.F90)	185
A.40.1 atmdrv	185
A.40.2 atm_getgrid	186
A.40.3 atm_openfile	187
A.40.4 atm_readdata	187
A.40.5 interpa2si	188
A.40.6 interpa2s	189
A.41 Module clm_csmMod (File: clm_csmMod.F90)	191
A.41.1 csm_setup	192
A.41.2 csm_shutdown	192
A.41.3 csm_initialize	192
A.41.4 csm_recvgrid	193
A.41.5 csm_sendalb	194
A.41.6 csm_dosndrcv	194
A.41.7 csm_recv	195
A.41.8 csm_send	195

A.41.9	csm_fxave	196
A.41.10	compat_check	196
A.41.11	csm_compat	197
A.41.12	restart_coupler	197
A.41.13	global_sum_fld2d	198
A.41.14	global_sum_fld1d	198
A.42	Module clm_varcon (File: clm_varcon.F90)	201
A.43	Module clm_varctl (File: clm_varctl.F90)	203
A.44	Module clm_varpar (File: clm_varpar.F90)	205
A.45	Module clm_varsur (File: clm_varsur.F90)	207
A.45.1	varsur_alloc	207
A.45.2	varsur_dealloc	208
A.46	Module clmtype (File: clmtype.F90)	209
A.47	Module clmtypeInitMod (File: clmtypeInitMod.F90)	233
A.47.1	initClmtype	234
A.47.2	init_pft_type	234
A.47.3	init_column_type	235
A.47.4	init_landunit_type	235
A.47.5	init_gridcell_type	235
A.47.6	init_energy_balance_type	236
A.47.7	init_water_balance_type	236
A.47.8	init_pft_pstate_type	237
A.47.9	init_pft_ecophys_constants	237
A.47.10	init_pft_DGVMecophys_constants	237
A.47.11	init_pft_energy_type	238
A.47.12	init_pft_wstate_type	238
A.47.13	init_pft_pdgvstate_type	238
A.47.14	init_pft_elflux_type	239
A.47.15	init_pft_mflux_type	239
A.47.16	init_pft_wflux_type	240
A.47.17	init_pft_cflux_type	240
A.47.18	init_pft_vflux_type	240
A.47.19	init_pft_dflux_type	241
A.47.20	init_column_pstate_type	241
A.47.21	init_column_estate_type	242
A.47.22	init_column_wstate_type	242
A.47.23	init_column_cstate_type	242
A.47.24	init_column_elflux_type	243
A.47.25	init_column_wflux_type	243
A.47.26	init_landunit_pstate_type	244
A.47.27	init_gridcell_dgvstate_type	244
A.47.28	init_gridcell_pstate_type	244
A.47.29	init_atm2lnd_state_type	245
A.47.30	init_lnd2atm_state_type	245
A.47.31	init_atm2lnd_flux_type	246
A.47.32	init_lnd2atm_flux_type	246
A.47.33	init_gridcell_wflux_type	246
A.48	Module controlMod (File: controlMod.F90)	247
A.48.1	control_init	249
A.48.2	control_spmc	250

A.48.3	control_print	250
A.49	Module decompMod (File: decompMod.F90)	253
A.49.1	initDecomp	254
A.49.2	get_nclumps	255
A.49.3	get_clump_cell_id_coord	255
A.49.4	get_clump_owner_id	256
A.49.5	get_clump_ncells_proc	256
A.49.6	get_clump_ncells_id	257
A.49.7	get_clump_coord_id	257
A.49.8	get_gcell_info	258
A.49.9	get_clump_gcell_info	259
A.49.10	get_clump_bounds	259
A.49.11	get_proc_bounds	260
A.49.12	get_proc_total	260
A.49.13	get_proc_global	261
A.49.14	get_proc_clumps	261
A.49.15	get_gcell_xyind	261
A.49.16	map_dc2sn_sl_real	262
A.49.17	map_dc2sn_sl_int	262
A.49.18	map_dc2sn_ml1_real	263
A.49.19	map_dc2sn_ml1_int	263
A.49.20	map_sn2dc_sl_real	264
A.49.21	map_sn2dc_sl_int	264
A.49.22	map_sn2dc_ml1_real	265
A.49.23	map_sn2dc_ml1_int	265
A.49.24	get_sn_land1d	266
A.49.25	get_sn_cols1d	266
A.49.26	get_sn_pfts1d	267
A.50	do_close_dispose (File: do_close_dispose.F90)	269
A.51	do_restwrite (File: do_restwrite.F90)	271
A.52	driver (File: driver.F90)	273
A.52.1	write_diagnostic	276
A.53	Module fileutils (File: fileutils.F90)	277
A.53.1	get_filename	277
A.53.2	set_filename	278
A.53.3	getfil	278
A.53.4	putfil	278
A.53.5	opnfil	279
A.53.6	getavu	279
A.53.7	relavu	280
A.53.8	shell_cmd	281
A.54	Module filterMod (File: filterMod.F90)	283
A.54.1	initFilters	283
A.55	getdatetime (File: getdatetime.F90)	285
A.56	Module histFileMod (File: histFileMod.F90)	287
A.56.1	masterlist_printflds	288
A.56.2	masterlist_addfld	289
A.56.3	htapes_build	290
A.56.4	masterlist_make_active	290
A.56.5	masterlist_change_timeavg	291

A.56.6	htapes_fieldlist	291
A.56.7	htape_addfld	291
A.56.8	update_hbuf	292
A.56.9	update_hbuf_field	292
A.56.10	hfields_normalize	293
A.56.11	hfields_zero	293
A.56.12	htape_create	294
A.56.13	htape_timeconst	294
A.56.14	hfields_write	295
A.56.15	hfields_1dinfo	295
A.56.16	htapes_wrapup	296
A.56.17	restart_history	297
A.56.18	getname	297
A.56.19	getflag	298
A.56.20	list_index	298
A.56.21	set_hist_filename	298
A.56.22	add_fld1d	299
A.56.23	add_fld2d	300
A.56.24	pointer_index	301
A.56.25	add_subscript	301
A.57	Module histFldsMod (File: histFldsMod.F90)	303
A.57.1	initHistFlds	303
A.58	iniTimeConst (File: iniTimeConst.F90)	305
A.59	iniTimeVar (File: iniTimeVar.F90)	307
A.59.1	mkarbinit	308
A.60	Module inicFileMod (File: inicFileMod.F90)	311
A.60.1	inicfile	311
A.60.2	inicfields	312
A.60.3	inicperp	313
A.60.4	do_inicwrite	313
A.60.5	set_init_filename	314
A.61	Module initGridcellsMod (File: initGridCellsMod.F90)	315
A.61.1	initGridcells	315
A.61.2	landunit_veg_compete	316
A.61.3	landunit_veg_noncompete	317
A.61.4	landunit_special	317
A.61.5	landunit_crop_noncompete	318
A.61.6	initGridcellsGlob	318
A.62	Module initializeMod (File: initializeMod.F90)	321
A.62.1	initialize	321
A.62.2	header	323
A.63	Module iobinary (File: iobinary.F90)	325
A.63.1	readin_1d_array_int	325
A.63.2	readin_1darray_real	326
A.63.3	readin_2d_arrayint	326
A.63.4	readin_2darray_real	327
A.63.5	wrtout_1d_array_int	327
A.63.6	wrtout_1darray_real	327
A.63.7	wrtout_2d_array_int	328
A.63.8	wrtout_2darray_real	328

A.63.9	getnum	329
A.64	Module lnd2atmMod (File: lnd2atmMod.F90)	331
A.64.1	lnd2atm	331
A.64.2	makel2a	331
A.65	Module lp_coupling (File: lp_coupling.F90)	333
A.65.1	lp_coupling_init	333
A.65.2	lp_coupling_finalize	334
A.65.3	alltoall_clump_to_chunk_init	334
A.65.4	alltoall_clump_to_chunk	335
A.65.5	alltoall_chunk_to_clump	335
A.65.6	mkglacier	337
A.66	Module mkgridMod (File: mkgridMod.F90)	339
A.66.1	mkgrid_offline	339
A.66.2	read_grid_offline	340
A.66.3	create_grid_offline	340
A.66.4	mkgrid_cam	341
A.67	Module mklai (File: mklai.F90)	343
A.67.1	mklai	343
A.68	mklanwat (File: mklanwat.F90)	345
A.69	Module mkpft (File: mkpft.F90)	347
A.69.1	mkpfts	347
A.70	mkrank (File: mkrank.F90)	349
A.71	mksoicol (File: mksoicol.F90)	351
A.72	mksoitex (File: mksoitex.F90)	353
A.73	Module mksrfdatMod (File: mksrfdatMod.F90)	355
A.73.1	mksrfdat	355
A.74	mkurban (File: mkurban.F90)	357
A.75	Module mpiinc (File: mpiinc.F90)	359
A.76	Module nanMod (File: nanMod.F90)	361
A.77	Module ncdioMod (File: ncdio.F90)	363
A.77.1	check_dim	363
A.77.2	check_var	364
A.77.3	check_ret	364
A.77.4	ncd_defvar	364
A.77.5	ncd_iolocal_int_1d	365
A.77.6	ncd_iolocal_real_1d	366
A.77.7	ncd_iolocal_int_2d	366
A.77.8	ncd_iolocal_real_2d	367
A.77.9	ncd_ioglobal_int_var	368
A.77.10	ncd_ioglobal_real_var	368
A.77.11	ncd_ioglobal_int_1d	369
A.77.12	ncd_ioglobal_real_1d	369
A.77.13	ncd_ioglobal_int_2d	370
A.77.14	ncd_ioglobal_real_2d	370
A.77.15	ncd_ioglobal_int_3d	371
A.77.16	ncd_ioglobal_real_3d	371
A.77.17	get_size_dim1	372
A.77.18	subroutine scam_field_offsets	372
A.78	Module pft2colMod (File: pft2colMod.F90)	375
A.78.1	pft2col	375

A.79 Module pftvarcon (File: pftvarcon.F90)	377
A.79.1 pftconrd	378
A.80 program_csm (File: program_csm.F90)	379
A.81 program_off (File: program_off.F90)	381
A.82 Module restFileMod (File: restFileMod.F90)	383
A.82.1 restart	383
A.82.2 restart_setup	384
A.82.3 restart_time	384
A.82.4 restart_biogeophys	385
A.82.5 restart_wrapup	386
A.82.6 write_rest_pfile	386
A.82.7 set_restart_filename	387
A.83 snowdp2lev (File: snowdp2lev.F90)	389
A.84 Module spmdGathScatMod (File: spmdGathScatMod.F90)	391
A.84.1 spmd_compute_mpigs	391
A.84.2 scatter_1darray_int	392
A.84.3 scatter_1darray_real	392
A.84.4 scatter_2darray_int	393
A.84.5 scatter_2darray_real	393
A.84.6 gather_1darray_int	394
A.84.7 gather_1darray_real	394
A.84.8 gather_2darray_int	394
A.84.9 gather_2darray_real	395
A.84.10 allgather_1darray_int	395
A.84.11 allgather_1darray_real	396
A.84.12 allgather_2darray_int	396
A.84.13 allgather_2darray_real	396
A.85 Module spmdMod (File: spmdMod.F90)	399
A.85.1 spmd_init	399
A.86 Module subgridAveMod (File: subgridAveMod.F90)	401
A.86.1 p2c_1d	402
A.86.2 p2c_2d	402
A.86.3 p2c_1d_filter	403
A.86.4 p2c_2d_filter	403
A.86.5 p2l_1d	404
A.86.6 p2l_2d	404
A.86.7 p2g_1d	405
A.86.8 p2g_2d	405
A.86.9 c2l_1d	406
A.86.10 c2l_2d	406
A.86.11 c2g_1d	407
A.86.12 c2g_2d	407
A.86.13 l2g_1d	408
A.86.14 l2g_2d	408
A.87 Module surfFileMod (File: surfFileMod.F90)	409
A.87.1 surfrd	409
A.88 system_cmd (File: system_cmd.c)	413
A.89 Module system_messages (File: system_messages.F90)	415
A.89.1 allocation_err	415
A.89.2 netcdf_err	415

A.89.3 iobin_err	416
A.90 Module time_manager (File: time_manager.F90)	417
A.90.1 timemgr_init	418
A.90.2 timemgr_restart	419
A.90.3 timemgr_print	419
A.90.4 advance_timestep	420
A.90.5 get_step_size	420
A.90.6 get_nstep	421
A.90.7 get_curr_date	422
A.90.8 get_prev_date	422
A.90.9 get_start_date	423
A.90.10 get_ref_date	423
A.90.11 get_curr_time	424
A.90.12 get_curr_calday	425
A.90.13 is_end_curr_day	425
A.90.14 is_end_curr_month	426
A.90.15 is_first_step	426
A.90.16 is_first_restart_step	427
A.90.17 is_last_step	427
A.90.18 timemgr_write_restart	428
A.90.19 timemgr_read_restart	428
A.90.20 chkrc	429
A.90.21 to_upper	429

Chapter 1

Introduction

This document describes the guidelines adopted for software development of the Community Land Model (CLM) and serves as a reference to the entire code base of the released version of the model. The version of the code described here is Version 3.0 which was released in the summer of 2004. This document, the *Community Land Model Version 3.0 (CLM3.0) User's Guide* (Vertenstein et al., 2004), the *Technical Description of the Community Land Model (CLM)* (Oleson et al., 2004), and the *Community Land Model's Dynamic Global Vegetation Model (CLM-DGVM): Technical Description and User's Guide* (Levis et al., 2004) provide the developer, user, or researcher with details of implementation, instructions for using the model, a scientific description of the model, and a scientific description of the Dynamic Global Vegetation Model integrated with CLM respectively.

The CLM is a single column (snow-soil-vegetation) biogeophysical model of the land surface which can be run serially (on a laptop or personal computer) or in parallel (using distributed or shared memory processors or both) on both vector and scalar computer architectures. Written in Fortran 90, CLM can be run offline (*i.e.*, run in isolation using stored atmospheric forcing data), coupled to an atmospheric model (*e.g.*, the Community Atmosphere Model (CAM)), or coupled to a climate system model (*e.g.*, the Community Climate System Model Version 3 (CCSM3)) through a flux coupler (*e.g.*, Coupler 6 (CPL6)). When coupled, CLM exchanges fluxes of energy, water, and momentum with the atmosphere.

The horizontal land surface heterogeneity is represented by a nested subgrid hierarchy composed of gridcells, landunits, columns, and plant functional types (PFTs). This hierarchical representation is reflected in the data structures used by the model code. Biophysical processes are simulated for each subgrid unit (landunit, column, and PFT) independently, and prognostic variables are maintained for each subgrid unit. Vertical heterogeneity is represented by a single vegetation layer, 10 layers for soil, and up to five layers for snow, depending on the snow depth.

For computational efficiency, gridcells are grouped into “clumps” which are divided in cyclic fashion among distributed memory processors. Additional parallel performance is obtained by distributing clumps of gridcells across shared memory processors on computer platforms that support hybrid Message Passing Interface (MPI)/OpenMP operation.

Significant modifications to the source code have been made over the last year to support efficient operation on newer vector architectures, specifically the Earth Simulator in Japan and the Cray X1 at Oak Ridge National Laboratory (Hoffman et al., 2004). These code modifications resulted in performance improvements even on the scalar architectures widely used for running CLM presently.

To better support vectorized processing in the code, subgrid units (columns and PFTs) are grouped into “filters” based on their process-specific categorization. For example, filters (vectors of integers) referring to all snow, non-snow, lake, non-lake, and soil covered columns and PFTs within each clump are built and maintained when the model is run. Many loops within the scientific subroutines use these filters to indirectly address the process-appropriate subgrid units.

Chapter 2

Code Structure

2.1 Source Code Organization

The CLM source code files are organized into separate directories below the *src/* directory based on their scientific functionality. The following directories contain individual algorithms for land surface physical and chemical processes, river routing, data structure creation and model control, and surface dataset construction.

biogeochem/ Contains modules for static ecosystem dynamics, the new Dynamic Global Vegetation Model (DGVM) (used optionally to simulate global biogeography through establishment, resource competition, growth, mortality, and fire parameterizations based on the Lund-Potsdam-Jena (LPJ) scheme), and volatile organic compound (VOC) emissions.

biogeophys/ Contains modules for calculating leaf temperature, surface fluxes, soil and snow hydrology, and soil, snow, and ground temperatures.

main/ Contains the data structure definition and initialization modules, the main driver routines, parallel domain decomposition and mapping code, wrappers for lower-level libraries, and input/output (I/O) and history output routines.

mkshrdata/ Contains modules for constructing surface datasets.

riverroute/ Contains the code for transporting model runoff to the ocean using the University of Texas at Austin's River Transport Model (RTM) ([Branstetter and Famiglietti, 1999](#); [Branstetter, 2001](#)).

2.2 Calling Tree

The following is a brief outline of the calling sequence for the main CLM **driver** routine contained in *driver.F90* (Section [A.52](#)). Conditionally-executed routines are denoted by the inclusion of the associated pre-processor variable in brackets after the routine name. An exclamation point preceding the variable reverses its meaning. The comprehensive listing of source code prologs contained in Appendix [A](#) provides a brief synopsis of the functionality provided by each of the routines listed below.

1. `csmdosndrcv` (Section [A.41.6](#)) [`COUP_CSM`]
2. `csmdrcv` (Section [A.41.7](#)) [`COUP_CSM`]
3. `interpMonthlyVeg` (Section [A.27.3](#)) [`!DGVM`]

- 3.1. readMonthlyVegetation (Section [A.27.4](#))
- 4. get_proc_clumps (Section [A.49.14](#))
- 5. **Begin Loop 1 Over Clumps**
 - 5.1. get_clump_bounds (Section [A.49.10](#))
 - 5.2. DriverInit (Section [A.18.1](#))
 - 5.3. Hydrology1 (Section [A.21.1](#))
 - 5.3.1. FracWet (Section [A.19.1](#))
 - 5.3.2. p2c (Section [A.86](#))
 - 5.4. SurfaceRadiation (Section [A.32.1](#))
 - 5.5. Biogeophysics1 (Section [A.3.1](#))
 - 5.5.1. QSat (Section [A.24.1](#))
 - 5.6. BareGroundFluxes (Section [A.2.1](#))
 - 5.6.1. MoninObukIni (Section [A.20.4](#))
 - 5.6.2. FrictionVelocity (Section [A.20.1](#))
 - 5.7. CanopyFluxes (Section [A.6.1](#))
 - 5.7.1. QSat (Section [A.24.1](#))
 - 5.7.2. MoninObukIni (Section [A.20.4](#))
 - 5.7.3. FrictionVelocity (Section [A.20.1](#))
 - 5.7.4. Stomata (Section [A.6.2](#))
 - 5.7.5. Stomata (Section [A.6.2](#))
 - 5.7.6. QSat (Section [A.24.1](#))
 - 5.8. BiogeophysicsLake (Section [A.5.1](#))
 - 5.8.1. QSat (Section [A.24.1](#))
 - 5.8.2. MoninObukIni (Section [A.20.4](#))
 - 5.8.3. FrictionVelocity (Section [A.20.1](#))
 - 5.8.4. QSat (Section [A.24.1](#))
 - 5.8.5. Tridiagonal (Section [A.33.1](#))
 - 5.9. VOCEmission (Section [A.34.1](#)) [VOC]
 - 5.10. DGVMRespiration (Section [A.8.3](#)) [DGVM]
 - 5.11. DGVMEcosystemDyn (Section [A.8.2](#)) [DGVM]
 - 5.11.1. Phenology (Section [A.8.4](#))
 - 5.11.2. FireSeason (Section [A.8.5](#))
 - 5.11.3. LitterSOM (Section [A.8.6](#))
 - 5.12. EcosystemDyn (Section [A.27.2](#)) [!DGVM]
 - 5.13. SurfaceAlbedo (Section [A.31.1](#))
 - 5.13.1. SnowAlbedo (Section [A.31.2](#))
 - 5.13.2. SoilAlbedo (Section [A.31.3](#))
 - 5.13.3. TwoStream (Section [A.31.4](#))
 - 5.14. Biogeophysics2 (Section [A.4.1](#))

- 5.14.1. SoilTemperature (Section A.30.1)
- 5.14.2. p2c (Section A.86)
- 6. csm_flgave (Section A.41.9) [COUP_CSM]
- 7. csm_send (Section A.41.8) [COUP_CSM]
- 8. **Begin Loop 2 Over Clumps**
 - 8.1. get_clump_bounds (Section A.49.10)
 - 8.2. Hydrology2 (Section A.22.1)
 - 8.2.1. BuildSnowFilter (Section A.28.6)
 - 8.2.2. SnowWater (Section A.28.1)
 - 8.2.3. SurfaceRunoff (Section A.29.1)
 - 8.2.4. Infiltration (Section A.29.2)
 - 8.2.5. SoilWater (Section A.29.3)
 - 8.2.6. Drainage (Section A.29.4)
 - 8.2.7. SnowCompaction (Section A.28.2)
 - 8.2.8. CombineSnowLayers (Section A.28.3)
 - 8.2.9. DivideSnowLayers (Section A.28.4)
 - 8.2.10. BuildSnowFilter (Section A.28.6)
 - 8.3. HydrologyLake (Section A.23.1)
 - 8.4. SnowAge (Section A.31.5)
 - 8.5. BalanceCheck (Section A.1.1)
- 9. lnd2atm (Section A.64.1) [OFFLINE]
- 10. get_nstep (Section A.90.6)
- 11. write_diagnostic (Section A.52.1)
- 12. Rtmriverflux (Section A.25.4) [RTM]
- 13. updateAccFlds (Section A.36.2)
- 14. update_hbuf (Section A.56.8)
- 15. get_curr_date (Section A.90.7) [DGVM]
- 16. **Begin Loop Over Clumps** [DGVM]
 - 16.1. get_clump_bounds (Section A.49.10)
 - 16.2. lpj (Section A.13.1)
 - 16.2.1. get_curr_date (Section A.90.7)
 - 16.2.2. get_ref_date (Section A.90.10)
 - 16.2.3. BuildNatVegFilter (Section A.13.9)
 - 16.2.4. Reproduction (Section A.15.1)
 - 16.2.5. Turnover (Section A.17.1)
 - 16.2.6. Kill (Section A.11.1)
 - 16.2.7. BuildNatVegFilter (Section A.13.9)

- 16.2.8. Allocation (Section A.7.1)
- 16.2.9. Light (Section A.12.1)
- 16.2.10. Mortality (Section A.14.1)
- 16.2.11. Fire (Section A.10.1)
- 16.2.12. Establishment (Section A.9.1)
- 16.3. lpjreset1 (Section A.13.2)
 - 16.3.1. DGVMecosystemDyn (Section A.8.2)
 - 16.3.2. SurfaceAlbedo (Section A.31.1)
 - 16.3.2.1. SnowAlbedo (Section A.31.2)
 - 16.3.2.2. SoilAlbedo (Section A.31.3)
 - 16.3.2.3. TwoStream (Section A.31.4)
 - 16.3.3. resetTimeConstDGVM (Section A.13.5)
- 17. htapes_wrapup (Section A.56.16)
- 18. **Begin Loop Over Clumps** [DGVM]
 - 18.1. get_clump_bounds (Section A.49.10)
 - 18.2. lpjreset2 (Section A.13.3)
 - 18.2.1. c2g (Section A.86)
 - 18.2.2. c2g (Section A.86)
 - 18.2.3. resetWeightsDGVM (Section A.13.6)
 - 18.2.4. c2g (Section A.86)
 - 18.2.5. c2g (Section A.86)
- 19. gatherWeightsDGVM (Section A.13.7) [DGVM] [SPMD]
- 20. histDGVM (Section A.13.4) [DGVM]
- 21. restart (Section A.82.1)
- 22. inicfile (Section A.60.1)

2.3 Operating Modes

The CLM can be built to run in one of three modes. In *offline mode*, it runs as a stand alone executable periodically reading stored atmospheric forcing data. In *cam mode*, CLM is compiled and run as a part of the Community Atmosphere Model (CAM). Communication between the atmosphere and land models occurs via variables passed in subroutine calls. In *csm mode*, CLM is run as a component in a group of climate system models making up the Community Climate System Model (CCSM). In this mode, all component models—including CLM—are compiled as separate executables, but run simultaneously communicating state and flux information via the flux coupler.

offline mode: The routine `program_off` (Section A.81) provides the program interface for running CLM in offline mode. This routine initializes timers, MPI communications, and the land model. As a part of model initialization, it calls `shr_orb_params` to determine planetary orbital parameters. Next, the time stepping loop of the model is executed. Inside this loop, the atmospheric forcing data is read, the `driver` (Section A.52) routine is called, and the time step is advanced. After the time stepping loop is completed, the timers and MPI communications are finalized and the program ends.

cam mode: The module `atm_lndMod` (Section A.39) contains the subroutine interfaces necessary to run CLM in cam mode. CLM is initialized by a call to `atm_lnd_ini` (Section A.39.1). At every time step in the atmosphere model, CLM is run by calling `atm_lnd_drv` (Section A.39.2). These routines handle the exchange of flux and state variables between the atmosphere and the land by communicating blocks of data using a mapping between parallel processes constructed during initialization.

csm mode: The routine `program_csm` (Section A.80) provides the program interface for running CLM in csm mode. This routine initializes timers, MPI communications, and the land model. In csm mode, orbital parameters are obtained from the flux coupler during initialization. Next, the time stepping loop of the model is executed; however, flux and state variable exchanges for each time step occur with the flux coupler instead of directly with the atmosphere model. After the time stepping loop is completed, the timers and MPI communications are finalized and the program ends.

2.4 Initialization

The following is a brief outline of the calling sequence in which CLM is initialized. Since initializing the model requires allocating memory, reading in datasets from disk, and establishing the model decomposition, the order in which initialization steps occur is significant. Model initialization occurs in the routine `initialize` contained in the module `initializeMod` (Section A.62). Conditionally-executed routines are denoted by the inclusion of the associated pre-processor variable in brackets after the routine name. An exclamation point preceding the variable reverses its meaning. The comprehensive listing of source code prologs contained in Appendix A provides a brief synopsis of the functionality provided by each of the routines listed below.

1. `control_init` (Section A.48.1)
2. `varsur_alloc` (Section A.45.1)
3. `timemgr_init` (Section A.90.1) [OFFLINE] or [COUP_CSM]
4. `advance_timestep` (Section A.90.4) [OFFLINE]
5. `Rtmgridini` (Section A.25.1) [RTM]
6. `csm_recvgrid` (Section A.41.4) [COUP_CSM]
7. `pftconrd` (Section A.79.1)
8. `surfrd` (Section A.87.1)
9. `initDecomp` (Section A.49.1)
10. `lp_coupling_init` (Section A.65.1) [COUP_CAM]
11. `initClmtype` (Section A.47.1)
12. `initGridcells` (Section A.61.1)
13. `initFilters` (Section A.54.1)
14. `EcosystemDynini` (Section A.27.1) [!DGVM]
15. `DGVM_EcosystemDynini` (Section A.8.1) [DGVM]
16. `iniTimeConst` (Section A.58)

17. `Rtmlandini` (Section [A.25.2](#)) [RTM]
18. `csm_initialize` (Section [A.41.3](#)) [COUP_CSM]
19. `initHistFlds` (Section [A.57.1](#))
20. `resetTimeConstDGVM` (Section [A.13.5](#)) [DGVM]
21. `atm_getgrid` (Section [A.40.2](#)) [OFFLINE]
22. `initAccClmtype` (Section [A.36.3](#))
23. `iniTimeVar` (Section [A.59](#))
24. `csm_sendalb` (Section [A.41.5](#)) [COUP_CSM]
25. `varsur_dealloc` (Section [A.45.2](#))

Chapter 3

Data Structures

3.1 Hierarchy of Grid Scales

The horizontal land surface heterogeneity is represented by a nested subgrid hierarchy composed of gridcells, landunits, columns, and plant functional types (PFTs) as shown in Figure 3.1. Each gridcell can have a different number of landunits, each landunit can have a different number of columns, and each column can have multiple PFTs. Gridcells represent the computational grid which is shared with the atmospheric physics.

The landunit, the first subgrid level, is intended to capture the broadest spatial pattern of subgrid heterogeneity. It serves primarily to distinguish physical soil properties. Specific landunits include glacier, lake, wetland, urban, and vegetated. The column captures variability in soil and snow state variables within a landunit. Water and energy states and fluxes are tracked at the column level. The PFT, the third subgrid level, captures the characteristic biophysical and biogeochemical functions of broad categories of vegetation and bare soil. Up to four out of the 15 possible PFTs differing in physiology and structure may be contained within a single column.

Biophysical processes are simulated for each subgrid unit independently, and prognostic variables are maintained for each subgrid unit. Processes related to soil and snow require PFT level properties to be aggregated up to the column level. Aggregation is usually accomplished by computing a weighted sum for each quantity over all PFTs within a column. Similarly, different PFTs compete for the resources tracked at the column level. A complete description of the biophysical processes simulated by CLM is available in the *Technical Description of the Community Land Model (CLM)* (Oleson et al., 2004).

The hierarchical subgrid representation is reflected in the data structures used in the model code. These data structures consist of derived data types for each subgrid unit as shown in Figure 3.2. Each of these data types contains arrays of integers which serve as indices to the higher subgrid levels or as initial and final bounds on the lower subgrid levels. For example, the column-level data type contains `landunit` and `gridcell` integer arrays which refer to the appropriate landunit and gridcell for a given column. The column-level data type also contains integer arrays named `pfti`, `pftf`, and `npfts` which refer to the first and last PFTs and the total number of PFTs, respectively, for a given column. Each subgrid data type also contains real arrays for surface areas and area weights at all higher grid levels.

Each grid and subgrid unit data type in the hierarchy also contains a number of physical and chemical state and flux data types. These data types typically contain real arrays for state and flux quantities. Vertical heterogeneity is represented by a single vegetation layer, 10 layers for soil, and up to five layers for snow, depending on the snow depth. Multi-layer quantities are stored as two-dimensional real arrays. Using real arrays at every level in the hierarchy maximizes opportunities for contiguous memory access thereby providing significantly better performance on vector architectures. All arrays contained in derived data types are implemented as Fortran 90 pointers. Memory for these arrays is allocated dynamically during model

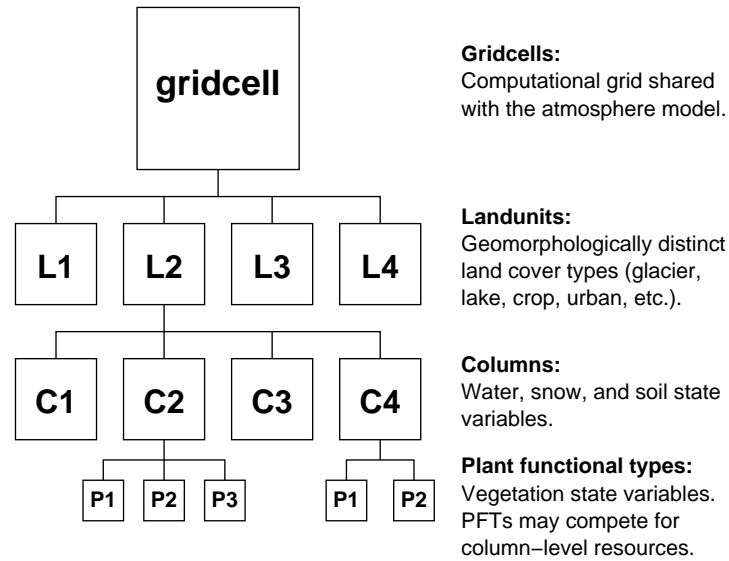
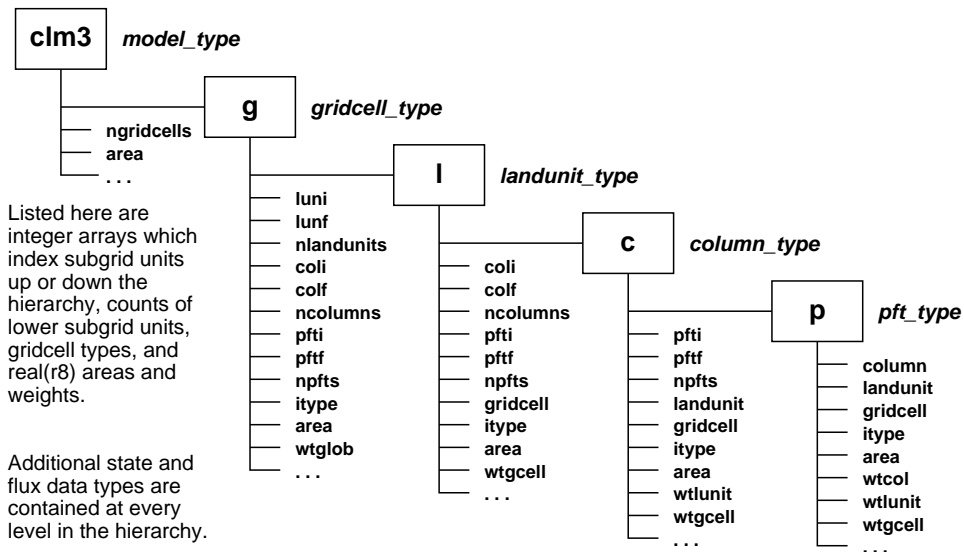


Figure 3.1: The Community Land Model (CLM) subgrid hierarchy

Figure 3.2: The CLM3 data structure hierarchy defined in `clmtype` (Section A.46)

initialization. The data structure hierarchy is defined in `clmtype` (Section A.46), and the structures are allocated and initialized in `clmtypeInitMod` (Section A.47).

3.1.1 Adding Quantities to the CLM Data Structures

Researchers will often wish to add new science subroutines to CLM to perform new simulation experiments or to develop model improvements. Since new science subroutines frequently require new variables for tracking additional quantities throughout simulations, all primary variables are declared and allocated in individual modules. Like the variables described above, new variables must be declared in the module `clmtype` (Section A.46) and allocated and initialized in the module `clmtypeInitMod` (Section A.47).

Within `clmtype`, quantities are arranged into process-specific derived data types. For instance, variables describing the physical state of PFTs are contained in the `pft_pstate_type` data type. New variables describing the physical state of PFTs should be added to this data type, usually as double precision pointers with an unspecified single dimension (*e.g.*, `real(r8)`, `pointer :: varname(:)`). Multi-level quantities would have two unspecified dimensions.

Within `clmtypeInitMod`, variables are allocated and initialized in subroutines having names similar to the derived data type name in which the variables reside. For example, quantities in `pft_pstate_type` are allocated and initialized in `init_pft_pstate_type` (Section A.47.8). A new PFT physical state variable should be allocated (*e.g.*, `allocate(pps%varname(beg:end))`) and initialized to NaN (*e.g.*, `pps%varname(beg:end) = nan`) in the order in which the quantity is declared in the derived data type in `clmtype`.

3.2 Decomposition and Clumps

CLM was implemented to provide performance portability across a wide range of current and future computer architectures. The model runs on hardware ranging from laptop computers to the largest commercial supercomputers available, and it offers good performance on both vector and scalar architectures. The model can be run serially or in parallel using distributed or shared memory processors or a combination of distributed and shared memory processors. The model uses MPI (the Message Passing Interface) for distributed memory parallelism and OpenMP for shared memory parallelism. On the Cray X1, the model can also stream across processing units (called Single Streaming Processors or SSPs) within a Multi-Streaming Processor (MSP) based on Cray Streaming Directives (CSDs) included in the code.

When compiled for distributed memory parallelism (with CPP variable `SPMD` defined), each MPI process will create an instance of the data structures shown in Figure 3.2 containing only the subset of data assigned to that process. A cache-friendly blocking structure is superimposed on the data structure hierarchy for improved computational efficiency. This blocking structure implicitly controls the vector length of most computations. Gridcells are grouped into blocks (called “clumps”) of nearly equal computational cost, and these clumps are subsequently assigned to MPI processes.

The computational cost of a gridcell is approximately proportional to the number of PFTs contained within it. However, since computational cost for some PFTs is higher than for others and since similar PFTs tend to cluster geographically, balancing the workload across MPI processes requires a more complex scheme than simply assigning contiguous blocks of gridcells to clumps. To minimize the potential for load imbalance, gridcells are assigned in cyclic (or round robin) fashion to a pre-determined number of clumps. The clumps are then assigned in cyclic fashion to available MPI processes. This scheme has proven to sufficiently distribute gridcells of various costs among MPI processes yielding very good parallel load balancing characteristics for most process counts and surface datasets. The clumped decomposition scheme is implemented in `initDecomp` (Section A.49.1).

Clumps not only define the workload for an MPI process, they also serve to block data for shared memory parallelism when using OpenMP. The number of clumps per MPI process is determined by the parallel configuration of the model at run time in `control_init` (Section A.48.1), but it may be set explicitly by

setting the `clump_pproc` namelist variable to the desired number of clumps per process. When run serially or with MPI-only parallelism, one clump per process is used. When OpenMP is used, the number of clumps per process is set to the maximum number of OpenMP threads available. The maximum number of threads is normally declared by setting the `OMP_NUM_THREADS` environment variable in the jobscript to the desired number. On the Cray X1 when OpenMP is disabled, CSDs are interpreted by the compiler in place of OpenMP directives, and the number of clumps per process is set to four to take maximum advantage of the four SSP processing units on an MSP.

3.3 Filters

In addition to clumps, another set of structures, called “filters,” are used to better support vectorized processing of columns and PFTs. Filters group like columns or PFTs based on their process-specific categorization. Like clumps, filters contain a vector of array indices and are used for indirect addressing into the main data structure hierarchy. Filters are created in `initFilters` (Section A.54.1) for snow, non-snow, lake, non-lake, vegetated, and bare soil columns and PFTs for each clump of gridcells. Most filters are initialized once, but the snow and non-snow filters must be reconstructed as snowfall and melting occur, and the vegetated PFT filter must be reconstructed in DGVM as die-off and establishment occur.

3.4 Loop Composition

In CLM3, the highest level loops in the `driver` (Section A.52) routine run over clumps for each MPI process and provide for OpenMP and Cray Streaming parallelism. Science subroutines, called within these loops, are passed local clump bounds for gridcells, landunits, columns, and PFTs as needed. Relevant filters, in the form of counts and vectors of array indices, are also passed as needed to science subroutines.

Shown below is a portion of a high level loop from the `driver` routine. First, the number of clumps assigned to the process is obtained and stored in `nclumps`. The subsequent loop over all clumps is wrapped with OpenMP and Cray Streaming Directives to support shared memory parallelism. Within the loop, the bounds for gridcells, landunits, columns, and PFTs are obtained for the clump being processed by calling `get_clump_bounds` (Section A.49.10). Then a science subroutine, `Hydrology1` (Section A.21.1), is called and passed the column and PFT bounds as well as the non-lake filters for columns and PFTs for the clump being processed. Additional science subroutines are subsequently called within the same loop. The `driver` routine consists primarily of two such high level loops which call most of the science subroutines used by the model.

```

nclumps = get_proc_clumps()
!$OMP PARALLEL DO PRIVATE (nc,begg,endg,begl,endl,begc,endc,begp,endp)
!CSD$ PARALLEL DO PRIVATE (nc,begg,endg,begl,endl,begc,endc,begp,endp)
  do nc = 1,nclumps
    call get_clump_bounds(nc, begg, endg, begl, endl, begc, endc, begp, endp)
    .
    .
    call Hydrology1(begc, endc, begp, endp, &
      filter(nc)%num_nolakec, filter(nc)%nolakec, &
      filter(nc)%num_nolakep, filter(nc)%nolakep)
    .
    .
  end do
!CSD$ END PARALLEL DO
!$OMP END PARALLEL DO
```

Within science subroutines, vector loops run over grid or subgrid units. Vector loops may run over an entire clump of grid or subgrid units, or they may use filters for indirect addressing of a specific list of subgrid units to process. Other loops, which tend to be very short, run over snow and soil levels within a column or PFT. In most cases, the vector (grid/subgrid) loops are contained within the short (level) loops to exploit vectorization opportunities. When writing code in this manner, it is often necessary to split lengthy loops into multiple loops and use temporary local arrays, called vector temporaries, to pass data from one loop to the next. Since arrays in data structures in CLM are implemented as pointers, compilers usually can not determine if vector dependencies exist. As a result, compiler directives are required in order to obtain loop vectorization.

Shown below is an example of a filter loop within a science subroutine. First, local pointers are created to shorten the notation used in equations. The subsequent loop over all non-lake columns is preceded by Cray X1 and NEC/Earth Simulator compiler directives. The first directive tells the Cray X1 compiler that the loop is concurrent, meaning it may be streamed and vectorized. The second directive tells the NEC/Earth Simulator compiler that no vector dependencies exist in the loop. Within the loop, the column index is obtained from the non-lake column filter vector, the appropriate landunit index is obtained from the column's landunit vector, and the appropriate gridcell index is obtained from the column's gridcell vector. Next, the landunit type and the ground temperature of the column are checked. If the landunit contains water and the ground temperature is above freezing, three variables are initialized to zero. Other computations are usually performed within such loops.

```

! Assign local pointers to derived type
! members (landunit-level)
clandunit => clm3%g%l%c%landunit
itype     => clm3%g%l%itype
! Assign local pointers to derived type
! members (column-level)
cgridcell => clm3%g%l%c%gridcell
t_grnd    => clm3%g%l%c%ces%t_grnd
h2osno    => clm3%g%l%c%cws%h2osno
snowdp    => clm3%g%l%c%cps%snowdp
snowage   => clm3%g%l%c%cps%snowage

!dir$ concurrent
!cdir nodep
do f = 1, num_nolakec
  c = filter_nolakec(f)
  l = clandunit(c)
  g = cgridcell(c)
  .
  .
  .
  if (itype(l) == istwet .and. t_grnd(c) > tfrz) then
    h2osno(c) = 0._r8
    snowdp(c) = 0._r8
    snowage(c) = 0._r8
  end if
  .
  .
  .
end do

```

3.5 Averaging from a Subgrid Level to an Encompassing Subgrid Level

Generic routines have been developed for performing averages from a subgrid level to every subgrid/grid level encompassing it. Shown below is table of routines which may be called to perform this averaging for clumps or filters of subgrid units. Required arguments are the subgrid bounds for both levels in the hierarchy and the two data arrays for both levels in the hierarchy. Additional arguments are required for multi-level variables and for scaling quantities. Individual routines called by the interfaces shown in the table below are contained in module `subgridAveMod` (Section A.86).

Routine Interface	Description
<code>p2c</code>	Average from PFTs to columns
<code>p2l</code>	Average from PFTs to landunits
<code>p2g</code>	Average from PFTs to gridcells
<code>c2l</code>	Average from columns to landunits
<code>c2g</code>	Average from columns to gridcells
<code>l2g</code>	Average from landunits to gridcells

Examples of their use can be found in module `histFileMod` (Section A.56), which calls many routines for updating history fields, and `pft2colMod` (Section A.78), which calls `p2c` to perform averages from PFT-level variables to column-level variables for water state and flux quantities.

Chapter 4

Input and Output Files

4.1 Surface Data Input

A list of high resolution “raw” datasets is provided with the CLM3.0 distribution. These raw datasets are only needed if a surface dataset is to be created at runtime (see *CLM3.0 User’s Guide*). The following is a list of these datasets:

- *mksrf_pft.nc*: raw vegetation type dataset
- *mksrf_soitex.10level.nc*: raw soil texture dataset
- *mksrf_soicol.clm2.nc*: raw soil color dataset
- *mksrf_lanwat.nc*: raw inland water dataset
- *mksrf_urban.nc*: raw urban dataset
- *mksrf_glacier.nc*: raw glacier dataset
- *mksrf_lai.nc*: raw leaf and stem area index, canopy top and bottom height dataset

The user has the option of replacing any of these raw datasets, as long as the structure of the netCDF files is preserved. A series of Fortran routines to convert ASCII to raw netCDF data is provided in the directory, *tools/convert_ascii/*. Alternatively, the user may wish to modify the model surface data directly as long as the original netCDF file structure is preserved.

4.2 History Dataset Fields

Traditionally, CLM has only output history fields on the model lat-lon grid. CLM3.0, however, has the new capability to output RTM river routing related fields directly on the RTM grid, rather than interpolating them back to the model grid as was the case in CLM2.1 and CLM2.0.

All aspects of history file output are controlled by the files *histFldsMod.F90* (Section A.57) and *histFileMod.F90* (Section A.56). The *histFileMod.F90* contains routines for creating, updating and outputting the history buffer during the course of a model simulation whereas *histFldsMod.F90* contains the user interface for defining all the output history fields (on both the model grid and the RTM grid).

In what follows, we provide a discussion of the CLM3.0 history field interface as well as a summary of the user modifications needed to add new history fields to the model output. We discuss the history field interface for both the model grid as well as RTM grid history output. By default, model grid output is

always assumed in the history field interface. It is assumed that the user is completely familiar with Fortran 90 pointer concepts and syntax.

4.2.1 History Output on the Model Grid

History output on the model grid may appear in either xy grid form or native subgrid form (*i.e.*, as one-dimensional gridcell, landunit, column or PFT output). The output format is controlled by the value of the namelist variable HIST_DOV2XY.

The interface to add history output fields in file *histFldsMod.F90* assumes that all arguments occur as keyword-value pairs. The interface also assumes that each field to be added is associated with either a gridcell, landunit, column or PFT level data structure component that has already been defined in `clmtype` (Section A.46). History fields are either single-level or multi-level fields. A single-level field will have either the total number of model PFTs, columns, landunits or gridcells as its only non-time dimension. A multi-level field will have the number of model PFTs, columns, landunits or gridcells as one non-time dimension, along with a level field (*e.g.*, number of soil levels) as a second non-time dimension.

To add a single-level history output field, a call to `add_fld1d` (Section A.56.22) needs to be added to routine `initHistFlds` (Section A.57.1) in module `histFldsMod` (Section A.57). Similarly, to add a multi-level history output field, a call to subroutine `add_fld2d` (Section A.56.23) needs to be added to `initHistFlds`. We describe the formats of calls to `add_fld1d` and `add_fld2d` via concrete examples in routine `inithistFlds`.

A single-level history field, FCEV, is added to the history output via the following call:

```
call add_fld1d (fname='FCEV', units='watt/m^2', &
               avgflag='A', long_name='canopy evaporation', &
               ptr_pft=clm3%g%l%c%p%pef%eflx_lh_vege, set_lake=0._r8)
```

`fname`
history file field name

`long_name`
descriptive name of field

`units`
units of field

`avg flag`
time averaging flag
valid values are "X" (maximum over time), "A" (time average),
"M" (minimum over time) or "I" (instantaneous)

`ptr_pft` or `ptr_col` or `ptr_lunit` or `ptr_gridcell`
pointer to `clmtype` PFT, column, landunit or gridcell
arrays

`set_lake`
value to set lake points to (optional)
if this value is not set, it is assumed that the model will
provide valid values for the field over lake points

Note that `ptr_pft` corresponds to the subgrid level at which the canopy evaporation, `eflx_lh_vege`, is defined in `clmtype` (Section A.46). Since the canopy evaporation is defined at the PFT level in `clmtype`,

a PFT level pointer, `ptr_pft`, needs to be declared in the call to `add_fld1d`. Alternatively, since the snow height, `SNOWDP`, is defined at the column level in `clmtype`, the following interface call is used to add this single-level field to the history output:

```
call add_fld1d (fname='SNOWDP', units='m', &
               avgflag='A', long_name='snow height', &
               ptr_col=clm3%g%l%c%cps%snowdp)
```

If a multi-level field is to be added to the history output, the following type of call needs to be added to `add_fld2d`:

```
call add_fld2d (fname='H2OSOI', units='mm3/mm3', type2d='levsoi', &
               avgflag='A', long_name='volumetric soil water', &
               ptr_col=clm3%g%l%c%cws%h2osoi_vol)
```

`fname`
history file field name

`long_name`
descriptive name of field

`units`
units of field

`type2d`
second-dimension output type - valid values are
'levsoi', 'levlak', or 'numrad'

`avg flag`
time averaging flag
valid values are "X" (maximum over time), "A" (time average),
"M" (minimum over time) or "I" (instantaneous)

`ptr_pft` or `ptr_col` or `ptr_lunit` or `ptr_gridcell`
pointer to `clmtype` PFT, column, landunit or gridcell
arrays

`set_lake`
value to set lake points to (optional)
if this value is not set, it is assumed that the model will
provide valid values for the field over lake points
here it is set to zero, but it could also be set to a
missing value (1.e36)

The call to `add_fld1d` or `add_fld2d` must include either `ptr_pft`, `ptr_col`, `ptr_lunit`, or `ptr_gcell` as an argument. The output name of the field (`fname`), a descriptive name (`long_name`), the field units (`units`), the output type of time averaging that will be used for the field must also be provided as arguments. Finally, an optional argument, `set_lake`, is provided to specify masking values over lake.

4.2.2 History Output on the RTM Grid

As mentioned above, CLM3.0 also has the new capability to output RTM-related fields directly on the RTM grid (rather than interpolating them back to the model grid). We only briefly discuss this new aspect

of the model. For RTM-grid related fields, the history interface is slightly different than for model grid related fields:

```
call add_fldid (fname='QCHANR', units='m3/s', typexy='rof', &
               avgflag='A', long_name='RTM river flow', &
               ptr_roflnd=runoff%lnd)

call add_fldid (fname='QCHOCNR', units='m3/s', typexy='rof', &
               avgflag='A', long_name='RTM river discharge into ocean', &
               ptr_rofocn=runoff%ocn)

typexy
denotes that the output xy grid is the RTM runoff grid
currently, the only valid values for typexy are 'clm' and 'rof'

ptr_roflnd or ptr_rofocn
pointer to one-dimensional land or ocn runoff vectors
```

We note that `typexy` is an optional argument whose valid values are `'clm'` or `'rof'`. The default value for `typexy` is `'clm'`, which is why it does not appear in the calls to the normal model output fields. However, if `typexy` is set to `'rof'` then either `ptr_roflnd` or `ptr_rofocn` must be supplied as arguments to `add_fldid`. Note that `runoff%ocn` and `runoff%lnd` are not defined in `clmtype` (Section A.46) (as is the case for `ptr_pft`, etc.) but rather are defined in the `RunoffMod` (Section A.26).

4.3 Restart Datasets

CLM3.0 produces “restart” files containing data necessary to describe the exact state of the model run when it was halted. Restart files allow CLM3.0 to be continued or branched to produce exactly the same bit-for-bit answers as if it had never stopped. Restart files contain instantaneous binary data for a set of time-dependent variables. The data is stored in binary format to ensure bit-for-bit agreement with a terminated run which was subsequently restarted. Restart files should be used to extend previously-started simulations or to branch from one simulation to another in which history file namelist variables have been modified. For a branch run, the user must change the simulation’s case name unless the namelist variable `BRNCH_RETAIN_CASENAME` is set to `.true.`. A restart file also contains a version number to ensure that the code being used is compatible with the restart file being read. If the user modifies the contents of the restart file, the version number should be incremented.

The criterion for a “successful restart” is that the model has available exactly the same information upon restart that it would have had if it had not stopped. Restart files may need to be modified during model development to include new time-dependent variables. A simple rule is that if a run produces even slightly different answers when restarted compared to when left uninterrupted, then certain variables are missing from the restart file. Similarly, a restart should match a non-restarted run even if the number of MPI tasks and/or the number of OpenMP threads is changed upon restarting the run (as long as both runs are run on the same machine).

The module `restFileMod` (Section A.82) controls restart file logic. Restart files are written and read in routine `restart` (Section A.82.1) (in module `restFileMod`). When routine `restart` is called with the argument `flag` set to `'read'` the model reads the restart dataset, whereas if the argument `flag` set to `'write'`, restart datasets are created. The actual reading and writing of restart files are carried out in the following routines:

- `restart_setup` (in `restFileMod` (Section A.82)) sets up restart files read/write and perform necessary consistency checks

- `restart_time` (in `restFileMod` (Section A.82)) reads/write time manager information to/from restart file
- `restart_biogeophysics` (in `restFileMod` (Section A.82)): reads/writes biogeophysics related restart data
- `restart_coupler` (in `clm_csmMod` (Section A.41)) reads/writes CLM restart data needed in csm mode (if CPP variable `COUP_CSM` is defined)
- `restart_rtm` (in `RtmMod` (Section A.25)) reads/writes RTM related restart data if CPP variable `RTM` is defined
- `restart_dgvm` (in `DGVMRestMod`) reads/writes DGVM related restart data if CPP variable `DGVM` is defined
- `restart_accum` (in `accumulMod` (Section A.37)) reads/write accumulation field restart data

If a user modifies the code such that it becomes necessary to add a new variable to the restart file, the user must first determine which of the above routines restart information must be added to. The user should then consult that routine for explicit examples of how to add restart data.

We briefly summarize key points of writing and reading restart data using biogeophysics related restart variables as examples. The following example, taken out of `restart.biogeophys` illustrates how a single-level biogeophysics related variable defined at the PFT-level in `clmtype` (Section A.46) is written to and read from the restart file:

```
! PFT type physical state variable - fwet

      if (flag == 'read') call readin (nio, rbuf1dp, clmlevel=namep)
!dir$ concurrent
!cdir nodep
      do p = begp, endp
        if (flag == 'read' ) clm3%g%1%c%p%pps%fwet(p) = rbuf1dp(p)
        if (flag == 'write') rbuf1dp(p) = clm3%g%1%c%p%pps%fwet(p)
      end do
      if (flag == 'write') call wrtout (nio, rbuf1dp, clmlevel=namep)
```

Subroutines `readin` and `wrtout` reside in module `iobinary` (Section A.63) and correspond to overloaded module interface routines which read in and write out binary data. The array, `rbuf1dp`, is a temporary array allocated in `restart.biogeophys` (Section A.82.4) and is dimensioned as the total number of model PFTs. Analogous temporary arrays, `rbuf1dg`, `rbuf1dl` and `rbuf1dc` are also defined and dimensioned as the total number of model gridcells, landunits and columns, respectively. Finally, the argument corresponding to `clmlevel` can take on the values `namep`, `namec`, `namel` and `nameg`, (see module `clmtype`). The following provides a summary of calls to binary writes for PFT-level, column-level, landunit-level, and gridcell-level arrays:

```
call wrtout (nio, rbuf1dp, clmlevel=namep)
call wrtout (nio, rbuf1dc, clmlevel=namec)
call wrtout (nio, rbuf1dl, clmlevel=namel)
call wrtout (nio, rbuf1dg, clmlevel=nameg)
```

Similar calls also pertain to binary read calls.

Multi-level fields can also be written to and read from the restart files as illustrated below:

```
! Column physical state - z (snow)

allocate (rbuf2dc(-nlevsno+1:0,numc))
if (flag == 'read') call readin (nio, rbuf2dc, clmlevel=namec)
do j = -nlevsno+1,0
!dir$ concurrent
!cdir nodep
    do c = begc, endc
        if (flag == 'read' ) clm3%g%1%c%cps%z(c,j) = rbuf2dc(j,c)
        if (flag == 'write') rbuf2dc(j,c) = clm3%g%1%c%cps%z(c,j)
    end do
end do
if (flag == 'write') call wrtout (nio, rbuf2dc, clmlevel=namec)
deallocate(rbuf2dc)
```

It is important to note that when two-dimensional arrays are passed to routines `readin` or `wrtout` in `iobinary` (Section A.63), arrays must be dimensioned as `array(level, subgrid_type)`.

4.4 Initial Datasets

In addition to the periodic generation of restart files, and depending on the setting of the namelist variable `HIST.CRTINIC`, CLM3.0 also periodically produces netCDF initial files containing instantaneous variable data in the native subgrid form. The output dimension of each dataset variable corresponds to its dimensionality when it is defined in `clmtype` (Section A.46).

If, at startup, a new CLM case uses the initial dataset produced by a reference case (via the setting of the namelist variable `FINIDAT`), the model will not continue in a bit-for-bit fashion with respect to the reference case. The resulting climate, however, should be continuous as long as the namelist or source code is not changed when compared to the reference case. Initial dataset files are smaller and more flexible than binary restart files and are used in cases where it is not crucial for the new run to be bit-for-bit the same as the run which produced the initial files. Since certain model variables take relatively long to spin up (*e.g.*, soil and snow related variables), the use of an initial dataset ensures faster spin up of the model than using arbitrary initialization. Furthermore, since initial files have a self-describing format, multiple versions of the model code can use a given initial dataset. Restart files, on the other hand, may change frequently during model development and are often difficult to use with different versions of the code.

All initial dataset logic is contained in the file `inicFileMod.F90`. Routine `inicfile` (Section A.60.1) controls the top level logic to create or read from an initial dataset depending on the value of the input argument `flag`. Routine `inicfields` (Section A.60.2) actually performs the write to or read from the initial dataset.

If new variables are to be added to the initial dataset, the user must modify routine `inicfields` (Section A.60.2). We provide several illustrative examples currently in `inicfields` (Section A.60.2) of the interface usage needed to add both single-level and multi-level variables to the initial dataset.

To add a new field to the initial dataset, a call must be made to `ncd_defvar` (Section A.77) to define the field followed by a call to `ncd_iolocal` (Section A.77) to read or write the field. The total set of initial dataset fields are defined, written out or read in via calls to `inicfields` (Section A.60.2) with `flag` set to `'define'`, `'write'` or `'read'`, respectively.

The following call to `ncd_defvar` is made to define the single-level initial dataset field, `T_GRND`:

```
call ncd_defvar(ncid=ncid, varname='T_GRND', xtype=nf_double, &
               dim1name='column', long_name='ground temperature', units='K')

ncid
    netcdf file id

varname
    variable name

xtype
    output netCDF type (nf_int or nf_double)

dim1name
    subgrid dimension: 'pft', 'column', 'landunit', or 'gridcell'

long_name
    variable descriptive name (optional)

units
    variable units (optional)
```

Subsequently, `T_GRND` is written to or read from the initial dataset field via the following call to `ncd_iolocal`:

```
call ncd_iolocal(varname='T_GRND', data=cptr%ces%t_grnd,
                 datadim1='column', &
                 ncid=ncid, flag=flag, readvar=readvar)
if (flag=='read' .and. .not. readvar) call endrun()

ncid
    netcdf file id

flag
    'read' or 'write'

varname
    variable name

data
    pointer to clmtype PFT, column, landunit or gridcell array

datadim1
    subgrid dimension: 'pft', 'column', 'landunit', or 'gridcell'

readvar
    only relevant when flag is set to 'read'
    used to determine action if variable is not on the initial dataset
```

Note that if `flag` is set to `'read'`, a check is made after the call to `ncd_iolocal` to determine if the variable is on the initial dataset. For most variables, the model will stop if the variable is not present. The user can

change this logic, however, to provide default values for user added variables if they are not present on the initial dataset.

Similarly, the following call to `ncd_defvar` is made to define the multi-level initial dataset field, `H2OSOI_LIQ`:

```
call ncd_defvar(ncid=ncid, varname='H2OSOI_LIQ', xtype=nf_double, &
               dim1name='column', dim2name='levtot', long_name='liquid water', units='kg/m2')

ncid
  netcdf file id

varname
  variable name

xtype
  output netCDF type (nf_int or nf_double)

dim1name
  subgrid dimension: 'pft', 'column', 'landunit', or 'gridcell'

dim2name
  level dimension: 'levtot', 'levlak' or 'levsno'

long_name
  variable descriptive name (optional)

units
  variable units (optional)
```

with the accompanying call to `ncd_iolocal` to read/write the field:

```
call ncd_iolocal(varname='H2OSOI_LIQ', data=cptr%cws%h2osoi_liq,
                datadim1='column', datadim2='levtot', &
                ncid=ncid, flag=flag, readvar=readvar)
if (flag=='read' .and. .not. readvar) call endrun()

ncid
  netcdf file id

flag
  'read' or 'write'

varname
  variable name

data
  pointer to clmtype PFT, column, landunit or gridcell array

datadim1
  subgrid dimension: 'pft', 'column', 'landunit', or 'gridcell'

datadim1
```

level dimension for variable: 'levtot', 'levlak' or 'levsno'

readvar

only relevant when flag is set to 'read'

used to determine action if variable is not on the initial dataset

Chapter 5

Error Conditions

Error codes exist in the model to warn the user of inconsistencies and errors. The model is released to the community as a package which should not trigger such inconsistencies or errors. However, the user may change the code or modify the input files in a way which triggers such a message. The user may even simply operate the model on a platform which brings up an error message. In many cases these errors will cause the model to terminate in order to alert the user of a problem which will render the simulation useless or will cause it to fail later.

In general, when an error message appears, the user must switch gears from production to debugging mode. The error message may make the corrective measures obvious. If not, the user will need to find the line in the code which wrote the error message and gradually back track until the cause for the error is found. The old fashioned placement of write statements in the code or, if available, debugging software may help understand the problem. When using debugging software, it is useful to change `DEBUG` to `TRUE` in the `jobscript`.

5.1 Energy and water balance errors

To make sure that the model conserves energy and mass, energy and water imbalances appearing in a simulation will trigger errors and halt the simulation,

To accomplish energy conservation, all energy inputs at the land surface are reflected or absorbed. Absorbed energy may return to the atmosphere as sensible heat or may be emitted as terrestrial radiation, stored as soil heat, used to evaporate or transpire water, or used to melt snow and ice. Similarly, water incident on the land will evaporate, transpire through leaf stomata, run off and drain from the soil, or be stored in the soil and snow pack. The error checks for energy and water balance are contained in module `BalanceCheckMod` (Section [A.1](#)).

Water which runs off or drains from the soil fills river channels and flows downstream. At every model time step, the global sum of water which runs off and drains from the soil into the rivers must equal the global sum of change in river volume. This is because the global sum of water flowing out of grid cells must cancel the global sum of water flowing into grid cells from upstream. This error check is contained in module `RtmMod` (Section [A.25](#)).

Bibliography

- Branstetter, M. (2001). *Development of a Parallel River Transport Algorithm and Applications to Climate Studies*. PhD thesis, University of Texas at Austin. J. Famiglietti, supervisor.
- Branstetter, M. L. and Famiglietti, J. S. (1999). Testing the sensitivity of GCM-simulated runoff to climate model resolution using a parallel river transport algorithm. In *Proceedings of 14th Conference on Hydrology, American Meteorological Society*, Dallas, Texas.
- Hoffman, F. M., Vertenstein, M., Kitabata, H., White, J. B., Worley, P., Drake, J., and Cordery, M. (2004). [Adventures in Vectorizing the Community Land Model](#). In *Proceedings of the 2004 Cray Users Group Meeting*, Knoxville, Tennessee.
- Levis, S., Bonan, G. B., Vertenstein, M., and Oleson, K. W. (2004). The Community Land Model's Dynamic Global Vegetation Model (CLM-DGVM): Technical description and user's guide. Technical Note NCAR/TN-459+IA, National Center for Atmospheric Research.
- Oleson, K., Dai, Y., Bonan, G., Bosilovich, M., Dickinson, R., Dirmeyer, P., Hoffman, F., Houser, P., Levis, S., Niu, G.-Y., Thornton, P., Vertenstein, M., Yang, Z.-L., and Zeng, X. (2004). Technical Description of the Community Land Model (CLM). Technical Note NCAR/TN-461+STR, National Center for Atmospheric Research.
- Vertenstein, M., Hoffman, F., Oleson, K., and Levis, S. (2004). Community Land Model version 3.0 (CLM3.0) user's guide. Technical report, National Center for Atmospheric Research.

Appendix A

ProT_EX Source Code Documentation

A.1 Module BalanceCheckMod

(Source File: BalanceCheckMod.F90)

Water and energy balance check.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils,   only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: BalanceCheck ! Water and energy balance check
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```

A.1.1 BalanceCheck

INTERFACE:

```
subroutine BalanceCheck(lbp, ubp, lbc, ubc)
```

DESCRIPTION:

This subroutine accumulates the numerical truncation errors of the water and energy balance calculation. It is helpful to see the performance of the process of integration.

The error for energy balance:

error = abs(Net radiation - change of internal energy - Sensible heat - Latent heat)

The error should be less than 0.02 W/m² in each time integration interval.

The error for water balance:

error = abs(precipitation - change of water storage - evaporation - runoff)

The error should be less than 0.001 mm in each time integration interval.

USES:

```
use clmtype
use subgridAveMod
use time_manager , only: get_step_size, get_nstep
```

ARGUMENTS:

```
implicit none
integer :: lbp, ubp ! pft-index bounds
integer :: lbc, ubc ! column-index bounds
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 10 November 2000: Mariana Vertenstein
 Migrated to new data structures by Mariana Vertenstein and
 Peter Thornton

LOCAL VARIABLES:

```

local pointers to original implicit in arrays
  integer , pointer :: pgridcell(:)      ! pft's gridcell index
  integer , pointer :: cgridcell(:)      ! column's gridcell index
  real(r8), pointer :: forc_rain(:)      ! rain rate [mm/s]
  real(r8), pointer :: forc_snow(:)      ! snow rate [mm/s]
  real(r8), pointer :: forc_lwrad(:)     ! downward infrared (longwave) ra
  real(r8), pointer :: endwb(:)         ! water mass end of the time step
  real(r8), pointer :: begwb(:)         ! water mass beginning of the time
  real(r8), pointer :: fsa(:)           ! solar radiation absorbed (total
  real(r8), pointer :: fsr(:)           ! solar radiation reflected (W/m*
  real(r8), pointer :: eflx_lwrad_out(:) ! emitted infrared (longwave) rad
  real(r8), pointer :: eflx_lwrad_net(:) ! net infrared (longwave) rad (W/
  real(r8), pointer :: sabv(:)          ! solar radiation absorbed by veg
  real(r8), pointer :: sabg(:)          ! solar radiation absorbed by gro
  real(r8), pointer :: eflx_sh_tot(:)    ! total sensible heat flux (W/m**
  real(r8), pointer :: eflx_lh_tot(:)    ! total latent heat flux (W/m8*2)
  real(r8), pointer :: eflx_soil_grnd(:) ! soil heat flux (W/m**2) [+ = in
  real(r8), pointer :: qflx_evap_tot(:)  ! qflx_evap_soi + qflx_evap_veg +
  real(r8), pointer :: qflx_surf(:)      ! surface runoff (mm H2O /s)
  real(r8), pointer :: qflx_qrgwl(:)     ! qflx_surf at glaciers, wetlands
  real(r8), pointer :: qflx_drain(:)     ! sub-surface runoff (mm H2O /s)

local pointers to original implicit out arrays
  real(r8), pointer :: errh2o(:)         ! water conservation error (mm H2
  real(r8), pointer :: errsol(:)         ! solar radiation conservation er
  real(r8), pointer :: errlon(:)         ! longwave radiation conservation
  real(r8), pointer :: errseb(:)         ! surface energy conservation err

local pointers to original implicit in multi-level arrays
  real(r8), pointer :: forc_solad(:, :)  ! direct beam radiation (vis=forc
  real(r8), pointer :: forc_solai(:, :)  ! diffuse radiation      (vis=forc

```


A.2 Module BareGroundFluxesMod (Source File: BareGroundFluxesMod.F90)

Compute sensible and latent fluxes and their derivatives with respect to ground temperature using ground temperatures from previous time step.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: BareGroundFluxes    ! Calculate sensible and latent heat fluxes
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```

A.2.1 BareGroundFluxes

INTERFACE:

```
subroutine BareGroundFluxes(lbp, ubp, num_nolakep, filter_nolakep)
```

DESCRIPTION:

Compute sensible and latent fluxes and their derivatives with respect to ground temperature using ground temperatures from previous time step.

USES:

```
use clmtype
use clm_varpar      , only : nlevsoi
use clm_varcon      , only : cpair, vkc, grav
use shr_const_mod   , only : SHR_CONST_RGAS
use FrictionVelocityMod, only : FrictionVelocity, MoninObukIni
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp                ! pft bounds
integer, intent(in) :: num_nolakep             ! number of pft non-1
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-
```

CALLED FROM:

```
subroutine Biogeophysics1 in module Biogeophysics1Mod
```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 12/19/01, Peter Thornton

This routine originally had a long list of parameters, and also a reference the entire clm derived type. For consistency, only the derived type refere is passed (now pointing to the current column and pft), and the other origi parameters are initialized locally. Using t_grnd instead of tg (tg eliminat as redundant).

1/23/02, PET: Added pft reference as parameter. All outputs will be written to the pft data structures, and averaged to the column level outside of this routine.

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: pcolumn(:)      ! pft's column index
  integer , pointer :: pgridcell(:)    ! pft's gridcell index
  integer , pointer :: frac_veg_nosno(:) ! fraction of vegetation not cover
  real(r8), pointer :: t_grnd(:)       ! ground surface temperature [K]
  real(r8), pointer :: thm(:)          ! intermediate variable (forc_t+0.
  real(r8), pointer :: qg(:)           ! specific humidity at ground surf
  real(r8), pointer :: thv(:)          ! virtual potential temperature (k
  real(r8), pointer :: dqgdT(:)        ! temperature derivative of "qg"
  real(r8), pointer :: htvp(:)         ! latent heat of evaporation (/sub
  real(r8), pointer :: beta(:)         ! coefficient of conective velocit
  real(r8), pointer :: zii(:)          ! convective boundary height [m]
  real(r8), pointer :: forc_u(:)        ! atmospheric wind speed in east d
  real(r8), pointer :: forc_v(:)        ! atmospheric wind speed in north
  real(r8), pointer :: forc_t(:)        ! atmospheric temperature (Kelvin)
  real(r8), pointer :: forc_th(:)       ! atmospheric potential temperatur
  real(r8), pointer :: forc_q(:)       ! atmospheric specific humidity (k
  real(r8), pointer :: forc_rho(:)      ! density (kg/m**3)
  real(r8), pointer :: forc_pbot(:)     ! atmospheric pressure (Pa)
  real(r8), pointer :: forc_hgt_u(:)    ! observational height of wind [m]
  real(r8), pointer :: psnsun(:)        ! sunlit leaf photosynthesis (umol
  real(r8), pointer :: psnsha(:)        ! shaded leaf photosynthesis (umol
  real(r8), pointer :: z0mg_col(:)      ! roughness length, momentum [m]

local pointers to implicit inout arguments
  real(r8), pointer :: z0hg_col(:)      ! roughness length, sensible heat
  real(r8), pointer :: z0qg_col(:)     ! roughness length, latent heat [m]

local pointers to implicit out arguments
  real(r8), pointer :: dlrاد(:)         ! downward longwave radiation below
  real(r8), pointer :: ulrad(:)         ! upward longwave radiation above t
  real(r8), pointer :: cgrnds(:)        ! deriv, of soil sensible heat flux
  real(r8), pointer :: cgrndl(:)        ! deriv of soil latent heat flux wr
  real(r8), pointer :: cgrnd(:)         ! deriv. of soil energy flux wrt to
  real(r8), pointer :: taux(:)          ! wind (shear) stress: e-w (kg/m/s*
  real(r8), pointer :: tauy(:)          ! wind (shear) stress: n-s (kg/m/s*
  real(r8), pointer :: eflx_sh_grnd(:)  ! sensible heat flux from ground (W
  real(r8), pointer :: eflx_sh_tot(:)   ! total sensible heat flux (W/m**2)
  real(r8), pointer :: qflx_evap_soi(:) ! soil evaporation (mm H2O/s) (+ =
  real(r8), pointer :: qflx_evap_tot(:) ! qflx_evap_soi + qflx_evap_veg + q
  real(r8), pointer :: t_ref2m(:)       ! 2 m height surface air temperatur
  real(r8), pointer :: q_ref2m(:)       ! 2 m height surface specific humid
  real(r8), pointer :: t_veg(:)         ! vegetation temperature (Kelvin)

```

```
real(r8), pointer :: btran(:)      ! transpiration wetness factor (0 t
real(r8), pointer :: rssun(:)      ! sunlit stomatal resistance (s/m)
real(r8), pointer :: rssha(:)      ! shaded stomatal resistance (s/m)
real(r8), pointer :: ram1(:)        ! aerodynamical resistance (s/m)
real(r8), pointer :: fpsn(:)        ! photosynthesis (umol CO2 /m**2 /s
real(r8), pointer :: rootr(:, :)    ! effective fraction of roots in ea
```


A.3 Module Biogeophysics1Mod

(Source File: Biogeophysics1Mod.F90)

Performs calculation of leaf temperature and surface fluxes. Biogeophysics2.F90 then determines soil/snow and ground temperatures and updates the surface fluxes for the new ground temperature.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Biogeophysics1    ! Calculate leaf temperature and surface fluxes
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.3.1 Biogeophysics1

INTERFACE:

```
subroutine Biogeophysics1(lbg, ubg, lbc, ubc, lbp, ubp, &
    num_nolakec, filter_nolakec, num_nolakep, filter_nolakep)
```

DESCRIPTION:

This is the main subroutine to execute the calculation of leaf temperature and surface fluxes. Biogeophysics2.F90 then determines soil/snow and ground temperatures and updates the surface fluxes for the new ground temperature.

Calling sequence is:

Biogeophysics1:	surface biogeophysics driver
-> QSat:	saturated vapor pressure, specific humidity, and derivatives at ground surface and derivatives at leaf surface using updated leaf temperature

Leaf temperature

Foliage energy conservation is given by the foliage energy budget equation:

$$R_{net} - H_f - LE_f = 0$$

The equation is solved by Newton-Raphson iteration, in which this iteration includes the calculation of the photosynthesis and stomatal resistance, and the integration of turbulent flux profiles. The sensible and latent heat transfer between foliage and atmosphere and ground is linked by the equations:

$$H_a = H_f + H_g \text{ and } E_a = E_f + E_g$$

USES:

```

use clmtype
use clm_varcon      , only : denh2o, denice, roverg, hvap, hsub, &
                      istic, istwet, zlnd, zsno
use clm_varpar      , only : nlevsoi, nlevsno
use QSatMod         , only : QSat

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbg, ubg      ! gridcell-index bound
integer, intent(in) :: lbc, ubc      ! column-index bounds
integer, intent(in) :: lbp, ubp      ! pft-index bounds
integer, intent(in) :: num_nolakec    ! number of column non
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for no
integer, intent(in) :: num_nolakep    ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-l

```

CALLED FROM:

```

subroutine driver

```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
Migrated to clm2.0 by Keith Oleson and Mariana Vertenstein
Migrated to clm2.1 new data structures by Peter Thornton and M. Vertenstein

```

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: ivt(:)          !pft vegetation type
integer , pointer :: ityplun(:)      !landunit type
integer , pointer :: clandunit(:)    !column's landunit index
integer , pointer :: cgridcell(:)    !column's gridcell index
real(r8), pointer :: forc_pbot(:)    !atmospheric pressure (Pa)
real(r8), pointer :: forc_q(:)       !atmospheric specific humidity (kg/
real(r8), pointer :: forc_t(:)       !atmospheric temperature (Kelvin)
real(r8), pointer :: forc_hgt_t(:)   !observational height of temperatur
real(r8), pointer :: forc_th(:)      !atmospheric potential temperature
real(r8), pointer :: forc_u(:)       !atmospheric wind speed in east dir
real(r8), pointer :: forc_v(:)       !atmospheric wind speed in north di
real(r8), pointer :: smpmin(:)       !restriction for min of soil potent
integer , pointer :: snl(:)          !number of snow layers
real(r8), pointer :: frac_sno(:)     !fraction of ground covered by snow
real(r8), pointer :: h2osno(:)       !snow water (mm H2O)
real(r8), pointer :: elai(:)         !one-sided leaf area index with bur
real(r8), pointer :: esai(:)         !one-sided stem area index with bur
real(r8), pointer :: z0mr(:)         !ratio of momentum roughness length
real(r8), pointer :: displar(:)      !ratio of displacement height to ca
real(r8), pointer :: htop(:)         !canopy top (m)
real(r8), pointer :: dz(:, :)        !layer depth (m)
real(r8), pointer :: t_soisno(:, :)  !soil temperature (Kelvin)
real(r8), pointer :: h2osoi_liq(:, :) !liquid water (kg/m2)
real(r8), pointer :: h2osoi_ice(:, :) !ice lens (kg/m2)

```

```

    real(r8), pointer :: watsat(:, :)      !volumetric soil water at saturati
    real(r8), pointer :: sucsat(:, :)      !minimum soil suction (mm)
    real(r8), pointer :: bsw(:, :)         !Clapp and Hornberger "b"
local pointers to implicit out arguments
    real(r8), pointer :: t_grnd(:)         !ground temperature (Kelvin)
    real(r8), pointer :: qg(:)             !ground specific humidity [kg/kg]
    real(r8), pointer :: dqgdT(:)          !d(qg)/dT
    real(r8), pointer :: emg(:)            !ground emissivity
    real(r8), pointer :: htvp(:)           !latent heat of vapor of water (or
    real(r8), pointer :: beta(:)           !coefficient of convective velocity
    real(r8), pointer :: zii(:)            !convective boundary height [m]
    real(r8), pointer :: thm(:)            !intermediate variable (forc_t+0.00
    real(r8), pointer :: thv(:)            !virtual potential temperature (kel
    real(r8), pointer :: z0mg(:)           !roughness length over ground, mome
    real(r8), pointer :: z0hg(:)           !roughness length over ground, sens
    real(r8), pointer :: z0qg(:)           !roughness length over ground, late
    real(r8), pointer :: emv(:)            !vegetation emissivity
    real(r8), pointer :: z0m(:)            !momentum roughness length (m)
    real(r8), pointer :: displa(:)         !displacement height (m)
    real(r8), pointer :: z0mv(:)           !roughness length over vegetation,
    real(r8), pointer :: z0hv(:)           !roughness length over vegetation,
    real(r8), pointer :: z0qv(:)           !roughness length over vegetation,
    real(r8), pointer :: eflx_sh_tot(:)    !total sensible heat flux (W/m**2)
    real(r8), pointer :: eflx_lh_tot(:)    !total latent heat flux (W/m8*2) [
    real(r8), pointer :: eflx_sh_veg(:)    !sensible heat flux from leaves (W/
    real(r8), pointer :: qflx_evap_tot(:) !qflx_evap_soi + qflx_evap_veg + qf
    real(r8), pointer :: qflx_evap_veg(:) !vegetation evaporation (mm H2O/s)
    real(r8), pointer :: qflx_tran_veg(:) !vegetation transpiration (mm H2O/s
    real(r8), pointer :: cgrnd(:)          !deriv. of soil energy flux wrt to
    real(r8), pointer :: cgrnds(:)         !deriv. of soil sensible heat flux
    real(r8), pointer :: cgrndl(:)         !deriv. of soil latent heat flux wr
    real(r8), pointer :: tssbef(:, :)      !soil/snow temperature before updat

```


A.4 Module Biogeophysics2Mod

(Source File: Biogeophysics2Mod.F90)

Performs the calculation of soil/snow and ground temperatures and updates surface fluxes based on the new ground temperature.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Biogeophysics2    ! Calculate soil/snow and ground temperatures
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.4.1 Biogeophysics2

INTERFACE:

```
subroutine Biogeophysics2 (lbc, ubc, lbp, ubp, num_nolakec, &
                           filter_nolakec, num_nolakep, filter_nolakep)
```

DESCRIPTION:

This is the main subroutine to execute the calculation of soil/snow and ground temperatures and update surface fluxes based on the new ground temperature

Calling sequence is:

```
Biogeophysics2:      surface biogeophysics driver
-> SoilTemperature:  soil/snow and ground temperatures
  -> SoilTermProp    thermal conductivities and heat capacities
  -> Tridiagonal     tridiagonal matrix solution
  -> PhaseChange     phase change of liquid/ice contents
```

(1) Snow and soil temperatures

- o The volumetric heat capacity is calculated as a linear combination in terms of the volumetric fraction of the constituent phases.
- o The thermal conductivity of soil is computed from the algorithm of Johansen (as reported by Farouki 1981), and the conductivity of snow is from the formulation used in SNTHERM (Jordan 1991).
- o Boundary conditions:
F = Rnet - Hg - LEg (top), F= 0 (base of the soil column).
- o Soil / snow temperature is predicted from heat conduction in 10 soil layers and up to 5 snow layers.
The thermal conductivities at the interfaces between two

neighboring layers (j, j+1) are derived from an assumption that the flux across the interface is equal to that from the node j to the interface and the flux from the interface to the node j+1. The equation is solved using the Crank-Nicholson method and results in a tridiagonal system equation.

(2) Phase change (see PhaseChange.F90)

USES:

```
use clmtype
use time_manager      , only : get_step_size
use clm_varcon        , only : hvap, cpair, grav, vkc, tfrz, sb
use clm_varpar        , only : nlevsno, nlevsoi, maxpatch_pft
use SoilTemperatureMod, only : SoilTemperature
use subgridAveMod     , only : p2c
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp          ! pft bounds
integer, intent(in) :: lbc, ubc          ! column bounds
integer, intent(in) :: num_nolakec       ! number of column non
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for no
integer, intent(in) :: num_nolakep       ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-l
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 Migrated to clm2.0 by Keith Oleson and Mariana Vertenstein
 Migrated to clm2.1 new data structures by Peter Thornton and M. Vertenstein

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: pcolumn(:)          !pft's column index
integer , pointer :: pgridcell(:)        !pft's gridcell index
integer , pointer :: npfts(:)             !column's number of pfts - ADD
integer , pointer :: pfti(:)             !column's beginning pft index - A
integer , pointer :: snl(:)              !number of snow layers
logical , pointer :: do_capsnow(:)        !true => do snow capping
real(r8), pointer :: forc_lwrad(:)        !downward infrared (longwave) rad
real(r8), pointer :: emg(:)              !ground emissivity
real(r8), pointer :: htvpr(:)            !latent heat of vapor of water (o
real(r8), pointer :: t_grnd(:)           !ground temperature (Kelvin)
integer , pointer :: frac_veg_nosno(:)    !fraction of vegetation not cover
real(r8), pointer :: cgrnds(:)           !deriv, of soil sensible heat flu
real(r8), pointer :: cgrndl(:)           !deriv of soil latent heat flux w
real(r8), pointer :: sabg(:)             !solar radiation absorbed by grou
real(r8), pointer :: dlrad(:)            !downward longwave radiation belo
real(r8), pointer :: ulrad(:)            !upward longwave radiation above
real(r8), pointer :: eflx_sh_veg(:)      !sensible heat flux from leaves (
```

```

real(r8), pointer :: qflx_evap_veg(:)    !vegetation evaporation (mm H2O/s)
real(r8), pointer :: qflx_tran_veg(:)    !vegetation transpiration (mm H2O)
real(r8), pointer :: qflx_evap_can(:)    !evaporation from leaves and stem
real(r8), pointer :: wtcol(:)            !pft weight relative to column
real(r8), pointer :: tssbef(:, :)        !soil/snow temperature before upd
real(r8), pointer :: t_soisno(:, :)      !soil temperature (Kelvin)
real(r8), pointer :: h2soi_ice(:, :)     !ice lens (kg/m2) (new)
real(r8), pointer :: h2soi_liq(:, :)     !liquid water (kg/m2) (new)
local pointers to implicit inout arguments
real(r8), pointer :: eflx_sh_grnd(:)      !sensible heat flux from ground (
real(r8), pointer :: qflx_evap_soil(:)    !soil evaporation (mm H2O/s) (+ =
real(r8), pointer :: qflx_snowcap(:)      !excess precipitation due to snow
local pointers to implicit out arguments
real(r8), pointer :: dt_grnd(:)           !change in t_grnd, last iteration
real(r8), pointer :: eflx_soil_grnd(:)    !soil heat flux (W/m**2) [+ = int
real(r8), pointer :: eflx_sh_tot(:)       !total sensible heat flux (W/m**2
real(r8), pointer :: qflx_evap_tot(:)     !qflx_evap_soil + qflx_evap_veg +
real(r8), pointer :: eflx_lh_tot(:)       !total latent heat flux (W/m8*2)
real(r8), pointer :: qflx_evap_grnd(:)    !ground surface evaporation rate
real(r8), pointer :: qflx_sub_snow(:)     !sublimation rate from snow pack
real(r8), pointer :: qflx_dew_snow(:)     !surface dew added to snow pack (
real(r8), pointer :: qflx_dew_grnd(:)     !ground surface dew formation (mm
real(r8), pointer :: eflx_lwrad_out(:)    !emitted infrared (longwave) radi
real(r8), pointer :: eflx_lwrad_net(:)    !net infrared (longwave) rad (W/m
real(r8), pointer :: eflx_lh_vege(:)      !veg evaporation heat flux (W/m**
real(r8), pointer :: eflx_lh_vegt(:)     !veg transpiration heat flux (W/m
real(r8), pointer :: eflx_lh_grnd(:)     !ground evaporation heat flux (W/
real(r8), pointer :: errsoi_pft(:)        !pft-level soil/lake energy conse
real(r8), pointer :: errsoi_col(:)        !column-level soil/lake energy co

```


A.5 Module BiogeophysicsLakeMod (Source File: BiogeophysicsLakeMod.F90)

Calculates lake temperatures and surface fluxes.

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: BiogeophysicsLake
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.5.1 BiogeophysicsLake

INTERFACE:

```
subroutine BiogeophysicsLake(lbc, ubc, lbp, ubp, num_lakec, filter_lakec, &
                             num_lakep, filter_lakep)
```

DESCRIPTION:

Calculates lake temperatures and surface fluxes.

Lake temperatures are determined from a one-dimensional thermal stratification model based on eddy diffusion concepts to represent vertical mixing of heat.

$$\frac{d\,ts}{dt} = \frac{d}{dz} \left[\frac{1}{(km + ke)} \frac{ds}{dz} \right] + \frac{s}{cw}$$

where: ts = temperature (kelvin)

t = time (s)

z = depth (m)

km = molecular diffusion coefficient (m**2/s)

ke = eddy diffusion coefficient (m**2/s)

cw = heat capacity (j/m**3/kelvin)

s = heat source term (w/m**2)

There are two types of lakes:

Deep lakes are 50 m.

Shallow lakes are 10 m deep.

For unfrozen deep lakes: ke > 0 and convective mixing

For unfrozen shallow lakes: ke = 0 and no convective mixing

Use the Crank-Nicholson method to set up tridiagonal system of equations to solve for ts at time n+1, where the temperature equation for layer i is

$$r_i = a_i [ts_{i-1}]_{n+1} + b_i [ts_i]_{n+1} + c_i [ts_{i+1}]_{n+1}$$

The solution conserves energy as:

$$cw * ([ts(1)]_{n+1} - [ts(1)]_n) * dz(1) / dt + \dots + cw * ([ts(nlevlak)]_{n+1} - [ts(nlevlak)]_n) * dz(nlevlak) / dt = fin$$

```

where:
[ts] n    = old temperature (kelvin)
[ts] n+1  = new temperature (kelvin)
fin       = heat flux into lake (w/m**2)
          = beta*sabg + forc_lwrad - eflx_lwrad_out - eflx_sh_tot - eflx_lh_
            - hm + phi(1) + ... + phi(nlevlak)
WARNING: This subroutine assumes lake columns have one and only one pft.

```

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use time_manager      , only : get_step_size
use clm_varpar        , only : nlevlak
use clm_varcon        , only : hvap, hsub, hfus, cpair, cpliq, tkwat, tk
                        ,      sb, vkc, grav, denh2o, tfrz, spval
use QSatMod           , only : QSat
use FrictionVelocityMod, only : FrictionVelocity, MoninObukIni
use TridiagonalMod    , only : Tridiagonal

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbc, ubc          ! column-index bounds
integer, intent(in) :: lbp, ubp          ! pft-index bounds
integer, intent(in) :: num_lakec         ! number of column non-lak
integer, intent(in) :: filter_lakec(ubc-lbc+1) ! column filter for non-la
integer, intent(in) :: num_lakep         ! number of column non-lak
integer, intent(in) :: filter_lakep(ubp-lbp+1) ! pft filter for non-lake

```

CALLED FROM:

```

subroutine driver

```

REVISION HISTORY:

```

Author: Gordon Bonan
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
Migrated to clm2.1 new data structures by Peter Thornton and M. Vertenstein

```

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: pcolumn(:)          ! pft's column index
integer , pointer :: pgridcell(:)        ! pft's gridcell index
integer , pointer :: cgridcell(:)        ! column's gridcell index
real(r8), pointer :: forc_t(:)           ! atmospheric temperature (Kelvin)
real(r8), pointer :: forc_pbot(:)        ! atmospheric pressure (Pa)
real(r8), pointer :: forc_hgt(:)         ! atmospheric reference height (m)
real(r8), pointer :: forc_hgt_q(:)       ! observational height of humidit
real(r8), pointer :: forc_hgt_t(:)       ! observational height of tempera
real(r8), pointer :: forc_hgt_u(:)       ! observational height of wind [m
real(r8), pointer :: forc_th(:)          ! atmospheric potential temperatu
real(r8), pointer :: forc_q(:)           ! atmospheric specific humidity (
real(r8), pointer :: forc_u(:)           ! atmospheric wind speed in east
real(r8), pointer :: forc_v(:)           ! atmospheric wind speed in north

```

```

real(r8), pointer :: forc_lwrad(:)      ! downward infrared (longwave) ra
real(r8), pointer :: forc_rho(:)       ! density (kg/m**3)
real(r8), pointer :: forc_snow(:)      ! snow rate [mm/s]
real(r8), pointer :: forc_rain(:)      ! rain rate [mm/s]
real(r8), pointer :: t_grnd(:)         ! ground temperature (Kelvin)
real(r8), pointer :: h2osno(:)         ! snow water (mm H2O)
real(r8), pointer :: snowdp(:)         ! snow height (m)
real(r8), pointer :: sabg(:)           ! solar radiation absorbed by gro
real(r8), pointer :: lat(:)            ! latitude (radians)
real(r8), pointer :: dz(:, :)          ! layer thickness (m)
real(r8), pointer :: z(:, :)           ! layer depth (m)
local pointers to implicit out arguments
real(r8), pointer :: qflx_prec_grnd(:) ! water onto ground including can
real(r8), pointer :: qflx_evap_soi(:)  ! soil evaporation (mm H2O/s) (+
real(r8), pointer :: qflx_evap_tot(:)  ! qflx_evap_soi + qflx_evap_veg +
real(r8), pointer :: eflx_sh_grnd(:)    ! sensible heat flux from ground
real(r8), pointer :: eflx_lwrad_out(:)  ! emitted infrared (longwave) rad
real(r8), pointer :: eflx_lwrad_net(:)  ! net infrared (longwave) rad (W/
real(r8), pointer :: eflx_soil_grnd(:)  ! soil heat flux (W/m**2) [+ = in
real(r8), pointer :: eflx_sh_tot(:)     ! total sensible heat flux (W/m**
real(r8), pointer :: eflx_lh_tot(:)     ! total latent heat flux (W/m8*2)
real(r8), pointer :: eflx_lh_grnd(:)    ! ground evaporation heat flux (W
real(r8), pointer :: t_veg(:)           ! vegetation temperature (Kelvin)
real(r8), pointer :: t_ref2m(:)         ! 2 m height surface air temperat
real(r8), pointer :: q_ref2m(:)         ! 2 m height surface specific hum
real(r8), pointer :: taux(:)            ! wind (shear) stress: e-w (kg/m/
real(r8), pointer :: tauy(:)            ! wind (shear) stress: n-s (kg/m/
real(r8), pointer :: qmelt(:)           ! snow melt [mm/s]
real(r8), pointer :: ram1(:)            ! aerodynamical resistance (s/m)
real(r8), pointer :: errsoi(:)          ! soil/lake energy conservation e
real(r8), pointer :: t_lake(:, :)       ! lake temperature (Kelvin)

```


A.6 Module CanopyFluxesMod

(Source File: CanopyFluxesMod.F90)

Calculates the leaf temperature and the leaf fluxes, transpiration, photosynthesis and updates the dew accumulation due to evaporation.

USES:

```
use abortutils,    only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: CanopyFluxes !Calculates the leaf temperature and leaf fluxes
!PRIVATE MEMBER FUNCTIONS:
private :: Stomata      !Leaf stomatal resistance and leaf photosynthesis
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.6.1 CanopyFluxes

INTERFACE:

```
subroutine CanopyFluxes(lbg, ubg, lbc, ubc, lbp, ubp, &
                        num_nolakep, filter_nolakep)
```

DESCRIPTION:

1. Calculates the leaf temperature:
2. Calculates the leaf fluxes, transpiration, photosynthesis and updates the dew accumulation due to evaporation.

Method:

Use the Newton-Raphson iteration to solve for the foliage temperature that balances the surface energy budget:

$$f(t_{veg}) = \text{Net radiation} - \text{Sensible} - \text{Latent} = 0$$

$$f(t_{veg}) + d(f)/d(t_{veg}) * dt_{veg} = 0 \quad (*)$$

Note:

- (1) In solving for t_{veg} , t_{grnd} is given from the previous timestep.
- (2) The partial derivatives of aerodynamical resistances, which cannot be determined analytically, are ignored for $d(H)/dT$ and $d(LE)/dT$
- (3) The weighted stomatal resistance of sunlit and shaded foliage is used
- (4) Canopy air temperature and humidity are derived from $\Rightarrow H_c + H_g = H_a$
 $\Rightarrow E_c + E_g = E_a$
- (5) Energy loss is due to: numerical truncation of energy budget equation (*); and "ecidif" (see the code) which is dropped into the sensible heat

- (6) The convergence criteria: the difference, $\text{del} = t_{\text{veg}}(n+1) - t_{\text{veg}}(n)$ and $\text{del2} = t_{\text{veg}}(n) - t_{\text{veg}}(n-1)$ less than 0.01 K, and the difference of water flux from the leaf between the iteration step (n+1) and (n) less than 0.1 W/m²; or the iterative steps over 40.

USES:

```

use shr_kind_mod      , only: r8 => shr_kind_r8
use clmtype
use time_manager      , only : get_step_size
use clm_varpar        , only : nlevsoi, nlevsno
use clm_varcon        , only : sb, cpair, hvap, vkc, grav, denice, &
                        denh2o, tfrz, csoilc
use QSatMod           , only : QSat
use FrictionVelocityMod, only : FrictionVelocity, MoninObukIni

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbg, ubg      ! gridcell bounds
integer, intent(in) :: lbc, ubc      ! column bounds
integer, intent(in) :: lbp, ubp      ! pft bounds
integer, intent(in) :: num_nolakep    ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-1

```

CALLED FROM:

```

subroutine Biogeophysics1 in module Biogeophysics1Mod

```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 12/19/01, Peter Thornton
 Changed tg to t_grnd for consistency with other routines
 1/29/02, Peter Thornton
 Migrate to new data structures, new calling protocol. For now co2 and o2 partial pressures are hardwired, but they should be coming in from forc_co2 and forc_o2. Keeping the same hardwired values as in CLM2 to assure bit-for-bit results in the first comparisons.

LOCAL VARIABLES:

```

local pointers to implicit in variables
integer , pointer :: ivt(:)      ! pft vegetation type
integer , pointer :: pcolumn(:) ! pft's column index
integer , pointer :: plandunit(:) ! pft's landunit index
integer , pointer :: pgridcell(:) ! pft's gridcell index
real(r8), pointer :: forc_th(:) ! atmospheric potential temperature (K)
real(r8), pointer :: t_grnd(:) ! ground surface temperature [K]
real(r8), pointer :: thm(:) ! intermediate variable (forc_t+0.0098)
real(r8), pointer :: qg(:) ! specific humidity at ground surface
real(r8), pointer :: thv(:) ! virtual potential temperature (kelvi)
real(r8), pointer :: z0mv(:) ! roughness length over vegetation, mo
real(r8), pointer :: z0hv(:) ! roughness length over vegetation, se
real(r8), pointer :: z0qv(:) ! roughness length over vegetation, la
real(r8), pointer :: z0mg(:) ! roughness length of ground, momentum

```

```

real(r8), pointer :: dqgdT(:)      ! temperature derivative of "qg"
real(r8), pointer :: htvpt(:)     ! latent heat of evaporation (/sublima
real(r8), pointer :: emv(:)       ! ground emissivity
real(r8), pointer :: emg(:)       ! vegetation emissivity
real(r8), pointer :: forc_pbot(:)  ! atmospheric pressure (Pa)
real(r8), pointer :: forc_q(:)    ! atmospheric specific humidity (kg/kg)
real(r8), pointer :: forc_u(:)    ! atmospheric wind speed in east direc
real(r8), pointer :: forc_v(:)    ! atmospheric wind speed in north dire
real(r8), pointer :: forc_hgt_u(:) ! observational height of wind [m]
real(r8), pointer :: forc_rho(:)   ! density (kg/m**3)
! real(r8), pointer :: forc_co2(:)  ! atmospheric CO2 concentration (Pa)
! real(r8), pointer :: forc_o2(:)  ! atmospheric O2 concentration (Pa)
real(r8), pointer :: forc_lwrad(:) ! downward infrared (longwave) radiati
real(r8), pointer :: displa(:)    ! displacement height (m)
real(r8), pointer :: parsun(:)    ! average absorbed PAR for sunlit leav
real(r8), pointer :: parsha(:)    ! average absorbed PAR for shaded leav
real(r8), pointer :: elai(:)      ! one-sided leaf area index with buryi
real(r8), pointer :: esai(:)      ! one-sided stem area index with buryi
real(r8), pointer :: fdry(:)      ! fraction of foliage that is green an
real(r8), pointer :: fwet(:)      ! fraction of canopy that is wet (0 to
real(r8), pointer :: laisun(:)    ! sunlit leaf area
real(r8), pointer :: laisha(:)    ! shaded leaf area
integer , pointer :: frac_veg_nosno(:) ! frac of veg not covered by snow
real(r8), pointer :: sabv(:)      ! solar radiation absorbed by vegetati
real(r8), pointer :: watsat(:, :) ! volumetric soil water at saturation
real(r8), pointer :: h2soi_ice(:, :) ! ice lens (kg/m2)
real(r8), pointer :: h2soi_liq(:, :) ! liquid water (kg/m2)
real(r8), pointer :: dz(:, :)    ! layer depth (m)
real(r8), pointer :: t_soisno(:, :) ! soil temperature (Kelvin)
real(r8), pointer :: sucsat(:, :) ! minimum soil suction (mm)
real(r8), pointer :: bsw(:, :)   ! Clapp and Hornberger "b"
real(r8), pointer :: rootfr(:, :) ! fraction of roots in each soil layer
real(r8), pointer :: smpmax(:)   ! wilting point potential in mm
real(r8), pointer :: dleaf(:)    ! characteristic leaf dimension (m)
local pointers to implicit inout arguments
real(r8), pointer :: cgrnds(:)   ! deriv. of soil sensible heat flux wr
real(r8), pointer :: cgrndl(:)   ! deriv. of soil latent heat flux wrt
real(r8), pointer :: t_veg(:)    ! vegetation temperature (Kelvin)
real(r8), pointer :: t_ref2m(:)  ! 2 m height surface air temperature (
real(r8), pointer :: q_ref2m(:)  ! 2 m height surface specific humidity
real(r8), pointer :: h2ocan(:)   ! canopy water (mm H2O)
#ifdef DGVM
real(r8), pointer :: annpsnpot(:) ! annual potential photosynthesis (umo
real(r8), pointer :: annpsn(:)    ! annual photosynthesis (umol CO2 /m**
#else
real(r8) :: annpsnpot(lbp:ubp)   ! temporary, set to zero
real(r8) :: annpsn(lbp:ubp)     ! temporary, set to zero
#endif
local pointers to implicit out arguments
real(r8), pointer :: cgrnd(:)    ! deriv. of soil energy flux wrt t
real(r8), pointer :: dlrاد(:)    ! downward longwave radiation belo
real(r8), pointer :: ulrad(:)    ! upward longwave radiation above
real(r8), pointer :: ram1(:)     ! aerodynamical resistance (s/m)
real(r8), pointer :: btran(:)    ! transpiration wetness factor (0

```



```

real(r8), pointer :: rssun(:)           ! sunlit stomatal resistance (s/m)
real(r8), pointer :: rssha(:)          ! shaded stomatal resistance (s/m)
real(r8), pointer :: psnsun(:)         ! sunlit leaf photosynthesis (umol
real(r8), pointer :: psnsa(:)          ! shaded leaf photosynthesis (umol
real(r8), pointer :: qflx_tran_veg(:)  ! vegetation transpiration (mm H2O
real(r8), pointer :: dt_veg(:)         ! change in t_veg, last iteration
real(r8), pointer :: qflx_evap_veg(:)  ! vegetation evaporation (mm H2O/s)
real(r8), pointer :: eflx_sh_veg(:)    ! sensible heat flux from leaves (
real(r8), pointer :: tauw(:)           ! wind (shear) stress: e-w (kg/m/s)
real(r8), pointer :: tauy(:)           ! wind (shear) stress: n-s (kg/m/s)
real(r8), pointer :: eflx_sh_grnd(:)   ! sensible heat flux from ground (
real(r8), pointer :: qflx_evap_soil(:) ! soil evaporation (mm H2O/s) (+ =
real(r8), pointer :: fpsn(:)           ! photosynthesis (umol CO2 /m**2 /
real(r8), pointer :: rootr(:, :)       ! effective fraction of roots in e

```

A.6.2 Stomata

INTERFACE:

```

subroutine Stomata (fn, filterp, lbp, ubp, lbc, ubc, lbg, ubg, &
                  apar, ei, ea, tgcm, forc_pbot, &
                  tl, o2, co2, btran, rb, &
                  rs, psn, annpsnpot, annpsn)

```

DESCRIPTION:

Leaf stomatal resistance and leaf photosynthesis.

USES:

```

use shr_kind_mod , only : r8 => shr_kind_r8
use shr_const_mod, only : SHR_CONST_TKFRZ, SHR_CONST_RGAS
use clmtype

```

ARGUMENTS:

```

implicit none
integer , intent(in)   :: fn           ! size of pft filter
integer , intent(in)   :: filterp(fn)  ! pft filter
integer , intent(in)   :: lbp, ubp     ! pft bounds
integer , intent(in)   :: lbc, ubc     ! column bounds
integer , intent(in)   :: lbg, ubg     ! gridcell bounds
real(r8), intent(in)   :: apar(lbp:ubp) ! par absorbed per unit la
real(r8), intent(in)   :: ei(lbp:ubp)  ! vapor pressure inside le
real(r8), intent(in)   :: ea(lbp:ubp)  ! vapor pressure of canopy
real(r8), intent(in)   :: tgcm(lbc:ubc) ! air temperature at agcm
real(r8), intent(in)   :: forc_pbot(lbg:ubg) ! atmospheric pressure (Pa)
real(r8), intent(in)   :: tl(lbp:ubp)  ! leaf temperature (Kelvin)
real(r8), intent(in)   :: o2(lbp:ubp)  ! atmospheric o2 concentra
real(r8), intent(in)   :: co2(lbp:ubp)  ! atmospheric co2 concentr
real(r8), intent(in)   :: btran(lbp:ubp) ! soil water transpiration
real(r8), intent(inout) :: rb(lbp:ubp)  ! boundary layer resistanc

```

```

real(r8), intent(inout) :: rs(lbp:ubp)      ! leaf stomatal resistance
real(r8), intent(inout) :: psn(lbp:ubp)     ! foliage photosynthesis (
real(r8), intent(inout) :: annpsnpot(lbp:ubp) ! annual potential photosy
real(r8), intent(inout) :: annpsn(lbp:ubp)   ! annual photosynthesis (u

```

CALLED FROM:

```

subroutine CanopyFluxes in this module

```

REVISION HISTORY:

```

author:          Gordon Bonan
standardized:    J. Truesdale, Feb. 1996
reviewed:        G. Bonan, Feb. 1996
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
1/30/02, PET Made all input and output parameters explicit (removed refere

```

LOCAL VARIABLES:

```

local pointers to implicit in variables
integer , pointer :: pcolumn(:)      ! pft's column index
integer , pointer :: pgridcell(:)    ! pft's gridcell index
integer , pointer :: ivt(:)          ! pft vegetation type
real(r8), pointer :: qe25(:)         ! quantum efficiency at 25C (umol CO
real(r8), pointer :: vcmx25(:)       ! max rate of carboxylation at 25C (
real(r8), pointer :: c3psn(:)        ! photosynthetic pathway: 0. = c4, 1
real(r8), pointer :: mp(:)           ! slope of conductance-to-photosynth

```


A.7 Module DGVMAllocationMod (Source File: DGVMAllocationMod.F90)

Performs yearly allocation calculation

USES:

use shr_kind_mod, only: r8 => shr_kind_r8

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Allocation
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.7.1 Allocation

INTERFACE:

```
subroutine Allocation (lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Performs yearly allocation calculation

Allocation of this year's biomass increment (bm_inc_ind) to the three living carbon pools, such that the basic allometric relationships (A-C below) are always satisfied.

TREE ALLOCATION

```
(A) (leaf area) = latosa * (sapwood xs area)
    (Pipe Model, Shinozaki et al. 1964a,b; Waring et al 1982)
(B) (leaf mass) = lmtorm * (root mass)
(C) height = allom2 * (stem diameter)**allom3 (source?)
(D) (crown area) = min (allom1 * (stem diameter)**reinickerp, crownarea_max)
Mathematical derivation:
(1) bm_inc_ind = lminc_ind + sminc_ind + rminc_ind
(2) leaf_area_new = latosa * sap_xsa_new [from (A)]
(3) leaf_area_new = (lm_ind + lminc_ind) * sla
from (2) & (3),
(4) (lm_ind + lminc_ind) * sla = latosa * sap_xsa_new
from (4),
(5) sap_xsa_new = (lm_ind + lminc_ind) * sla / latosa
(6) (lm_ind + lminc_ind) = lmtorm * (rm_ind + rminc_ind) [from (B)]
(7) height_new = allom2 * stemdiam_new**allom3 [from (C)]
from (1),
(8) sminc_ind = bm_inc_ind - lminc_ind - rminc_ind
```

```

from (6),
  (9) rminc_ind=((lm_ind + lminc_ind) / lmtorm) - rm_ind
from (8) & (9),
  (10) sminc_ind = bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind
  (11) wooddens = (sm_ind + sminc_ind + hm_ind) / stemvolume_new
  (12) stemvolume_new = height_new * pi * stemdiam_new**2 / 4
from (10), (11) & (12)
  (13) stemdiam_new = [ ((sm_ind + bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind + hm_ind)
      / wooddens) / (height_new * pi / 4) ]**(1/2)
combining (7) and (13),
  (14) height_new = allom2 * [ ((sm_ind + bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind + hm_ind)
      / wooddens) / (height_new * pi / 4) ]**(1/2 * allom3)
from (14),
  (15) height_new**(1 + 2 / allom3) = allom2**(2 / allom3)
      * ((sm_ind + bm_inc_ind - lminc_ind - ((lm_ind + lminc_ind)
      / lmtorm) + rm_ind + hm_ind) / wooddens) / (pi / 4)
  (16) wooddens = (sm_ind + sminc_ind) / sapvolume_new
from (10) and (16),
  (17) wooddens = (sm_ind + bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind) / sapvolume_new
  (18) sapvolume_new = height_new * sap_xsa_new
from (17) and (18),
  (19) sap_xsa_new = (sm_ind + bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind)
      / (height_new * wooddens)
from (19),
  (20) height_new = (sm_ind + bm_inc_ind - lminc_ind
      - ((lm_ind + lminc_ind) / lmtorm) + rm_ind )
      / (sap_xsa_new * wooddens)
from (5) and (20),
  (21) height_new**(1 + 2 / allom3) = [ (sm_ind + bm_inc_ind
      - lminc_ind - ((lm_ind + lminc_ind) / lmtorm) + rm_ind )
      / ((lm_ind + lminc_ind) * sla * wooddens / latosa) ]
      *(1 + 2 / allom3)

```

(15) and (21) are two alternative expressions for
height_new**(1 + 2 / allom3). Combining these,

```

(22) allom2**(2 / allom3) * ((sm_ind + bm_inc_ind - lminc_ind
    - ((lm_ind + lminc_ind) / lmtorm) + rm_ind + hm_ind)
    / wooddens) / (pi / 4) - [ (sm_ind + bm_inc_ind - lminc_ind
    - ((lm_ind + lminc_ind) / lmtorm) + rm_ind )
    / ((lm_ind + lminc_ind) * sla * wooddens / latosa) ]
    *(1 + 2 / allom3)
    = 0

```

Equation (22) can be expressed in the form $f(lminc_ind)=0$.
Numerical methods are used to solve the equation for the
unknown $lminc_ind$.

Work out minimum leaf production to maintain current sapmass

```

(23) sap_xsa = sm_ind / wooddens / height

```

```

from (A) and (23),
  (24) leaf_mass * sla = latosa * sap_mass / wooddens / height
from (24),
  (25) leaf_mass = latosa * sap_mass / (wooddens * height * sla)
from (25), assuming sminc_ind=0,
  (26) lm_ind + lminc_ind_min = latosa * sm_ind
      / (wooddens * height * sla)
from (26),
  (27) lminc_ind_min = latosa * sm_ind / (wooddens * height * sla)
      - lm_ind
Work out minimum root production to support this leaf mass
(i.e. lm_ind + lminc_ind_min)
May be negative following a reduction in soil water limitation
(increase in lmtorm) relative to last year.

from (B) and (25),
  (28) root_mass = latosa * sap_mass / (wooddens * height * sla)
      / lmtorm
from (28), assuming sminc_ind=0,
  (29) rm_ind + rminc_ind_min = latosa * sm_ind
      / (wooddens * height * sla * lmtorm)
from (29),
  (30) rminc_ind_min = latosa * sm_ind
      / (wooddens * height * sla * lmtorm) - rm_ind
-----
Attempt to distribute this year's production among leaves and roots only
  (31) bm_inc_ind = lminc_ind + rminc_ind
from (31) and (9),
  (32) bm_inc_ind = lminc_ind + ((lm_ind + lminc_ind) / lmtorm)
      - rm_ind
from (32)
  (33) lminc_ind = (bm_inc_ind - lm_ind / lmtorm + rm_ind) /
      (1 + 1 / lmtorm)
-----
from (25),
  (34) lm_ind + lminc_ind = latosa * (sm_ind + sminc_ind)
      / (wooddens * height * sla)
from (34),
  (35) sminc_ind = (lm_ind + lminc_ind) * wooddens * height * sla
      / latosa - sm_ind
-----

```

USES:

```

use clmtype
use shr_const_mod, ONLY: SHR_CONST_PI

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natveg        ! number of naturally-ve
integer, intent(in) :: filter_natveg     ! pft filter for natural

```

CALLED FROM:

```

subroutine lpj in module DGVMMod

```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. allocation)

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)           ! pft vegetation type
  real(r8), pointer :: sla(:)           ! ecophys const - specific leaf a
  logical , pointer :: tree(:)          ! ecophys const - whether this pf
  real(r8), pointer :: allom1(:)        ! ecophys const - parameter in al
  real(r8), pointer :: allom2(:)        ! ecophys const - parameter in al
  real(r8), pointer :: allom3(:)        ! ecophys const - parameter in al
  real(r8), pointer :: latosa(:)        ! ecophys const - ratio of leaf a
  real(r8), pointer :: wooddens(:)      ! ecophys const - wood density (g
  real(r8), pointer :: reinickerp(:)    ! ecophys const - parameter in al
  real(r8), pointer :: crownarea_max(:) ! ecophys const - tree maximum cr
  real(r8), pointer :: init_lmtorm(:)   ! ecophys const - leaf:root ratio
  real(r8), pointer :: bm_inc(:)        ! biomass increment
  real(r8), pointer :: nind(:)          ! number of individuals (#/m**2)
  real(r8), pointer :: annpsn(:)        ! annual photosynthesis (umol CO2
  real(r8), pointer :: annpsnpot(:)     ! annual potential photosynthesis

local pointers to implicit inout arguments
  real(r8), pointer :: fpc_grid(:)      ! foliar projective cover on gri
  real(r8), pointer :: crownarea(:)     ! area that each individual tree
  real(r8), pointer :: height(:)        ! canopy top (m)
  real(r8), pointer :: lm_ind(:)        ! individual leaf mass
  real(r8), pointer :: sm_ind(:)        ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)        ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)        ! individual root mass
  real(r8), pointer :: litter_ag(:)     ! above ground litter
  real(r8), pointer :: litter_bg(:)     ! below ground litter

local pointers to implicit out arguments
  real(r8), pointer :: lai_ind(:)       ! LAI per individual
  real(r8), pointer :: fpc_inc(:)       ! foliar projective cover increm

```

A.8 Module DGVMEcosystemDynMod (Source File: DGVMEcosystemDynMod.F90)

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: DGVMEcosystemDynini ! LPJ and other DGVM related initializations
public :: DGVMEcosystemDyn    ! Ecosystem dynamics: phenology, vegetation
public :: DGVMRrespiration    ! Compute plant respiration
```

PUBLIC MEMBER FUNCTIONS:

```
private :: Phenology      ! Compute summer and drought phenology
private :: FireSeason     ! Calculate length of fire season in a year
private :: LitterSOM      ! Calculate litter and soil decomposition
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.8.1 DGVMEcosystemDynini

INTERFACE:

```
subroutine DGVMEcosystemDynini()
```

DESCRIPTION:

LPJ and other DGVM related initializations

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use nanMod
use clmtype
use decompMod, only : get_proc_bounds, get_proc_global
use clm_varpar, only : numpft, npftpar, maxpatch_pft
use shr_const_mod, only : SHR_CONST_PI, SHR_CONST_TKFRZ
use pftvarcon, only : pftpar, tree, summergreen, raingreen, sla, &
                    lm_sapl, sm_sapl, hm_sapl, rm_sapl, latosa, &
                    allom1, allom2, allom3, reinickerp, wooddens, &
                    noveg
```

ARGUMENTS:


```
implicit none
```

CALLED FROM:

```
subroutine initialize in module initializeMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from LPJ initialization subroutines)

A.8.2 DGVMEcosystemDyn

INTERFACE:

```
subroutine DGVMEcosystemDyn(lbp, ubp, num_nolakep, filter_nolakep, &
                           doalb, endofyr)
```

DESCRIPTION:

Ecosystem dynamics: phenology, vegetation
 Calculates leaf areas (tlai, elai), stem areas (tsai, esai) and
 height (htop)

USES:

```
use clmtype
use shr_const_mod, only: SHR_CONST_CDAY
use time_manager, only : get_step_size, get_nstep, get_curr_date
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_nolakep       ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-1
logical, intent(in) :: doalb             ! true = surface albed
logical, intent(in) :: endofyr
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

Author: Gordon Bonan
 2/1/02, Peter Thornton: Migrated to new data structure.

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: ivt(:)      ! vegetation type for this pft
integer , pointer :: pcolumn(:) ! column index of corresponding pft
real(r8), pointer :: lai_ind(:) ! LAI per individual
real(r8), pointer :: dphen(:)   ! phenology [0 to 1]
real(r8), pointer :: fpcgrid(:) ! foliar projective cover on gridcell (f
real(r8), pointer :: snowdp(:)  ! snow height (m) (column-level)
```

```

local pointers to implicit in/out arguments
  real(r8), pointer :: htop(:)      ! canopy top (m)
local pointers to implicit out arguments
  real(r8), pointer :: tlai(:)      ! one-sided leaf area index, no burying
  real(r8), pointer :: tsai(:)      ! one-sided stem area index, no burying
  real(r8), pointer :: hbot(:)      ! canopy bottom (m)
  real(r8), pointer :: elai(:)      ! one-sided leaf area index with burying
  real(r8), pointer :: esai(:)      ! one-sided stem area index with burying
  integer , pointer :: frac_veg_nosno_alb(:) ! frac of vegetation not cover

```

A.8.3 DGVMRespiration

INTERFACE:

```

subroutine DGVMRespiration(lbc, ubc, lbp, ubp, &
                           num_nolakec, filter_nolakec, &
                           num_nolakep, filter_nolakep)

```

DESCRIPTION:

Calculates surface biogeochemical fluxes

USES:

```

use clmtype
use shr_const_mod , only: SHR_CONST_CDAY, SHR_CONST_TKFRZ
use clm_varpar    , only : nlevsoi
use time_manager  , only : get_step_size

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbc, ubc      ! column bounds
integer, intent(in) :: lbp, ubp      ! pft bounds
integer, intent(in) :: num_nolakec    ! number of column non
integer, intent(in) :: filter_nolakec(num_nolakec) ! column filter for no
integer, intent(in) :: num_nolakep    ! number of column non
integer, intent(in) :: filter_nolakep(num_nolakep) ! pft filter for non-l

```

CALLED FROM:

```

subroutine driver

```

REVISION HISTORY:

Author: Gordon Bonan and Sam Levis
 1/31/02, PET: migrated to new data structures

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: pcolumn(:)    ! pft's column
  integer , pointer :: ivt(:)         ! vegetation type for current pft
  real(r8), pointer :: t_veg(:)       ! vegetation temperature (Kelvin)

```

```

real(r8), pointer :: fpcgrid(:)      ! foliar projective cover on gridcell
real(r8), pointer :: nind(:)         ! number of individuals
real(r8), pointer :: dphen(:)        ! phenology [0 to 1]
real(r8), pointer :: lm_ind(:)       ! individual leaf mass
real(r8), pointer :: sm_ind(:)       ! individual stem mass
real(r8), pointer :: rm_ind(:)       ! individual root mass
real(r8), pointer :: respcoeff(:)    ! respiration coefficient (LPJ)
real(r8), pointer :: l_cton(:)       ! c/n for leaves (LPJ)
real(r8), pointer :: s_cton(:)       ! c/n for stems (LPJ)
real(r8), pointer :: r_cton(:)       ! c/n for roots (LPJ)
real(r8), pointer :: z(:, :)         ! layer thickness (m)
real(r8), pointer :: dz(:, :)        ! layer depth (m)
real(r8), pointer :: t_soisno(:, :)  ! soil temperature (Kelvin)
real(r8), pointer :: fpsn(:)         ! photosynthesis (umol CO2 /m**2 /s)
real(r8), pointer :: fmicr(:)        ! microbial respiration (umol CO2 /m**2 /s)

local pointers to implicit inout arguments
real(r8), pointer :: bm_inc(:)       ! biomass increment
real(r8), pointer :: afmicr(:)       ! annual microbial respiration

local pointers to implicit out arguments
real(r8), pointer :: frmf(:)         ! leaf maintenance respiration (umol /m**2 /s)
real(r8), pointer :: frms(:)         ! stem maintenance respiration (umol /m**2 /s)
real(r8), pointer :: frmr(:)         ! root maintenance respiration (umol /m**2 /s)
real(r8), pointer :: frm(:)          ! total maintenance respiration (umol /m**2 /s)
real(r8), pointer :: frg(:)          ! growth respiration (umol CO2 /m**2 /s)
real(r8), pointer :: dmi(:)          ! total dry matter production (ug /m**2 /s)
real(r8), pointer :: fco2(:)         ! net CO2 flux (umol CO2 /m**2 /s) [+/-]
real(r8), pointer :: tsoi25(:)       ! soil temperature to 0.25 m (Kelvin)

```

A.8.4 Phenology

INTERFACE:

```
subroutine Phenology (lbp, ubp, num_nolakep, filter_nolakep)
```

DESCRIPTION:

Summer and drought phenology. Called once per day.

USES:

```

use clmtype
use shr_const_mod, only : SHR_CONST_TKFRZ

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbp, ubp          ! pft bounds
integer, intent(in) :: num_nolakep       ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-1

```

CALLED FROM:

```
subroutine Ecosysdyn in this module
```

REVISION HISTORY:

Author: Sam Levis (adapted from Jon Foley's IBIS subroutine pheno)
 2/1/02, Peter Thornton: Migrated to new data structure

LOCAL VARIABLES:

```

local pointers to implicit in scalars
  integer , pointer :: ivt(:)          ! vegetation type for this pft
  real(r8), pointer :: t10(:)          ! 10-day running mean of the 2 m temp
  real(r8), pointer :: agdd0(:)        ! accumulated growing degree days abo
  real(r8), pointer :: agdd5(:)        ! accumulated growing degree days abo
  real(r8), pointer :: fnpsn10(:)      ! 10-day running mean net photosynthe
  real(r8), pointer :: tmomin20(:)     ! 20 year running mean of monthly min
  real(r8), pointer :: l_long(:)       ! ecophys constant - leaf longevity [
  logical , pointer :: tree(:)         ! ecophys constant
  logical , pointer :: raingreen(:)    ! ecophys constant
  logical , pointer :: summergreen(:) ! ecophys constant
local pointers to implicit in/out scalars
  real(r8), pointer :: t10min(:)       ! annual minimum of 10-day running me
  real(r8), pointer :: leafon(:)       ! leafon days
  real(r8), pointer :: leafof(:)       ! leafoff days
local pointers to implicit out scalars
  real(r8), pointer :: dphen(:)        ! phenology [0 to 1]
```

A.8.5 FireSeason

INTERFACE:

```
subroutine FireSeason (lbp, ubp, num_nolakep, filter_nolakep)
```

DESCRIPTION:

Calculate length of fire season in a year
 Orig. code was called once per day.
 slevis adapted to call every timestep.
 Orig. code operated on a grid cell basis.
 slevis adapted to operate on a patch basis.

USES:

```

use clmtype
use time_manager, only : get_step_size
use shr_const_mod, only : SHR_CONST_PI, SHR_CONST_CDAY, SHR_CONST_TKFRZ
```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbp, ubp          ! pft bounds
integer, intent(in) :: num_nolakep      ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-l
```

CALLED FROM:

subroutine Ecosysdyn in this module

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subroutine fire)

LOCAL VARIABLES:

```

local pointers toimplicit in arguments
  integer , pointer :: pcolumn(:)      ! column index for corresponding pft
  integer , pointer :: ivt(:)          ! vegetation type for this pft
  real(r8), pointer :: t_ref2m(:)      ! 2 m height surface air temperature
  real(r8), pointer :: litterag(:)     ! above ground litter
  real(r8), pointer :: wf(:)           ! soil water as frac. of whc for top
  real(r8), pointer :: flam(:)        ! ecophys constant - flammability th
local pointers toimplicit in/out arguments
  real(r8), pointer :: firelength(:)   ! fire season in days

```

A.8.6 LitterSOM

INTERFACE:

```
subroutine LitterSOM (lbp, ubp, num_nolakep, filter_nolakep, kyr)
```

DESCRIPTION:

Litter and soil decomposition
 Incorporates analytical solution for soil pool sizes
 once litter inputs are (assumed to be) at equilibrium,
 reducing spin-up time for carbon fluxes due to soil respiration.

USES:

```

use clmtype
use time_manager , only : get_step_size
use shr_const_mod, only : SHR_CONST_CDAY, SHR_CONST_TKFRZ

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbp, ubp          ! pft bounds
integer, intent(in) :: num_nolakep      ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-l
integer, intent(in) :: kyr              ! year (0, ...) for ns

```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subroutine littersom)
 2/1/02, Peter Thornton: Migrate to new data structures

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: pcolumn(:)      ! column index for corresponding
  real(r8), pointer :: wf(:)           ! soil water as frac. of whc for
  real(r8), pointer :: tsoi25(:)       ! soil temperature to 0.25 m (Ke
local pointers to implicit in/out arguments
  real(r8), pointer :: litterag(:)     ! above ground litter
  real(r8), pointer :: litterbg(:)     ! below ground litter
  real(r8), pointer :: cpool_fast(:)   ! fast carbon pool
  real(r8), pointer :: cpool_slow(:)   ! slow carbon pool
  real(r8), pointer :: k_fast_ave(:)   ! decomposition rate
  real(r8), pointer :: k_slow_ave(:)   ! decomposition rate
  real(r8), pointer :: litter_decom_ave(:) ! decomposition rate
local pointers to implicit out arguments
  real(r8), pointer :: fmicr(:)        ! microbial respiration (umol CO

```


A.9 Module DGVMEestablishmentMod (Source File: DGVMEestablishmentMod.F90)

Calculates establishment of new pfts
Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils, only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Establishment
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.9.1 Establishment

INTERFACE:

```
subroutine Establishment(lbp, ubp, lbq, ubq)
```

DESCRIPTION:

Calculates establishment of new pfts
Called once per year

USES:

```
use clmtype
use clm_varpar , only : numpft
use clm_varcon , only : istsoil
use pftvarcon , only : noveg
use shr_const_mod, only : SHR_CONST_CDAY, SHR_CONST_PI, SHR_CONST_TKFRZ
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp      ! pft bounds
integer , intent(in) :: lbq, ubq      ! gridcell bounds
```

CALLED FROM:

```
subroutine lpj in module DGVMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. establishment)
 3/4/02, Peter Thornton: Migrated to new data structures.

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: plandunit(:)      ! landunit of corresponding pft
  integer , pointer :: pgridcell(:)      ! gridcell of corresponding pft
  integer , pointer :: ltype(:)          ! landunit type for corresponding p
  real(r8), pointer :: wtgcell(:)        ! pft weight relative to grid cell
  real(r8), pointer :: tmomin20(:)       ! 20-yr running mean of tmomin
  real(r8), pointer :: agdd20(:)         ! 20-yr running mean of agdd
  real(r8), pointer :: agddtw(:)         ! accumulated growing degree days a
  real(r8), pointer :: prec365(:)        ! 365-day running mean of tot. prec
  real(r8), pointer :: sla(:)            ! ecophys const - sp. leaf area [m2
  logical , pointer :: tree(:)           ! ecophys const - true=> tree is pr
  real(r8), pointer :: crownarea_max(:)  ! ecophys const - tree maximum crow
  real(r8), pointer :: lm_sapl(:)        ! ecophys const - leaf mass of sapl
  real(r8), pointer :: sm_sapl(:)        ! ecophys const - stem mass of sapl
  real(r8), pointer :: hm_sapl(:)        ! ecophys const - heartwood mass of
  real(r8), pointer :: rm_sapl(:)        ! ecophys const - root mass of sapi
  real(r8), pointer :: reinickerp(:)     ! ecophys const - parameter in allo
  real(r8), pointer :: wooddens(:)       ! ecophys const - wood density (gC/
  real(r8), pointer :: latosa(:)         ! ecophys const - ratio of leaf are
  real(r8), pointer :: allom1(:)         ! ecophys const - parameter in allo
  real(r8), pointer :: allom2(:)         ! ecophys const - parameter in allo
  real(r8), pointer :: allom3(:)         ! ecophys const - parameter in allo
  real(r8), pointer :: tcmin(:)          ! ecophys const - minimum coldest m
  real(r8), pointer :: tcmax(:)          ! ecophys const - maximum coldest m
  real(r8), pointer :: gddmin(:)         ! ecophys const - minimum growing d

local pointers to implicit in/out arguments
  integer , pointer :: ivt(:)            ! vegetation type for this pft
  logical , pointer :: present(:)        ! true=> PFT present in patch
  real(r8), pointer :: nind(:)           ! number of individuals (#/m**2)
  real(r8), pointer :: lm_ind(:)         ! individual leaf mass
  real(r8), pointer :: sm_ind(:)         ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)         ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)         ! individual root mass
  real(r8), pointer :: litterag(:)       ! above ground litter
  real(r8), pointer :: litterbg(:)       ! below ground litter

local pointers to implicit out arguments
  real(r8), pointer :: fpcgrid(:)        ! foliar projective cover on gridce
  real(r8), pointer :: htop(:)           ! canopy top (m)
  real(r8), pointer :: lai_ind(:)        ! LAI per individual
  real(r8), pointer :: crownarea(:)      ! area that each individual tree ta

```

A.10 Module FireMod

(Source File: DGVMFireMod.F90)

Effect of the fire on vegetation structure and litter
Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Fire
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.10.1 Fire

INTERFACE:

```
subroutine Fire(lbp, ubp, afire_frac, acflux_fire)
```

DESCRIPTION:

Effect of the fire on vegetation structure and litter

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp          ! pft bounds
real(r8), intent(out) :: afire_frac(lbp:ubp)
real(r8), intent(out) :: acflux_fire(lbp:ubp)
```

CALLED FROM:

```
subroutine EcosystemDyn in module EcosystemdynMod
subroutine lpj in module DGVMMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subroutine fire)

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)           ! pft vegetation type
  real(r8), pointer :: lm_ind(:)        ! individual leaf mass
  real(r8), pointer :: sm_ind(:)        ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)        ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)        ! individual root mass
  real(r8), pointer :: fpc_grid(:)      ! foliar projective cover on grid
  logical , pointer :: present(:)       ! whether this pft present in pat
  real(r8), pointer :: fire_length(:)   ! fire season in days
  logical , pointer :: tree(:)          ! ecophys const - whether this pf
  real(r8), pointer :: resist(:)        ! ecophys const - fire resistance
local pointers to implicit inout arguments
  real(r8), pointer :: litter_ag(:)      ! above ground litter
  real(r8), pointer :: nind(:)          ! number of individuals (#/m**2)

```

A.11 Module KillMod

(Source File: DGVMKillMod.F90)

Removal of PFTs with negative annual C increment
 NB: PFTs newly beyond their bioclimatic limits are removed in
 subroutine establishment
 Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Kill
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.11.1 Kill

INTERFACE:

```
subroutine Kill(lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Removal of PFTs with negative annual C increment
 NB: PFTs newly beyond their bioclimatic limits are removed in
 subroutine establishment
 Called once per year

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natvegp       ! number of naturally-ve
integer, intent(in) :: filter_natvegp(ubp-lbp+1) ! pft filter for natural
```

CALLED FROM:

```
subroutine lpj in module DGVMMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. kill)

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)           ! pft vegetation type
  real(r8), pointer :: nind(:)          ! number of individuals (#/m**2)
  real(r8), pointer :: lm_ind(:)        ! individual leaf mass
  real(r8), pointer :: sm_ind(:)        ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)        ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)        ! individual root mass
  real(r8), pointer :: bm_inc(:)        ! biomass increment
  logical , pointer :: tree(:)          ! ecophys const - whether this pf
local pointers to implicit inout arguments
  logical , pointer :: present(:)       ! whether PFT present in patch
  real(r8), pointer :: litter_ag(:)     ! above ground litter
  real(r8), pointer :: litter_bg(:)     ! below ground litter

```

A.12 Module LightMod

(Source File: DGVMLightMod.F90)

Calculate light competition
 Update fpc (for establishment routine)
 Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Light
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.12.1 Light

INTERFACE:

```
subroutine Light(lbg, ubg, lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Calculate light competition
 Update fpc (for establishment routine)
 Called once per year

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbg, ubg           ! gridcell bounds
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natvegp        ! number of naturally-ve
integer, intent(in) :: filter_natvegp(ubp-lbp+1) ! pft filter for natural
```

CALLED FROM:

```
subroutine lpj in module DGVMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subroutine light)
 3/4/02, Peter Thornton: Migrated to new data structures.

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)      ! pft vegetation type
  integer , pointer :: pgridcell(:) ! gridcell index of corresponding pft
  real(r8), pointer :: fpcinc(:)   ! foliar projective cover increment (fr
  real(r8), pointer :: sm_ind(:)   ! individual stem mass
  real(r8), pointer :: hm_ind(:)   ! individual heartwood mass
  real(r8), pointer :: crownarea(:) ! area that each individual tree takes
  real(r8), pointer :: sla(:)      ! ecophys const - specific leaf area [m
  logical , pointer :: tree(:)     ! ecophys const - whether this pft is a
local pointers to implicit inout arguments
  real(r8), pointer :: fpcgrid(:)  ! foliar projective cover on gridcell (
  real(r8), pointer :: nind(:)     ! number of individuals
  real(r8), pointer :: litterag(:) ! above ground litter
  real(r8), pointer :: litterbg(:) ! below ground litter
  real(r8), pointer :: lm_ind(:)   ! individual leaf mass
  real(r8), pointer :: rm_ind(:)   ! individual root mass

```

A.13 Module DGVMMod

(Source File: DGVMMod.F90)

Module containing routines to drives the annual portion of lpj (called once per year), reset variables related to lpj, and initialize/Reset time invariant dgvm variables

USES:

```
use shr_kind_mod    , only : r8 => shr_kind_r8
use clm_varpar      , only : maxpatch_pft, lsmlon, lsmlat, nlevsoi
use abortutils      , only : endrun
```

PUBLIC TYPES:

```
implicit none
private
save
```

PUBLIC MEMBER FUNCTIONS:

```
public lpj                ! Drives the annual portion of lpj, called once
                           ! per year
public lpjreset1          ! Resets variables related to lpj
public lpjreset2          ! Resets variables related to lpj
public resetTimeConstDGVM ! Initialize/Reset time invariant dgvm variables
public resetWeightsDGVM   ! Reset DGVM subgrid weights and areas
public gatherWeightsDGVM  ! Gather DGVM subgrid weights to masterproc
public histDGVM           ! Output DGVM history file
```

REVISION HISTORY:

```
Module created by Mariana Vertenstein
```

A.13.1 lpj

INTERFACE:

```
subroutine lpj(lbg, ubg, lbp, ubp, num_natvegp, filter_natvegp, kyr)
```

DESCRIPTION:

Drives the annual portion of lpj, called once per year

USES:

```
use clmtype
use DGVMReproductionMod , only : Reproduction
use DGVMTurnoverMod     , only : Turnover
use DGVMAllocationMod   , only : Allocation
use DGVMLightMod        , only : Light
use DGVM MortalityMod    , only : Mortality
use DGVMFireMod         , only : Fire
use DGVMEstablishmentMod, only : Establishment
use DGVMKillMod         , only : Kill
```


ARGUMENTS:

```

implicit none
integer, intent(in) :: lbg, ubg      ! gridcell bounds
integer, intent(in) :: lbp, ubp      ! pft bounds
integer, intent(inout) :: num_natvegp ! number of naturally-vegetated
                                     ! pfts in filter
integer, intent(inout) :: filter_natvegp(ubp-lbp+1) ! filter for
                                     ! naturally-vegetated pfts
integer, intent(in) :: kyr           ! used in routine climate20 below

```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: mxy(:)          ! pft m index (for laixy(i,j,m),etc.)
integer , pointer :: pgridcell(:)    ! gridcell of corresponding pft
real(r8), pointer :: fpcgrid(:)      ! foliar projective cover on gridcell
real(r8), pointer :: agdd(:)         ! accumulated growing degree days above
real(r8), pointer :: t_mo_min(:)     ! annual min of t_mo (Kelvin)
local pointers to implicit inout arguments
real(r8), pointer :: tmomin20(:)     ! 20-yr running mean of tmomin
real(r8), pointer :: agdd20(:)       ! 20-yr running mean of agdd
real(r8), pointer :: bm_inc(:)        ! biomass increment
real(r8), pointer :: afmicr(:)       ! annual microbial respiration
real(r8), pointer :: afirefrac_gcell(:) ! fraction of gridcell affected by
real(r8), pointer :: acfluxfire_gcell(:) ! gridcell C flux to atmosphere
real(r8), pointer :: bmfm_gcell(:, :) ! gridcell biomass
real(r8), pointer :: afmicr_gcell(:, :) ! gridcell microbial respiration

```

A.13.2 *lpjreset1*

INTERFACE:

```

subroutine lpjreset1(lbg, ubg, lbc, ubc, lbp, ubp, &
                    num_nolakep, filter_nolakep, &
                    caldayp1, eccen, obliqr, lambm0, mvelpp)

```

DESCRIPTION:

Resets variables related to *lpj*!

USES:

```

use clmtype
use SurfaceAlbedoMod, only : SurfaceAlbedo
use DGVMEcosystemDynMod, only : DGVMEcosystemDyn

```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbg, ubg      ! gridcell bounds
integer , intent(in) :: lbc, ubc      ! column bounds
integer , intent(in) :: lbp, ubp      ! pft bounds
integer , intent(in) :: num_nolakep    ! number of non-lake pfts in filte
integer , intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-la
real(r8), intent(in) :: caldayp1 !calendar day at Greenwich (1.00, ..., 3
real(r8), intent(in) :: eccen        !Earth's orbital eccentricity
real(r8), intent(in) :: obliqr       !Earth's obliquity in radians
real(r8), intent(in) :: lambm0       !Mean longitude of perihelion at the vern
real(r8), intent(in) :: mvelpp       !Earth's moving vernal equinox long. of

```

CALLED FROM:

```

subroutine driver() in driver.F90

```

REVISION HISTORY:

Author: Sam Levis

A.13.3 lpjreset2

INTERFACE:

```

subroutine lpjreset2(lbg, ubg, lbl, ubl, lbc, ubc, lbp, ubp)

```

DESCRIPTION:

Resets variables related to lpj

USES:

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbg, ubg      ! gridcell bounds
integer, intent(in) :: lbl, ubl      ! landunit bounds
integer, intent(in) :: lbc, ubc      ! column bounds
integer, intent(in) :: lbp, ubp      ! pft bounds

```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis

A.13.4 histDGVM

INTERFACE:

```
subroutine histDGVM()
```

DESCRIPTION:

Create DGVM history file

USES:

```
use clmtype
use ncdio
use decompMod , only : get_proc_bounds, get_proc_global
use clm_varpar , only : lsmlon, lsmlat, maxpatch_pft
use clm_varsur , only : fullgrid, offline_rdgrid, landmask, &
    longxy, latixy, lsmedge, numlon
use clm_varctl , only : caseid, ctitle, finidat, fsurdat, fpftcon, &
    frivinp_rtm, archive_dir, mss_wpass, mss_irt
use clm_varcon , only : spval
use time_manager , only : get_ref_date, get_nstep, get_curr_date, &
    get_curr_time
use fileutils , only : set_filename, putfil, get_filename
use shr_sys_mod , only : shr_sys_getenv
use spmdMod , only : masterproc
use shr_const_mod, only : SHR_CONST_CDAY
```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis

LOCAL VARIABLES:

```
local pointers to implicit in arguments
logical , pointer :: ifspecial(:) ! true=>landunit is not vegetated
integer , pointer :: pgridcell(:) ! gridcell index of corresponding
integer , pointer :: plandunit(:) ! landunit index of corresponding
integer , pointer :: ivt(:) ! pft vegetation (pft-level)
integer , pointer :: mxy(:) ! pft m index (for laixy(i,j,m),e
real(r8), pointer :: fpcgrid(:) ! foliar projective cover on grid
real(r8), pointer :: lm_ind(:) ! individual leaf mass
real(r8), pointer :: sm_ind(:) ! individual sapwood mass
real(r8), pointer :: hm_ind(:) ! individual heartwood mass
real(r8), pointer :: rm_ind(:) ! individual root mass
real(r8), pointer :: nind(:) ! number of individuals (#/m**2)
real(r8), pointer :: afirefrac_gcell(:) ! fraction of gridcell affected b
real(r8), pointer :: acfluxfire_gcell(:) ! gridcell C flux to atmosphere f
real(r8), pointer :: bmfm_gcell(:, :) ! gridcell biomass
real(r8), pointer :: afmicr_gcell(:, :) ! gridcell microbial respiration
```

A.13.5 resetTimeConstDGVM

INTERFACE:

```
subroutine resetTimeConstDGVM(lbp, ubp)
```

DESCRIPTION:

Initialize/reset time invariant DGVM variables

USES:

```
use clmtype
use pftvarcon , only : roota_par, rootb_par, noveg
use clm_varcon, only : spval
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp      ! pft bounds
```

CALLED FROM:

```
lpjreset1() in this module
initialize() in initializeMod.F90
iniTimeVar() in iniTimeVar.F90
```

REVISION HISTORY:

Author: Gordon Bonan

LOCAL VARIABLES:

```
local pointers to implicit in arguments
real(r8), pointer :: zi(:, :)      ! interface level below a "z" level (m)
integer , pointer :: ivt(:)        ! pft vegetation
integer , pointer :: pcolumn(:)    ! column of corresponding pft
real(r8), pointer :: rootfr(:, :)  ! fraction of roots in each soil layer
```

A.13.6 resetWeightsDGVM

INTERFACE:

```
subroutine resetWeightsDGVM(lbg, ubg, lbc, ubc, lbp, ubp)
```

DESCRIPTION:

Reset DGVM weights

USES:

```
use clmtype
```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbg, ubg      ! gridcell bounds
integer, intent(in) :: lbc, ubc      ! column bounds
integer, intent(in) :: lbp, ubp      ! pft bounds

```

CALLED FROM:

```

subroutine lpjreset2 in this module: as part of the DGVM calculation
subroutine restart_dgvm in module DGVMRestMod: if the restart file is read
subroutine inicrd in module inicFileMod: if the initial file is read
subroutine mkarbinit in module iniTimeVar

```

REVISION HISTORY:

Author: Gordon Bonan

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: ixy(:)          ! gridcell lon index (gridcell leve
integer , pointer :: jxy(:)          ! gridcell lat index (gridcell leve
real(r8), pointer :: garea(:)        ! total land area for this gridcell
integer , pointer :: ltype(:)        ! landunit type
real(r8), pointer :: larea(:)        ! total land area for this landunit
logical , pointer :: ifspecial(:)    ! true=>landunit is not vegetated
integer , pointer :: clandunit(:)    ! index into landunit for each colu
integer , pointer :: cgridcell(:)    ! index into gridcell for each colu
real(r8), pointer :: fpcgrid(:)      ! weight of pft relative to vegetat
real(r8), pointer :: h2ocan_pft(:)   ! canopy water (mm H2O) (pft-level)
local pointers to implicit out arguments
real(r8), pointer :: cwtgcell(:)     ! weight (relative to gridcell) for
real(r8), pointer :: cwtlunit(:)     ! weight (relative to landunit) for
real(r8), pointer :: carea(:)        ! total land area for this column (
real(r8), pointer :: h2ocan_col(:)   ! canopy water (mm H2O) (column-lev
integer , pointer :: pcolumn(:)      ! index into column for each pft
integer , pointer :: plandunit(:)    ! index into landunit for each pft
integer , pointer :: pgridcell(:)    ! index into gridcell for each pft
real(r8), pointer :: pwtcol(:)       ! weight (relative to column) for t
real(r8), pointer :: pwtlunit(:)     ! weight (relative to landunit) for
real(r8), pointer :: pwtgcell(:)     ! weight (relative to gridcell) for
real(r8), pointer :: parea(:)        ! total land area for this pft (km^

```

A.13.7 gatherWeightsDGVM

INTERFACE:

```

subroutine gatherWeightsDGVM()

```

DESCRIPTION:

Gather DGVM weights to master process. Should be called after all invocations of resetWeightsDGVM() are complete to update weights on master process if SPMD is defined.

USES:

```

    use clmtype
    use decompMod      , only : get_proc_bounds, get_proc_global
#ifdef SPMD
    use spmdGathScatMod, only : gather_data_to_master, masterproc
#endif
    use spmdMod        , only : masterproc

```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```

subroutine driver in module driver: after lpjreset2 is called for all clum
subroutine restart_dgvm in module DGVMRestMod: if the restart file is read
subroutine inicrd in module inicFileMod: if the initial file is read
subroutine mkarbinit in module iniTimeVar

```

REVISION HISTORY:

Author: Gordon Bonan

LOCAL VARIABLES:

```

local pointers to implicit inout arguments
real(r8), pointer :: cwtgcell(:)      ! weight (relative to gridcell) for
real(r8), pointer :: cwtlunit(:)     ! weight (relative to landunit) for
real(r8), pointer :: pwtcol(:)       ! weight (relative to column) for t
real(r8), pointer :: pwtlunit(:)     ! weight (relative to landunit) for
real(r8), pointer :: pwtgcell(:)     ! weight (relative to gridcell) for

```

A.13.8 set_dgvm_filename

INTERFACE:

```
character(len=256) function set_dgvm_filename ()
```

DESCRIPTION:

Determine initial dataset filenames

USES:

```

    use clm_varctl , only : caseid
    use time_manager, only : get_curr_date

```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

REVISION HISTORY:

Author: Mariana Vertenstein

A.13.9 BuildNatVegFilter

INTERFACE:

```
subroutine BuildNatVegFilter(lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Reconstruct a filter of naturally-vegetated PFTs for use in DGVM

USES:

```
use clmtype
use pftvarcon , only : crop
```

ARGUMENTS:

```
implicit none
integer, intent(in)  :: lbp, ubp           ! pft bounds
integer, intent(out) :: num_natvegp        ! number of pfts in na
integer, intent(out) :: filter_natvegp(ubp-lbp+1) ! pft filter for natur
```

CALLED FROM:

```
subroutine lpj in this module
```

REVISION HISTORY:

Author: Forrest Hoffman

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: ivt(:)           ! pft vegetation (pft level)
logical , pointer :: present(:)       ! whether this pft present in patch
```

A.14 Module MortalityMod

(Source File: DGVMMortalityMod.F90)

Tree background and stress mortality
Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Mortality
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.14.1 Mortality

INTERFACE:

```
subroutine Mortality(lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Tree background and stress mortality

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natvegp       ! number of naturally-ve
integer, intent(in) :: filter_natvegp(ubp-lbp+1) ! pft filter for natural
```

CALLED FROM:

```
subroutine lpj in module DGVMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. mortality)

LOCAL VARIABLES:


```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)      ! pft vegetation type
  real(r8), pointer :: bm_inc(:)   ! biomass increment
  real(r8), pointer :: lm_ind(:)   ! individual leaf mass
  real(r8), pointer :: sm_ind(:)   ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)   ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)   ! individual root mass
  real(r8), pointer :: agddtw(:)   ! accumulated growing degree days ab
  real(r8), pointer :: turnover_ind(:) !
  logical , pointer :: tree(:)     ! ecophys const - whether this pft i
  real(r8), pointer :: sla(:)      ! ecophys const - specific leaf area
local pointers to implicit inout arguments
  logical , pointer :: present(:)  ! whether PFT present in patch
  real(r8), pointer :: nind(:)     ! number of individuals
  real(r8), pointer :: litterag(:) ! above ground litter
  real(r8), pointer :: litterbg(:) ! below ground litter

```

A.15 Module ReproductionMod

(Source File: DGVMReproductionMod.F90)

Deduction of reproduction costs from annual biomass increment
Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Reproduction
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.15.1 Reproduction

INTERFACE:

```
subroutine Reproduction(lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natvegp       ! number of naturally-ve
integer, intent(in) :: filter_natvegp(ubp-lbp+1) ! pft filter for natural
```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. reproduction)

LOCAL VARIABLES:

```
local pointers to implicit inout arguments
real(r8), pointer :: litter_ag(:)        ! above ground litter
real(r8), pointer :: bm_inc(:)          ! biomass increment
```


A.16 Module restDGVMMod (Source File: DGVMRestMod.F90)

Read/Write to/from DGVM info to CLM restart file.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use spmdMod      , only : masterproc
use abortutils   , only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: restart_dgvm
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.16.1 restart_dgvm

INTERFACE:

```
subroutine restart_dgvm (nio, flag)
```

DESCRIPTION:

Read/write DGVM restart data

USES:

```
use clmtype
use iobinary
use decompMod, only : get_proc_bounds, get_proc_global
use DGVMMod   , only : resetWeightsDGVM, gatherWeightsDGVM
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: nio          ! restart unit
character(len=*), intent(in) :: flag ! 'read' or 'write'
```

CALLED FROM:

```
subroutine restart in module restFileMod
```

REVISION HISTORY:

Author: Mariana Vertenstein

A.17 Module TurnoverMod

(Source File: DGVMTurnoverMod.F90)

Turnover of PFT-specific fraction from each living C pool
 Leaf and root C transferred to litter, sapwood C to heartwood
 Called once per year

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Turnover
```

REVISION HISTORY:

Module created by Mariana Vertenstein

A.17.1 Turnover

INTERFACE:

```
subroutine Turnover(lbp, ubp, num_natvegp, filter_natvegp)
```

DESCRIPTION:

Turnover of PFT-specific fraction from each living C pool
 Leaf and root C transferred to litter, sapwood C to heartwood
 Called once per year

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_natvegp       ! number of naturally-ve
integer, intent(in) :: filter_natvegp(ubp-lbp+1) ! pft filter for natural
```

CALLED FROM:

```
subroutine lpj in module DGVMMod
```

REVISION HISTORY:

Author: Sam Levis (adapted from Stephen Sitch's LPJ subr. turnover)

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: ivt(:)           ! pft vegetation type
  real(r8), pointer :: nind(:)          ! number of individuals (#/m**2)
  real(r8), pointer :: l_turn(:)        ! ecophys const - leaf turnover p
  real(r8), pointer :: s_turn(:)        ! ecophys const - sapwood turnove
  real(r8), pointer :: r_turn(:)        ! ecophys const - root turnover p
local pointers to implicit inout arguments
  real(r8), pointer :: litter_ag(:)      ! above ground litter
  real(r8), pointer :: litter_bg(:)      ! below ground litter
  real(r8), pointer :: lm_ind(:)         ! individual leaf mass
  real(r8), pointer :: sm_ind(:)         ! individual sapwood mass
  real(r8), pointer :: hm_ind(:)         ! individual heartwood mass
  real(r8), pointer :: rm_ind(:)         ! individual root mass
local pointers to implicit out arguments
  real(r8), pointer :: turnover_ind(:)   !

```

A.18 Module DriverInitMod

(Source File: DriverInitMod.F90)

Initialization of driver variables needed from previous timestep

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: DriverInit
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.18.1 DriverInit

INTERFACE:

```
subroutine DriverInit(lbc, ubc, lbp, ubp, &
                     num_nolakec, filter_nolakec, num_lakec, filter_lakec)
```

DESCRIPTION:

Initialization of driver variables needed from previous timestep

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varpar, only : nlevsoi
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbc, ubc           ! column-index bounds
integer, intent(in) :: lbp, ubp           ! pft-index bounds
integer, intent(in) :: num_nolakec        ! number of column non
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for no
integer, intent(in) :: num_lakec          ! number of column non
integer, intent(in) :: filter_lakec(ubc-lbc+1) ! column filter for no
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.19 Module FracWetMod

(Source File: FracWetMod.F90)

Determine fraction of vegetated surfaces which are wet and fraction of elai which is dry.

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: FracWet
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.19.1 FracWet

INTERFACE:

```
subroutine FracWet(numf, filter)
```

DESCRIPTION:

Determine fraction of vegetated surfaces which are wet and fraction of elai which is dry. The variable “fwet” is the fraction of all vegetation surfaces which are wet including stem area which contribute to evaporation. The variable “fdry” is the fraction of elai which is dry because only leaves can transpire. Adjusted for stem area which does not transpire.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: numf           ! number of filter non-lake
integer, intent(in) :: filter(numf)  ! pft filter for non-lake po
```

CALLED FROM:

```
subroutine Hydrology1 in module Hydrology1Mod
```

REVISION HISTORY:

Created by Keith Oleson and M. Vertenstein
03/08/29 Mariana Vertenstein : Migrated to vectorized code

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: frac_veg_nosno(:) ! fraction of veg not covered by s
real(r8), pointer :: dewmx(:)         ! Maximum allowed dew [mm]
real(r8), pointer :: elai(:)          ! one-sided leaf area index with b
real(r8), pointer :: esai(:)          ! one-sided stem area index with b
real(r8), pointer :: h2ocan(:)        ! total canopy water (mm H2O)
local pointers to implicit out arguments
real(r8), pointer :: fwet(:)          ! fraction of canopy that is wet (
real(r8), pointer :: fdry(:)         ! fraction of foliage that is gree
```


A.20 Module FrictionVelocityMod (Source File: FrictionVelocityMod.F90)

Calculation of the friction velocity, relation for potential temperature and humidity profiles of surface boundary layer.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: FrictionVelocity      ! Calculate friction velocity
public :: MoninObukIni         ! Initialization of the Monin-Obukhov leng
!PRIVATE MEMBER FUNCTIONS:
private :: StabilityFunc1      ! Stability function for rib < 0.
private :: StabilityFunc2      ! Stability function for rib < 0.
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.20.1 FrictionVelocity

INTERFACE:

```
subroutine FrictionVelocity(lbp, ubp, fn, filterp, &
                           displa, z0m, z0h, z0q, &
                           obu, iter, ur, um, ustar, &
                           temp1, temp2, temp12m, temp22m, fm)
```

DESCRIPTION:

Calculation of the friction velocity, relation for potential temperature and humidity profiles of surface boundary layer. The scheme is based on the work of Zeng et al. (1998): Intercomparison of bulk aerodynamic algorithms for the computation of sea surface fluxes using TOGA CORE and TAO data. J. Climate, Vol. 11, 2628-2644.

USES:

```
use clmtype
use clm_varcon, only : vkc
```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbp, ubp      ! pft array bounds
integer , intent(in) :: fn           ! number of filtered pft element
integer , intent(in) :: filterp(fn)  ! pft filter
real(r8), intent(in) :: displa(lbp:ubp) ! displacement height (m)
real(r8), intent(in) :: z0m(lbp:ubp) ! roughness length over vegetati
real(r8), intent(in) :: z0h(lbp:ubp) ! roughness length over vegetati
real(r8), intent(in) :: z0q(lbp:ubp) ! roughness length over vegetati
real(r8), intent(in) :: obu(lbp:ubp) ! monin-obukhov length (m)
integer , intent(in) :: iter         ! iteration number
real(r8), intent(in) :: ur(lbp:ubp) ! wind speed at reference height
real(r8), intent(in) :: um(lbp:ubp) ! wind speed including the stabl
real(r8), intent(out) :: ustar(lbp:ubp) ! friction velocity [m/s]
real(r8), intent(out) :: temp1(lbp:ubp) ! relation for potential tempera
real(r8), intent(out) :: temp12m(lbp:ubp) ! relation for potential tempera
real(r8), intent(out) :: temp2(lbp:ubp) ! relation for specific humidity
real(r8), intent(out) :: temp22m(lbp:ubp) ! relation for specific humidity
real(r8), intent(inout) :: fm(lbp:ubp) ! needed for DGVM only to diagno

```

CALLED FROM:

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 12/19/01, Peter Thornton
 Added arguments to eliminate passing clm derived type into this function.
 Created by Mariana Vertenstein

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: pgridcell(:) ! pft's gridcell index
real(r8), pointer :: forc_hgt(:) ! atmospheric reference height (m)
real(r8), pointer :: forc_hgt_u(:) ! observational height of wind [m]
real(r8), pointer :: forc_hgt_t(:) ! observational height of temperature
real(r8), pointer :: forc_hgt_q(:) ! observational height of humidity [m]
local pointers to implicit out arguments
real(r8), pointer :: u10(:) ! 10-m wind (m/s) (for dust model)
real(r8), pointer :: fv(:) ! friction velocity (m/s) (for dust mo

```

A.20.2 StabilityFunc

INTERFACE:

```
real(r8) function StabilityFunc1(zeta)
```

DESCRIPTION:

Stability function for $\text{rib} < 0$.

USES:

```
use shr_const_mod, only: SHR_CONST_PI
```

ARGUMENTS:

```
implicit none
real(r8), intent(in) :: zeta ! dimensionless height used in Monin-Obuk
```

CALLED FROM:

```
subroutine FrictionVelocity in this module
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
```

A.20.3 StabilityFunc2

INTERFACE:

```
real(r8) function StabilityFunc2(zeta)
```

DESCRIPTION:

Stability function for $\text{rib} < 0$.

USES:

```
use shr_const_mod, only: SHR_CONST_PI
```

ARGUMENTS:

```
implicit none
real(r8), intent(in) :: zeta ! dimensionless height used in Monin-Obukh
```

CALLED FROM:

```
subroutine FrictionVelocity in this module
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
```

A.20.4 MoninObukIni

INTERFACE:

```
subroutine MoninObukIni (ur, thv, dthv, zldis, zOm, um, obu)
```

DESCRIPTION:

Initialization of the Monin-Obukhov length.
 The scheme is based on the work of Zeng et al. (1998):
 Intercomparison of bulk aerodynamic algorithms for the computation
 of sea surface fluxes using TOGA CORE and TAO data. J. Climate,
 Vol. 11, 2628-2644.

USES:

```
use clm_varcon, only : grav
```

ARGUMENTS:

```
implicit none
real(r8), intent(in)  :: ur      ! wind speed at reference height [m/s]
real(r8), intent(in)  :: thv     ! virtual potential temperature (kelvin)
real(r8), intent(in)  :: dthv    ! diff of vir. poten. temp. between ref. h
real(r8), intent(in)  :: zldis   ! reference height "minus" zero displaceme
real(r8), intent(in)  :: z0m     ! roughness length, momentum [m]
real(r8), intent(out) :: um      ! wind speed including the stability effec
real(r8), intent(out) :: obu     ! monin-obukhov length (m)
```

CALLED FROM:

```
subroutine BareGroundFluxes in module BareGroundFluxesMod.F90
subroutine BiogeophysicsLake in module BiogeophysicsLakeMod.F90
subroutine CanopyFluxes in module CanopyFluxesMod.F90
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
```

A.21 Module Hydrology1Mod

(Source File: Hydrology1Mod.F90)

Calculation of

- (1) water storage of intercepted precipitation
- (2) direct throughfall and canopy drainage of precipitation
- (3) the fraction of foliage covered by water and the fraction of foliage that is dry and transpiring.
- (4) snow layer initialization if the snow accumulation exceeds 10 mm.

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Hydrology1
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.21.1 Hydrology1

INTERFACE:

```
subroutine Hydrology1(lbc, ubc, lbp, ubp, num_nolakec, filter_nolakec, &
                     num_nolakep, filter_nolakep)
```

DESCRIPTION:

Calculation of

- (1) water storage of intercepted precipitation
- (2) direct throughfall and canopy drainage of precipitation
- (3) the fraction of foliage covered by water and the fraction of foliage that is dry and transpiring.
- (4) snow layer initialization if the snow accumulation exceeds 10 mm.

Note: The evaporation loss is taken off after the calculation of leaf temperature in the subroutine `clm_leafem.f90`, not in this subroutine.

USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
use clmtype
use clm_varcon , only : tfrz, istice, istwet, istsoil
use FracWetMod , only : FracWet
use time_manager , only : get_step_size
use subgridAveMod, only : p2c
```

ARGUMENTS:


```

implicit none
integer, intent(in) :: lbp, ubp                ! pft bounds
integer, intent(in) :: lbc, ubc                ! column bounds
integer, intent(in) :: num_nolakec            ! number of column no
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for n
integer, intent(in) :: num_nolakep            ! number of pft non-l
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-

```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
2/15/02, Peter Thornton: Migrated to new data structures. Required
adding a PFT loop.

```

LOCAL VARIABLES:

```

local pointers to original implicit in arrays
integer , pointer :: cgridcell(:)             ! columns's gridcell
integer , pointer :: clandunit(:)             ! columns's landunit
integer , pointer :: pgridcell(:)             ! pft's gridcell
integer , pointer :: plandunit(:)             ! pft's landunit
integer , pointer :: pcolumn(:)               ! pft's column
integer , pointer :: npfts(:)                 ! number of pfts in column
integer , pointer :: pfti(:)                 ! column's beginning pft index
integer , pointer :: itype(:)                 ! landunit type
real(r8), pointer :: forc_rain(:)             ! rain rate [mm/s]
real(r8), pointer :: forc_snow(:)             ! snow rate [mm/s]
real(r8), pointer :: forc_t(:)               ! atmospheric temperature (Kelvin)
#if defined OFFLINE
real(r8), pointer :: flfall(:)               ! fraction of liquid water within
#endif
logical , pointer :: do_capsnow(:)           ! true => do snow capping
real(r8), pointer :: t_grnd(:)               ! ground temperature (Kelvin)
real(r8), pointer :: dewmx(:)               ! Maximum allowed dew [mm]
integer , pointer :: frac_veg_nosno(:)       ! fraction of veg not covered by s
real(r8), pointer :: elai(:)                ! one-sided leaf area index with b
real(r8), pointer :: esai(:)                ! one-sided stem area index with b
local pointers to original implicit inout arrays
integer , pointer :: snl(:)                  ! number of snow layers
real(r8), pointer :: snowage(:)              ! non dimensional snow age [-]
real(r8), pointer :: snowdp(:)              ! snow height (m)
real(r8), pointer :: h2osno(:)              ! snow water (mm H2O)
real(r8), pointer :: h2ocan(:)              ! total canopy water (mm H2O)
local pointers to original implicit out arrays
real(r8), pointer :: qflx_prec_intr(:)       ! interception of precipitation
real(r8), pointer :: qflx_prec_grnd(:)       ! water onto ground including
real(r8), pointer :: qflx_snowcap(:)         ! excess precipitation due to
real(r8), pointer :: qflx_snow_grnd_pft(:)   ! snow on ground after intercep
real(r8), pointer :: qflx_snow_grnd_col(:)   ! snow on ground after intercep
real(r8), pointer :: qflx_rain_grnd(:)       ! rain on ground after interce
real(r8), pointer :: fwet(:)                 ! fraction of canopy that is w

```

```
real(r8), pointer :: fdry(:)           ! fraction of foliage that is
real(r8), pointer :: zi(:, :)          ! interface level below a "z"
real(r8), pointer :: dz(:, :)          ! layer depth (m)
real(r8), pointer :: z(:, :)           ! layer thickness (m)
real(r8), pointer :: t_soisno(:, :)    ! soil temperature (Kelvin)
real(r8), pointer :: h2osoi_ice(:, :)  ! ice lens (kg/m2)
real(r8), pointer :: h2osoi_liq(:, :)  ! liquid water (kg/m2)
real(r8), pointer :: frac_iceold(:, :) ! fraction of ice relative to
```


A.22 Module Hydrology2Mod (Source File: Hydrology2Mod.F90)

Calculation of soil/snow hydrology.

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Hydrology2      ! Calcultes soil/snow hydrology
```

REVISION HISTORY:

```
2/28/02 Peter Thornton: Migrated to new data structures.
7/12/03 Forrest Hoffman ,Mariana Vertenstein : Migrated to vector code
```

A.22.1 Hydrology2

INTERFACE:

```
subroutine Hydrology2(lbc, ubc, num_nolakec, filter_nolakec, &
                     num_soilc, filter_soilc, num_snowc, filter_snowc, &
                     num_nosnowc, filter_nosnowc)
```

DESCRIPTION:

This is the main subroutine to execute the calculation of soil/snow hydrology

Calling sequence is:

Hydrology2:	surface hydrology driver
-> SnowWater:	change of snow mass and snow water onto soil
-> SurfaceRunoff:	surface runoff
-> Infiltration:	infiltration into surface soil layer
-> SoilWater:	soil water movement between layers
-> Tridiagonal	tridiagonal matrix solution
-> Drainage:	subsurface runoff
-> SnowCompaction:	compaction of snow layers
-> CombineSnowLayers:	combine snow layers that are thinner than mini
-> DivideSnowLayers:	subdivide snow layers that are thicker than ma

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon      , only : denh2o, denice, istice, istwet, istsoil, spv
use clm_varpar      , only : nlevsoi, nlevsno
use SnowHydrologyMod, only : SnowCompaction, CombineSnowLayers, DivideSno
                             SnowWater, BuildSnowFilter
use SoilHydrologyMod, only : Infiltration, SoilWater, Drainage, SurfaceRu
#if (defined COUP_CAM)
```

```

    use time_manager      , only : get_step_size, get_nstep, is_perpetual
#else
    use time_manager      , only : get_step_size, get_nstep
#endif

```

ARGUMENTS:

```

    implicit none
    integer, intent(in) :: lbc, ubc                ! column bounds
    integer, intent(in) :: num_nolakec             ! number of column non
    integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for no
    integer, intent(in) :: num_soilc               ! number of column soi
    integer, intent(in) :: filter_soilc(ubc-lbc+1) ! column filter for so

```

CALLED FROM:

```

    subroutine driver

```

REVISION HISTORY:

Created by Mariana Vertenstein

LOCAL VARIABLES:

```

    local pointers to implicit in arguments
    integer , pointer :: cgridcell(:) ! column's gridcell
    integer , pointer :: clandunit(:) ! column's landunit
    integer , pointer :: ityplun(:)   ! landunit type
    integer , pointer :: snl(:)       ! number of snow layers
    real(r8), pointer :: h2ocan(:)    ! canopy water (mm H2O)
    real(r8), pointer :: h2osno(:)    ! snow water (mm H2O)
    real(r8), pointer :: watsat(:, :) ! volumetric soil water at saturati
    real(r8), pointer :: sucsat(:, :) ! minimum soil suction (mm)
    real(r8), pointer :: bsw(:, :)    ! Clapp and Hornberger "b"
    real(r8), pointer :: z(:, :)      ! layer depth (m)
    real(r8), pointer :: forc_rain(:) ! rain rate [mm/s]
    real(r8), pointer :: forc_snow(:) ! snow rate [mm/s]
    real(r8), pointer :: begwb(:)     ! water mass begining of the time s
    real(r8), pointer :: qflx_evap_tot(:) ! qflx_evap_soi + qflx_evap_veg + q

    local pointers to implicit inout arguments
    real(r8), pointer :: dz(:, :) ! layer thickness depth (m)
    real(r8), pointer :: zi(:, :) ! interface depth (m)

    local pointers to implicit out arguments
    real(r8), pointer :: endwb(:) ! water mass end of the time step
    real(r8), pointer :: snowage(:) ! non dimensional snow age [-]
    real(r8), pointer :: wf(:) ! soil water as frac. of whc for to
    real(r8), pointer :: snowice(:) ! average snow ice lens
    real(r8), pointer :: snowliq(:) ! average snow liquid water
    real(r8), pointer :: t_snow(:) ! vertically averaged snow temperat
    real(r8), pointer :: t_grnd(:) ! ground temperature (Kelvin)
    real(r8), pointer :: t_soisno(:, :) ! soil temperature (Kelvin)
    real(r8), pointer :: h2osoi_ice(:, :) ! ice lens (kg/m2)
    real(r8), pointer :: h2osoi_liq(:, :) ! liquid water (kg/m2)
    real(r8), pointer :: h2osoi_vol(:, :) ! volumetric soil water (0<=h2osoi_
    real(r8), pointer :: qflx_drain(:) ! sub-surface runoff (mm H2O /s)
    real(r8), pointer :: qflx_surf(:) ! surface runoff (mm H2O /s)
    real(r8), pointer :: qflx_infl(:) ! infiltration (mm H2O /s)
    real(r8), pointer :: qflx_qrgwl(:) ! qflx_surf at glaciers, wetlands,

```

A.23 Module HydrologyLakeMod

(Source File: HydrologyLakeMod.F90)

Calculate lake hydrology

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: HydrologyLake
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.23.1 HydrologyLake

INTERFACE:

```
subroutine HydrologyLake(lbp, ubp, num_lakep, filter_lakep)
```

DESCRIPTION:

Calculate lake hydrology

WARNING: This subroutine assumes lake columns have one and only one pft.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use time_manager, only : get_step_size
use clm_varcon , only : hfus, tfrz, spval
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft-index bounds
integer, intent(in) :: num_lakep         ! number of pft non-lake p
integer, intent(in) :: filter_lakep(ubp-lbp+1) ! pft filter for non-lake
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

Author: Gordon Bonan
 15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 3/4/02: Peter Thornton; Migrated to new data structures.

LOCAL VARIABLES:

```

local pointers to implicit in arrays
  integer , pointer :: pcolumn(:)      !pft's column index
  integer , pointer :: pgridcell(:)    !pft's gridcell index
  real(r8), pointer :: begwb(:)        !water mass begining of the time st
  real(r8), pointer :: forc_snow(:)    !snow rate [mm/s]
  real(r8), pointer :: forc_rain(:)    !rain rate [mm/s]
  logical , pointer :: do_capsnow(:)   !true => do snow capping
  real(r8), pointer :: t_grnd(:)       !ground temperature (Kelvin)
  real(r8), pointer :: qmelt(:)        !snow melt [mm/s]
  real(r8), pointer :: qflx_evap_soi(:) !soil evaporation (mm H2O/s) (+ = t
  real(r8), pointer :: qflx_evap_tot(:) !qflx_evap_soi + qflx_evap_veg + qf
local pointers to implicit inout arrays
  real(r8), pointer :: h2osno(:)       !snow water (mm H2O)
local pointers to implicit out arrays
  real(r8), pointer :: endwb(:)        !water mass end of the time step
  real(r8), pointer :: snowdp(:)       !snow height (m)
  real(r8), pointer :: snowice(:)      !average snow ice lens
  real(r8), pointer :: snowliq(:)      !average snow liquid water
  real(r8), pointer :: eflx_snomelt(:) !snow melt heat flux (W/m**2)
  real(r8), pointer :: qflx_infl(:)    !infiltration (mm H2O /s)
  real(r8), pointer :: qflx_snomelt(:) !snow melt (mm H2O /s)
  real(r8), pointer :: qflx_surf(:)    !surface runoff (mm H2O /s)
  real(r8), pointer :: qflx_drain(:)   !sub-surface runoff (mm H2O /s)
  real(r8), pointer :: qflx_qrgwl(:)   !qflx_surf at glaciers, wetlands, l
  real(r8), pointer :: qflx_evap_tot_col(:) !pft quantity averaged to the c
local pointers to implicit out multi-level arrays
  real(r8), pointer :: rootr_column(:, :) !effective fraction of roots in ea
  real(r8), pointer :: h2osoi_vol(:, :)   !volumetric soil water (0<=h2osoi_
  real(r8), pointer :: h2osoi_ice(:, :)   !ice lens (kg/m2)
  real(r8), pointer :: h2osoi_liq(:, :)   !liquid water (kg/m2)

```

A.24 Module QSatMod

(Source File: QSatMod.F90)

Computes saturation mixing ratio and the change in saturation

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: QSat
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.24.1 QSat

INTERFACE:

```
subroutine QSat (T, p, es, esdT, qs, qsdT)
```

DESCRIPTION:

Computes saturation mixing ratio and the change in saturation mixing ratio with respect to temperature.

Reference: Polynomial approximations from:

Piotr J. Flatau, et al., 1992: Polynomial fits to saturation vapor pressure. *Journal of Applied Meteorology*, 31, 1507-1513.

USES:

```
use shr_kind_mod , only: r8 => shr_kind_r8
use shr_const_mod, only: SHR_CONST_TKFRZ
```

ARGUMENTS:

```
implicit none
real(r8), intent(in)  :: T           ! temperature (K)
real(r8), intent(in)  :: p           ! surface atmospheric pressure (pa)
real(r8), intent(out) :: es          ! vapor pressure (pa)
real(r8), intent(out) :: esdT        ! d(es)/d(T)
real(r8), intent(out) :: qs          ! humidity (kg/kg)
real(r8), intent(out) :: qsdT        ! d(qs)/d(T)
```

CALLED FROM:

```
subroutine Biogeophysics1 in module Biogeophysics1Mod
subroutine BiogeophysicsLake in module BiogeophysicsLakeMod
subroutine CanopyFluxesMod CanopyFluxesMod
```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision

A.25 Module RtmMod

(Source File: RtmMod.F90)

River Routing Model (U. of Texas River Transport Model) ([Branstetter, 2001](#))

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clm_varpar , only : lsmlon, lsmlat, rtmlon, rtmlat
use shr_sys_mod , only : shr_sys_flush
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
integer , parameter, public :: rtmloni = 1      ! RTM grid - per-proc beg
integer , parameter, public :: rtmlonf = rtmlon ! RTM grid - per-proc end
integer , parameter, public :: rtmlati = 1      ! RTM grid - per-proc beg
integer , parameter, public :: rtmlatf = rtmlat ! RTM grid - per-proc end
real(r8), public, pointer  :: latixy_r(:, :)    ! RTM grid - latitudes o
real(r8), public, pointer  :: longxy_r(:, :)    ! RTM grid - longitudes o
real(r8), public, pointer  :: area_r(:, :)      ! RTM grid - gridcell are
integer , public, pointer  :: mask_r(:, :)      ! RTM grid - landmask (la
real(r8), allocatable, public :: volr(:, :)     ! RTM fluxes - water volu
```

PUBLIC MEMBER FUNCTIONS:

```
public Rtmgridini    ! Initialize RTM grid and land mask
public Rtmlandini    ! Initialize RTM-land interpolation weights
public Rtmfluxini    ! Initialize RTM fluxout
public Rtmriverflux  ! Interface with RTM river routing model
public restart_rtm   ! Read/write RTM restart data
```

REVISION HISTORY:

Author: Sam Levis

A.25.1 Rtmgridini

INTERFACE:

```
subroutine Rtmgridini
```

DESCRIPTION:

Initialize RTM grid and land mask.

USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
#if (defined SPMD)
use spmdMod      , only : mpicom, MPI_REAL8, MPI_INTEGER, masterproc
#else
```

```

        use spmdMod      , only : masterproc
    #endif
        use areaMod      , only : celledge, cellarea
        use clm_varctl   , only : frivinp_rtm
        use clm_varcon   , only : re
        use shr_const_mod, only : SHR_CONST_PI

```

ARGUMENTS:

```

    implicit none

```

CALLED FROM:

```

    subroutine initialize in module initializeMod

```

REVISION HISTORY:

```

    Author: Sam Levis

```

A.25.2 Rtmlandini

INTERFACE:

```

    subroutine Rtmlandini

```

DESCRIPTION:

Initialize RTM-land interpolation weights and variables related to runoff time averaging.

USES:

```

    use shr_kind_mod, only : r8 => shr_kind_r8
    use clmtype
    use decompMod   , only : get_proc_bounds, get_proc_global
    use clm_varsur  , only : numlon, area, lats, lonw, landmask
    use spmdMod     , only : masterproc
    use areaMod     , only : areaini_point, mkmxovr
    use time_manager, only : get_curr_date
    use RunoffMod   , only : set_proc_rof_bounds, set_roflnd, set_rofocn

```

ARGUMENTS:

```

    implicit none

```

CALLED FROM:

REVISION HISTORY:

```

    Author: Mariana Vertenstein

```

LOCAL VARIABLES:

```

    local pointers to implicit in arguments
    integer , pointer :: ixy(:)           ! gricell xy lon index
    integer , pointer :: jxy(:)           ! gricell xy lat index

```

A.25.3 Rtmfluxini()

INTERFACE:

```
subroutine Rtmfluxini()
```

DESCRIPTION:

Initialize RTM fluxout for case of initial run when initial data is read in. For restart run, RTM fluxout is read from restart dataset.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use time_manager, only : get_step_size
use clm_varctl , only : rtm_nsteps
```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

REVISION HISTORY:

Author: Mariana Vertenstein

A.25.4 Rtmriverflux

INTERFACE:

```
subroutine Rtmriverflux()
```

DESCRIPTION:

Interface with RTM river routing model.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use spmdMod      , only : masterproc
use clm_varpar   , only : lsmlon, lsmlat
use clm_varsur   , only : landfrac
use RunoffMod    , only : UpdateRunoff
```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

Author: Sam Levis

A.25.5 UpdateInput

INTERFACE:

```
subroutine UpdateInput(do_rtm, precxy, evapxy, totruninxy)
```

DESCRIPTION:

Update RTM inputs.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use decompMod      , only : get_proc_bounds, get_proc_global
use clm_varpar     , only : lsmlon, lsmlat
use clm_varsur     , only : area
use clm_varctl     , only : rtm_nsteps
use time_manager   , only : get_step_size, get_nstep
#if (defined SPMD)
use spmdMod        , only : masterproc, mpicom
use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_mast
#else
use spmdMod        , only : masterproc
#endif
#endif
```

ARGUMENTS:

```
implicit none
logical , intent(out) :: do_rtm
real(r8), intent(out) :: precxy(lsmlon,lsmlat) ! precipitation (mm H2
real(r8), intent(out) :: evapxy(lsmlon,lsmlat) ! evaporation (mm H2O
real(r8), intent(out) :: totruninxy(lsmlon,lsmlat) ! surface runoff (mm H
```

CALLED FROM:

```
subroutine driver
```

REVISION HISTORY:

Author: Sam Levis

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: ixy(:) ! gricell xy lon index
integer , pointer :: jxy(:) ! gricell xy lat index
integer , pointer :: cgridcell(:) ! corresponding gridcell index
real(r8), pointer :: wtgcell(:) ! weight (relative to gridcell)
real(r8), pointer :: forc_rain(:) ! rain rate [mm/s]
real(r8), pointer :: forc_snow(:) ! snow rate [mm/s]
real(r8), pointer :: qflx_qrgwl(:) ! qflx_surf at glaciers, wetlan
real(r8), pointer :: qflx_drain(:) ! sub-surface runoff (mm H2O /s
real(r8), pointer :: qflx_evap_tot(:) ! qflx_evap_soi + qflx_evap_veg
real(r8), pointer :: qflx_surf(:) ! surface runoff (mm H2O /s)
```

A.25.6 Rtm

INTERFACE:

```
subroutine Rtm
```

DESCRIPTION:

River routing model (based on U. Texas code). Input is totrunin_r. Input/output is fluxout, volr. Outputs are dvolrdt_r, flxocn_r, flxlnd_r.

USES:

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine Rtmriverflux in this module
```

REVISION HISTORY:

```
Author: Sam Levis
```

A.25.7 UpdateGlobal

INTERFACE:

```
subroutine UpdateGlobal(totruninxy, precxy, evapxy)
```

DESCRIPTION:

Update Global quantities. Input is totrunin_r. Input/output is fluxout, volr. Outputs are dvolrdt_r, flxocn_r, flxlnd_r.

USES:

```
use clm_varsur , only : numlon, area, landfrac
use time_manager, only : get_nstep, get_curr_date
```

ARGUMENTS:

```
implicit none
real(r8), intent(inout) :: totruninxy(lsmlon,lsmlat) !surface runoff (mm)
real(r8), intent(inout) :: precxy(lsmlon,lsmlat)    !precipitation (mm)
real(r8), intent(inout) :: evapxy(lsmlon,lsmlat)    !evaporation (mm H2)
```

CALLED FROM:

```
subroutine Rtmriverflux in this module
```

REVISION HISTORY:

```
Author: Mariana Vertenstein
```

A.25.8 restart_rtm

INTERFACE:

```
subroutine restart_rtm (nio, flag, rest_id)
```

DESCRIPTION:

Read/write RTM restart data.

USES:

```

    use shr_kind_mod, only      : r8 => shr_kind_r8
    use clmtype                , only      : nameg
    #if (defined SPMD)
        use spmdMod            , only : masterproc, MPI_INTEGER, MPI_REAL8, mpicom
        use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_mast
    #else
        use spmdMod            , only : masterproc
    #endif
    use decompMod              , only : get_proc_global, map_sn2dc, map_dc2sn
    use runoffMod              , only : runoff

```

ARGUMENTS:

```

    implicit none
    integer, intent(in) :: nio          ! restart unit
    character(len=*), intent(in) :: flag ! 'read' or 'write'
    integer, intent(in) :: rest_id      ! restart id flag

```

CALLED FROM:

```
subroutine restart in module restFileMod
```

REVISION HISTORY:

Author: Mariana Vertenstein

A.26 Module RunoffMod

(Source File: RunoffMod.F90)

Module containing utilities for history file and coupler runoff data

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
type runoff_flow
  real(r8), pointer :: lnd(:)      ! RTM river (channel) flow (m**3 H2O /s)
  real(r8), pointer :: ocn(:)      ! RTM river flow into ocean (m**3/s)
  integer , pointer :: lnd_ixy(:)  ! RTM longitude index of channel runoff
  integer , pointer :: ocn_ixy(:)  ! RTM longitude index of ocean runoff p
  integer , pointer :: lnd_jxy(:)  ! RTM latitude index of channel runoff
  integer , pointer :: ocn_jxy(:)  ! RTM latitude index of ocean runoff po
  real(r8), pointer :: lnd_area(:) ! RTM gridcell area (km^2)
  real(r8), pointer :: ocn_area(:) ! RTM gridcell area (km^2)
  integer :: beg_ocn               ! RTM beginning ocn runoff indices on t
  integer :: end_ocn               ! RTM ending ocn runoff indices on this
  integer :: beg_lnd               ! RTM beginning land runoff indices on
  integer :: end_lnd               ! RTM ending land runoff indices on thi
  integer , pointer :: num_ocn(:)  ! RTM per- proc ending ocn runoff indic
  integer , pointer :: num_lnd(:)  ! RTM per- proc ending ocn runoff indic
end type runoff_flow
type (runoff_flow) :: runoff
public runoff_flow
```

PUBLIC MEMBER FUNCTIONS:

```
public set_roflnd
public set_rofocn
public set_proc_rof_bounds
public get_proc_rof_total
public get_proc_rof_global
public UpdateRunoff
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.1 set_roflnd

INTERFACE:

```
subroutine set_roflnd(nrlon, nrlat, mask, area, nsize)
```

DESCRIPTION:

Allocate memory and initialize ocean runoff

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: nrlon
integer , intent(in) :: nrlat
integer , intent(in) :: mask(nrlon,nrlat)
real(r8), intent(in) :: area(nrlon,nrlat)
integer , intent(in) :: nsize
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.2 set_rofocn

INTERFACE:

```
subroutine set_rofocn(nrlon, nrlat, mask, area, nsize)
```

DESCRIPTION:

Allocate memory and initialize ocean runoff

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: nrlon
integer , intent(in) :: nrlat
integer , intent(in) :: mask(nrlon,nrlat)
real(r8), intent(in) :: area(nrlon,nrlat)
integer , intent(in) :: nsize
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.3 set_proc_rof_bounds

INTERFACE:

```
subroutine set_proc_rof_bounds()
```

DESCRIPTION:

Set the beginning and ending indices of the per-process
beginning and ending indices for land and ocean runoff

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use spmdMod        , only : npes, iam
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.4 UpdateRunoff

INTERFACE:

```
subroutine UpdateRunoff(nrloni, nrlonf, nrlati, nrlatf, flxocn_r, flxlnd_r)
```

DESCRIPTION:

Update the land and ocean runoff vectors. This determine the
ocean runoff vectors to send to the coupler as well as the
ocean and land runoff vectors for history output.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: nrloni
integer , intent(in) :: nrlonf
integer , intent(in) :: nrlati
integer , intent(in) :: nrlatf
real(r8), intent(in) :: flxocn_r(nrloni:nrlonf, nrlati:nrlatf)
real(r8), intent(in) :: flxlnd_r(nrloni:nrlonf, nrlati:nrlatf)
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.5 `get_proc_rof_total`

INTERFACE:

```
subroutine get_proc_rof_total(pid, num_lnd, num_ocn)
```

DESCRIPTION:

Determine number of land and ocean runoff points for this process

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer, intent(in)  :: pid
integer, intent(out) :: num_lnd
integer, intent(out) :: num_ocn
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.6 `get_proc_rof_bounds`

INTERFACE:

```
subroutine get_proc_rof_bounds(beg_lnd, end_lnd, beg_ocn, end_ocn)
```

DESCRIPTION:

Determine beginning and ending indices of land and ocean runoff for this processor.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer, intent(out) :: beg_lnd
integer, intent(out) :: end_lnd
integer, intent(out) :: beg_ocn
integer, intent(out) :: end_ocn
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.26.7 get_proc_rof_global

INTERFACE:

```
subroutine get_proc_rof_global(num_lnd, num_ocn)
```

DESCRIPTION:

Determine number of land and ocean runoff points across all processors.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use spmdMod      , only : npes
```

ARGUMENTS:

```
implicit none
integer, intent(out) :: num_lnd
integer, intent(out) :: num_ocn
```

REVISION HISTORY:

Mariana Vertenstein: Created 10/2003

A.27 Module STATICEcosysDynMod (Source File: STATICEcosysDynMod.F90)

Static Ecosystem dynamics: phenology, vegetation.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils,    only: endrun
#if ( defined SCAM )
use scamMod, only :initlonidx,initlatidx
#endif
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: EcosystemDyn      ! Ecosystem dynamics: phenology, vegetation
public :: EcosystemDynini   ! Dynamically allocate memory
public :: interpMonthlyVeg ! interpolate monthly vegetation data
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.27.1 EcosystemDynini

INTERFACE:

```
subroutine EcosystemDynini ()
```

DESCRIPTION:

Dynamically allocate memory and set to signaling NaN.

USES:

```
use nanMod
use decompMod, only : get_proc_global
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

A.27.2 EcosystemDyn

INTERFACE:

```
subroutine EcosystemDyn(lbp, ubp, num_nolakep, filter_nolakep, doalb)
```

DESCRIPTION:

Ecosystem dynamics: phenology, vegetation Calculates leaf areas (tlai, elai), stem areas (tsai, esai) and height (htop).

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_nolakep        ! number of column non
integer, intent(in) :: filter_nolakep(ubp-lbp+1) ! pft filter for non-l
logical, intent(in) :: doalb              ! true = surface albed
```

CALLED FROM:

REVISION HISTORY:

Author: Gordon Bonan
2/1/02, Peter Thornton: Migrated to new data structure.

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: pcolumn(:) ! column index associated with each pft
real(r8), pointer :: snowdp(:) ! snow height (m)
local pointers to implicit out arguments
real(r8), pointer :: tlai(:) ! one-sided leaf area index, no burying
real(r8), pointer :: tsai(:) ! one-sided stem area index, no burying
real(r8), pointer :: htop(:) ! canopy top (m)
real(r8), pointer :: hbot(:) ! canopy bottom (m)
real(r8), pointer :: elai(:) ! one-sided leaf area index with burying
real(r8), pointer :: esai(:) ! one-sided stem area index with burying
integer , pointer :: frac_veg_nosno_alb(:) ! frac of vegetation not cover
```

A.27.3 interpMonthlyVeg

INTERFACE:

```
subroutine interpMonthlyVeg ()
```

DESCRIPTION:

Determine if 2 new months of data are to be read.

USES:

```

        use clm_varctl , only : fsurdat
#ifdef COUP_CAM
        use time_manager, only : get_curr_date, get_step_size, get_perp_date, is_
#else
        use time_manager, only : get_curr_date, get_step_size
#endif

```

ARGUMENTS:

```

        implicit none

```

REVISION HISTORY:

Created by Mariana Vertenstein

A.27.4 readMonthlyVegetation

INTERFACE:

```

        subroutine readMonthlyVegetation (fveg, kmo, kda, months)

```

DESCRIPTION:

Read monthly vegetation data for two consec. months.

USES:

```

        use clmtype
        use decompMod , only : get_proc_global
        use clm_varpar , only : lsmlon, lsmlat, maxpatch_pft, maxpatch, npatch_c
        use clm_varsur , only : all_pfts_on_srfdat
        use fileutils , only : getfil
#ifdef (defined SPMD)
        use spmdMod , only : masterproc, mpicom, MPI_REAL8
#else
        use spmdMod , only : masterproc
#endif
        use time_manager, only : get_nstep
        use ncdio , only : check_ret
#ifdef ( defined SCAM )
        use getnetcdfdata
#endif

```

ARGUMENTS:

```

        implicit none
        include 'netcdf.inc'
        character(len=*) , intent(in) :: fveg ! file with monthly vegetation data
        integer, intent(in) :: kmo ! month (1, ..., 12)
        integer, intent(in) :: kda ! day of month (1, ..., 31)
        integer, intent(in) :: months(2) ! months to be interpolated (1 to 1

```

REVISION HISTORY:

Created by Sam Levis

A.28 Module SnowHydrologyMod (Source File: SnowHydrologyMod.F90)

Calculate snow hydrology.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: SnowWater          ! Change of snow mass and the snow water onto s
public :: SnowCompaction     ! Change in snow layer thickness due to compact
public :: CombineSnowLayers  ! Combine snow layers less than a min thickness
public :: DivideSnowLayers   ! Subdivide snow layers if they exceed maximum
public :: BuildSnowFilter     ! Construct snow/no-snow filters
!PRIVATE MEMBER FUNCTIONS:
private :: Combo             ! Returns the combined variables: dz, t, wliq,
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.28.1 SnowWater

INTERFACE:

```
subroutine SnowWater(lbc, ubc, num_snowc, filter_snowc, &
                    num_nosnowc, filter_nosnowc)
```

DESCRIPTION:

Evaluate the change of snow mass and the snow water onto soil. Water flow within snow is computed by an explicit and non-physical based scheme, which permits a part of liquid water over the holding capacity (a tentative value is used, i.e. equal to $0.033 \times \text{porosity}$) to percolate into the underlying layer. Except for cases where the porosity of one of the two neighboring layers is less than 0.05, zero flow is assumed. The water flow out of the bottom of the snow pack will participate as the input of the soil water and runoff. This subroutine uses a filter for columns containing snow which must be constructed prior to being called.

USES:

```
use clmtype
use clm_varcon , only : denh2o, denice, wimp, ssi
use time_manager, only : get_step_size
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbc, ubc          ! column bounds
integer, intent(in) :: num_snowc         ! number of snow point
integer, intent(in) :: filter_snowc(ubc-lbc+1) ! column filter for sn
integer, intent(in) :: num_nosnowc       ! number of non-snow p
integer, intent(in) :: filter_nosnowc(ubc-lbc+1) ! column filter for no
```

CALLED FROM:

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 15 November 2000: Mariana Vertenstein
 2/26/02, Peter Thornton: Migrated to new data structures.

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer , pointer :: snl(:)           !number of snow layers
  logical , pointer :: do_capsnow(:)    !true => do snow capping
  real(r8), pointer :: qflx_snomelt(:)   !snow melt (mm H2O /s)
  real(r8), pointer :: qflx_rain_grnd(:) !rain on ground after intercepti
  real(r8), pointer :: qflx_sub_snow(:)  !sublimation rate from snow pack
  real(r8), pointer :: qflx_evap_grnd(:) !ground surface evaporation rate
  real(r8), pointer :: qflx_dew_snow(:)  !surface dew added to snow pack
  real(r8), pointer :: qflx_dew_grnd(:)  !ground surface dew formation (m
  real(r8), pointer :: dz(:, :)         !layer depth (m)
local pointers to implicit out arguments
  real(r8), pointer :: qflx_top_soil(:)  !net water input into soil from
local pointers to implicit inout arguments
  real(r8), pointer :: h2soi_ice(:, :)   !ice lens (kg/m2)
  real(r8), pointer :: h2soi_liq(:, :)   !liquid water (kg/m2)

```

A.28.2 SnowCompaction

INTERFACE:

```
subroutine SnowCompaction(lbc, ubc, num_snowc, filter_snowc)
```

DESCRIPTION:

Determine the change in snow layer thickness due to compaction and settling. Three metamorphisms of changing snow characteristics are implemented, i.e., destructive, overburden, and melt. The treatments of the former two are from SNTHERM.89 and SNTHERM.99 (1991, 1999). The contribution due to melt metamorphism is simply taken as a ratio of snow ice fraction after the melting versus before the melting.

USES:

```

use clmtype
use time_manager, only : get_step_size
use clm_varcon , only : denice, denh2o, tfrz

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbc, ubc           ! column bounds
integer, intent(in) :: num_snowc         ! number of column snow po
integer, intent(in) :: filter_snowc(ubc-lbc+1) ! column filter for snow p

```

CALLED FROM:

```
subroutine Hydrology2 in module Hydrology2Mod
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
2/28/02, Peter Thornton: Migrated to new data structures
```

LOCAL VARIABLES:

```
local pointers to implicit in scalars
integer, pointer :: snl(:)           !number of snow layers
local pointers to implicit in arguments
integer, pointer :: imelt(:, :)      !flag for melting (=1), freezing (
real(r8), pointer :: frac_iceold(:, :) !fraction of ice relative to the t
real(r8), pointer :: t_soisno(:, :)  !soil temperature (Kelvin)
real(r8), pointer :: h2soi_ice(:, :) !ice lens (kg/m2)
real(r8), pointer :: h2soi_liq(:, :) !liquid water (kg/m2)
local pointers to implicit inout arguments
real(r8), pointer :: dz(:, :)       !layer depth (m)
```

A.28.3 CombineSnowLayers

INTERFACE:

```
subroutine CombineSnowLayers(lbc, ubc, num_snowc, filter_snowc)
```

DESCRIPTION:

Combine snow layers that are less than a minimum thickness or mass. If the snow element thickness or mass is less than a prescribed minimum, then it is combined with a neighboring element. The subroutine `clm_combo.f90` then executes the combination of mass and energy.

USES:

```
use clmtype
use clm_varcon, only : istsoil
```

ARGUMENTS:

```
implicit none
integer, intent(in)    :: lbc, ubc           ! column bounds
integer, intent(inout) :: num_snowc         ! number of column
integer, intent(inout) :: filter_snowc(ubc-lbc+1) ! column filter for
```

CALLED FROM:

```
subroutine Hydrology2 in module Hydrology2Mod
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
2/28/02, Peter Thornton: Migrated to new data structures.
```

LOCAL VARIABLES:

```

local pointers to implicit in arguments
  integer, pointer :: clandunit(:)      !landunit index for each column
  integer, pointer :: ityplun(:)        !landunit type
local pointers to implicit inout arguments
  integer , pointer :: snl(:)           !number of snow layers
  real(r8), pointer :: h2osno(:)        !snow water (mm H2O)
  real(r8), pointer :: snowdp(:)       !snow height (m)
  real(r8), pointer :: dz(:, :)        !layer depth (m)
  real(r8), pointer :: zi(:, :)        !interface level below a "z" level
  real(r8), pointer :: t_soisno(:, :)  !soil temperature (Kelvin)
  real(r8), pointer :: h2osoi_ice(:, :) !ice lens (kg/m2)
  real(r8), pointer :: h2osoi_liq(:, :) !liquid water (kg/m2)
local pointers to implicit out arguments
  real(r8), pointer :: z(:, :)         !layer thickness (m)

```

A.28.4 DivideSnowLayers

INTERFACE:

```
subroutine DivideSnowLayers(lbc, ubc, num_snowc, filter_snowc)
```

DESCRIPTION:

Subdivides snow layers if they exceed their prescribed maximum thickness.

USES:

```
use clmtype
```

ARGUMENTS:

```

implicit none
integer, intent(in)    :: lbc, ubc           ! column bounds
integer, intent(inout) :: num_snowc         ! number of column
integer, intent(inout) :: filter_snowc(ubc-lbc+1) ! column filter for

```

CALLED FROM:

```
subroutine Hydrology2 in module Hydrology2Mod
```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
2/28/02, Peter Thornton: Migrated to new data structures.

```

LOCAL VARIABLES:

```

local pointers to implicit inout arguments
  integer , pointer :: snl(:)           !number of snow layers
  real(r8), pointer :: dz(:, :)        !layer depth (m)
  real(r8), pointer :: zi(:, :)        !interface level below a "z" level
  real(r8), pointer :: t_soisno(:, :)  !soil temperature (Kelvin)
  real(r8), pointer :: h2osoi_ice(:, :) !ice lens (kg/m2)
  real(r8), pointer :: h2osoi_liq(:, :) !liquid water (kg/m2)
local pointers to implicit out arguments
  real(r8), pointer :: z(:, :)         !layer thickness (m)

```

A.28.5 Combo

INTERFACE:

```
subroutine Combo(dz, wliq, wice, t, dz2, wliq2, wice2, t2)
```

DESCRIPTION:

Combines two elements and returns the following combined variables: dz, t, wliq, wice. The combined temperature is based on the equation: the sum of the enthalpies of the two elements = that of the combined element.

USES:

```
use clm_varcon, only : cpice, cpliq, tfrz, hfus
```

ARGUMENTS:

```
implicit none
real(r8), intent(in)    :: dz2    ! nodal thickness of 2 elements being co
real(r8), intent(in)    :: wliq2  ! liquid water of element 2 [kg/m2]
real(r8), intent(in)    :: wice2  ! ice of element 2 [kg/m2]
real(r8), intent(in)    :: t2     ! nodal temperature of element 2 [K]
real(r8), intent(inout) :: dz     ! nodal thickness of 1 elements being co
real(r8), intent(inout) :: wliq   ! liquid water of element 1
real(r8), intent(inout) :: wice   ! ice of element 1 [kg/m2]
real(r8), intent(inout) :: t      ! nodal temperature of element 1 [K]
```

CALLED FROM:

```
subroutine CombineSnowLayers in this module
subroutine DivideSnowLayers in this module
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
```

A.28.6 BuildSnowFilter

INTERFACE:

```
subroutine BuildSnowFilter(lbc, ubc, num_nolakec, filter_nolakec, &
                           num_snowc, filter_snowc, &
                           num_nosnowc, filter_nosnowc)
```

DESCRIPTION:

Constructs snow filter for use in vectorized loops for snow hydrology.

USES:

```
use clmtype
```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbc, ubc                ! column bounds
integer, intent(in) :: num_nolakec             ! number of column no
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for n
integer, intent(out) :: num_snowc              ! number of column sn
integer, intent(out) :: filter_snowc(ubc-lbc+1) ! column filter for s
integer, intent(out) :: num_nosnowc           ! number of column no
integer, intent(out) :: filter_nosnowc(ubc-lbc+1) ! column filter for n

```

CALLED FROM:

```

subroutine Hydrology2 in Hydrology2Mod
subroutine CombineSnowLayers in this module

```

REVISION HISTORY:

2003 July 31: Forrest Hoffman

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: snl(:)                ! number of snow layer

```

A.29 Module SoilHydrologyMod (Source File: SoilHydrologyMod.F90)

Calculate soil hydrology

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: SurfaceRunoff ! Calculate surface runoff
public :: Infiltration  ! Calculate infiltration into surface soil layer
public :: SoilWater     ! Calculate soil hydrology
public :: Drainage      ! Calculate subsurface drainage
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.29.1 SurfaceRunoff

INTERFACE:

```
subroutine SurfaceRunoff (lbc, ubc, num_soilc, filter_soilc, &
    zwice, vol_liq, s, zwt, fcov)
```

DESCRIPTION:

Calculate surface runoff

The original code was provide by Robert E. Dickinson based on following clues: exponential decrease of Ksat, a water table level determination level including highland and lowland levels and fractional area of wetland (water table above the surface). Runoff is parameterized from the lowlands in terms of precip incident on wet areas and a base flow, where these are estimated using ideas from TOPMODEL.

The original scheme was modified by Z.-L. Yang and G.-Y. Niu,

- o using a new method to determine water table depth and the fractional wet area (fcov)
- o computing runoff (surface and subsurface) from this fraction and the remaining fraction (i.e. 1-fcov)
- o for the 1-fcov part, using BATS1e method to compute surface and subsurface runoff.

The original code on soil moisture and runoff were provided by R. E. Dickinson in July 1996.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon, only : denice, denh2o, wimp
use clm_varpar, only : nlevsoi
```


ARGUMENTS:

```

implicit none
integer , intent(in)  :: lbc, ubc           ! column bounds
integer , intent(in)  :: num_soilc         ! number of column so
integer , intent(in)  :: filter_soilc(ubc-lbc+1) ! column filter for s
real(r8), intent(out) :: zwise(lbc:ubc)     ! the sum of ice mass
real(r8), intent(out) :: vol_liq(lbc:ubc,1:nlevsoi) ! partial volume of l
real(r8), intent(out) :: s(lbc:ubc,1:nlevsoi) ! wetness of soil (in
real(r8), intent(out) :: zwt(lbc:ubc)       ! water table depth
real(r8), intent(out) :: fcov(lbc:ubc)      ! fractional area wit

```

CALLED FROM:

```

subroutine BiogeophysicsLake in module BiogeophysicsLakeMod
subroutine Hydrology2 in module Hydrology2Mod

```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
12 November 1999:  Z.-L. Yang and G.-Y. Niu
15 December 1999:  Paul Houser and Jon Radakovich; F90 Revision
2/26/02, Peter Thornton: Migrated to new data structures.

```

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
integer , pointer :: cgridcell(:)      !gridcell index for each column
real(r8), pointer :: wtfact(:)         !Fraction of model area with high
real(r8), pointer :: qflx_top_soil(:)  !net water input into soil from to
real(r8), pointer :: watsat(:, :)      !volumetric soil water at saturati
real(r8), pointer :: dz(:, :)         !layer depth (m)
real(r8), pointer :: zi(:, :)         !interface level below a "z" level
real(r8), pointer :: h2osoi_ice(:, :)  !ice lens (kg/m2)
real(r8), pointer :: h2osoi_liq(:, :)  !liquid water (kg/m2)
local pointers to original implicit out arguments
real(r8), pointer :: qflx_surf(:)      !surface runoff (mm H2O /s)
real(r8), pointer :: eff_porosity(:, :) !effective porosity = porosity - v

```

A.29.2 Infiltration

INTERFACE:

```

subroutine Infiltration(lbc, ubc, num_soilc, filter_soilc)

```

DESCRIPTION:

Calculate infiltration into surface soil layer (minus the evaporation)
The original code on soil moisture and runoff were provided by
R. E. Dickinson in July 1996.

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype

```

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbc, ubc           ! column bounds
integer, intent(in) :: num_soilc         ! number of column soil
integer, intent(in) :: filter_soilc(ubc-lbc+1) ! column filter for soi

```

CALLED FROM:

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 12 November 1999: Z.-L. Yang and G.-Y. Niu
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 2/27/02, Peter Thornton: Migrated to new data structures.

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
integer , pointer :: snl(:)           ! minus number of snow layers
real(r8), pointer :: qflx_top_soil(:) ! net water input into soil from to
real(r8), pointer :: qflx_surf(:)     ! surface runoff (mm H2O /s)
real(r8), pointer :: qflx_evap_grnd(:)! ground surface evaporation rate (
local pointers to original implicit out arguments
real(r8), pointer :: qflx_infl(:)     !infiltration (mm H2O /s)

```

A.29.3 SoilWater

INTERFACE:

```

subroutine SoilWater(lbc, ubc, num_soilc, &
  filter_soilc, vol_liq, dwat, hk, dhkdw)

```

DESCRIPTION:

Soil hydrology

Soil moisture is predicted from a 10-layer model (as with soil temperature), in which the vertical soil moisture transport is governed by infiltration, runoff, gradient diffusion, gravity, and root extraction through canopy transpiration. The net water applied to the surface layer is the snowmelt plus precipitation plus the throughfall of canopy dew minus surface runoff and evaporation.

The vertical water flow in an unsaturated porous media is described by Darcy's law, and the hydraulic conductivity and the soil negative potential vary with soil water content and soil texture based on the work of Clapp and Hornberger (1978) and Cosby et al. (1984). The equation is integrated over the layer thickness, in which the time rate of change in water mass must equal the net flow across the bounding interface, plus the rate of internal source or sink. The terms of water flow across the layer interfaces are linearly expanded by using first-order Taylor expansion. The equations result in a tridiagonal system equation.

Note: length units here are all millimeter

```

(in temperature subroutine uses same soil layer
structure required but lengths are m)
Richards equation:
d wat      d      d wat d psi
----- = - -- [ k(----- - 1) ] + S
dt         dz         dz d wat
where: wat = volume of water per volume of soil (mm**3/mm**3)
psi = soil matrix potential (mm)
dt = time step (s)
z = depth (mm)
dz = thickness (mm)
qin = inflow at top (mm h2o /s)
qout= outflow at bottom (mm h2o /s)
s = source/sink flux (mm h2o /s)
k = hydraulic conductivity (mm h2o /s)

qin[n+1] = qin[n] + ----- d qin      d qin
                    d wat(j-1) + ----- d wat(j)
                    d wat(j-1)      d wat(j)
                    =====|=====
                        < qin
                    d wat(j)/dt * dz = qin[n+1] - qout[n+1] + S(j)
                        > qout
                    =====|=====
                    d qout      d qout
qout[n+1] = qout[n] + ----- d wat(j) + ----- d wat(j+1)
                    d wat(j)      d wat(j+1)
Solution: linearize k and psi about d wat and use tridiagonal
system of equations to solve for d wat,
where for layer j
r_j = a_j [d wat_j-1] + b_j [d wat_j] + c_j [d wat_j+1]

```

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon      , only : wimp
use clm_varpar      , only : nlevsoi
use shr_const_mod   , only : SHR_CONST_TKFRZ, SHR_CONST_LATICE, SHR_CONST_G
use TridiagonalMod, only : Tridiagonal
use time_manager    , only : get_step_size

```

ARGUMENTS:

```

implicit none
integer , intent(in)  :: lbc, ubc           ! column bounds
integer , intent(in)  :: num_soilc         ! number of column so
integer , intent(in)  :: filter_soilc(ubc-lbc+1) ! column filter for
real(r8), intent(in)  :: vol_liq(lbc:ubc,1:nlevsoi) ! soil water per unit
real(r8), intent(out) :: d wat(lbc:ubc,1:nlevsoi) ! change of soil wate
real(r8), intent(out) :: hk(lbc:ubc,1:nlevsoi) ! hydraulic conductiv
real(r8), intent(out) :: dhkdw(lbc:ubc,1:nlevsoi) ! d(hk)/d(vol_liq)

```

CALLED FROM:

```

subroutine Hydrology2 in module Hydrology2Mod

```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 2/27/02, Peter Thornton: Migrated to new data structures. Includes
 treatment of multiple PFTs on a single soil column.

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
  integer , pointer :: npfts(:)           !column's number of pfts - ADD
  real(r8), pointer :: z(:, :)           !layer depth (m)
  real(r8), pointer :: dz(:, :)          !layer thickness (m)
  real(r8), pointer :: smpmin(:)         !restriction for min of soil po
  real(r8), pointer :: qflx_infl(:)      !infiltration (mm H2O /s)
  real(r8), pointer :: qflx_tran_veg_pft(:) !vegetation transpiration (mm H
  real(r8), pointer :: qflx_tran_veg_col(:) !vegetation transpiration (mm H
  real(r8), pointer :: eff_porosity(:, :) !effective porosity = porosity
  real(r8), pointer :: watsat(:, :)      !volumetric soil water at satur
  real(r8), pointer :: hksat(:, :)       !hydraulic conductivity at satu
  real(r8), pointer :: bsw(:, :)         !Clapp and Hornberger "b"
  real(r8), pointer :: sucsat(:, :)      !minimum soil suction (mm)
  real(r8), pointer :: t_soisno(:, :)    !soil temperature (Kelvin)
  real(r8), pointer :: rootr_pft(:, :)   !effective fraction of roots in
  real(r8), pointer :: wtcol(:)          !weight relative to column for
  integer , pointer :: pfti(:)           !beginning pft index for each c
local pointers to original implicit inout arguments
  real(r8), pointer :: h2osoi_liq(:, :)  !liquid water (kg/m2)
local pointer s to original implicit out arguments
  real(r8), pointer :: rootr_col(:, :)   !effective fraction of roots in

```

A.29.4 Drainage

INTERFACE:

```

subroutine Drainage(lbc, ubc, num_soilc, filter_soilc, &
  twice, vol_liq, s, zwt, fcov, hk, dhkdw, dwat)

```

DESCRIPTION:

Calculate subsurface drainage

The original code was provide by Robert E. Dickinson based on following clues: exponential decrease of Ksat, a water table level determination level including highland and lowland levels and fractional area of wetland (water table above the surface). Runoff is parameterized from the lowlands in terms of precip incident on wet areas and a base flow, where these are estimated using ideas from TOPMODEL.

The original scheme was modified by Z.-L. Yang and G.-Y. Niu,

- * using a new method to determine water table depth and the fractional wet area (fcov)
- * computing runoff (surface and subsurface) from this fraction and the remaining fraction (i.e. 1-fcov)
- * for the 1-fcov part, using BATS1e method to compute

surface and subsurface runoff.

The original code on soil moisture and runoff were provided by
R. E. Dickinson in July 1996.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use time_manager, only : get_step_size
use clm_varcon  , only: pondmx
use clm_varpar  , only : nlevsoi
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc           ! column bounds
integer , intent(in) :: num_soilc         ! number of column soi
integer , intent(in) :: filter_soilc(ubc-lbc+1) ! column filter for so
real(r8), intent(in) :: zwise(lbc:ubc)    ! the sum of ice mass
real(r8), intent(in) :: vol_liq(lbc:ubc,1:nlevsoi) ! partial volume of li
real(r8), intent(in) :: s(lbc:ubc,1:nlevsoi) ! wetness of soil (inc
real(r8), intent(in) :: zwt(lbc:ubc)      ! water table depth
real(r8), intent(in) :: fcov(lbc:ubc)     ! fractional area with
real(r8), intent(in) :: hk(lbc:ubc,1:nlevsoi) ! hydraulic conductivi
real(r8), intent(in) :: dhkdw(lbc:ubc,1:nlevsoi) ! d(hk)/d(vol_liq)
real(r8), intent(in) :: dwat(lbc:ubc,1:nlevsoi) ! change in soil water
```

CALLED FROM:

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
12 November 1999: Z.-L. Yang and G.-Y. Niu
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision

LOCAL VARIABLES:

```
local pointers to original implicit in arguments
integer , pointer:: snl(:)           !number of snow layers
real(r8), pointer:: qflx_snowcap(:) !excess precipitation due to snow c
real(r8), pointer:: qflx_dew_grnd(:) !ground surface dew formation (mm H
real(r8), pointer:: qflx_dew_snow(:) !surface dew added to snow pack (mm
real(r8), pointer:: qflx_sub_snow(:) !sublimation rate from snow pack (m
real(r8), pointer:: dz(:,:)         !layer depth (m)
real(r8), pointer:: bsw(:,:)        !Clapp and Hornberger "b"
real(r8), pointer:: eff_porosity(:,:) !effective porosity = porosity - vo
local pointers to original implicit inout arguments
real(r8), pointer:: h2osoi_ice(:,:) !ice lens (kg/m2)
real(r8), pointer:: h2osoi_liq(:,:) !liquid water (kg/m2)
local pointers to original implicit out arguments
real(r8), pointer:: qflx_drain(:)    !sub-surface runoff (mm H2O /s)
real(r8), pointer:: qflx_qrgwl(:)    !qflx_surf at glaciers, wetlands, l
real(r8), pointer:: eflx_impsoil(:)  !implicit evaporation for soil temp
```

A.30 Module SoilTemperatureMod (Source File: SoilTemperatureMod.F90)

Calculates snow and soil temperatures including phase change

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: SoilTemperature ! Snow and soil temperatures including phase cha
!PRIVATE MEMBER FUNCTIONS:
private :: SoilThermProp ! Set therm conductivities and heat cap of snow/
private :: PhaseChange ! Calculation of the phase change within snow an
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.30.1 SoilTemperature

INTERFACE:

```
subroutine SoilTemperature(lbc, ubc, num_nolakec, filter_nolakec, &
                          xmf, fact)
```

DESCRIPTION:

Snow and soil temperatures including phase change

- o The volumetric heat capacity is calculated as a linear combination in terms of the volumetric fraction of the constituent phases.
- o The thermal conductivity of soil is computed from the algorithm of Johansen (as reported by Farouki 1981), and the conductivity of snow is from the formulation used in SNTHERM (Jordan 1991).
- o Boundary conditions:
F = Rnet - Hg - LEg (top), F= 0 (base of the soil column).
- o Soil / snow temperature is predicted from heat conduction in 10 soil layers and up to 5 snow layers.
The thermal conductivities at the interfaces between two neighboring layers (j, j+1) are derived from an assumption that the flux across the interface is equal to that from the node j to the interface and the flux from the interface to the node j+1. The equation is solved using the Crank-Nicholson method and results in a tridiagonal system equation.

USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
use clmtype
use time_manager , only : get_step_size
use clm_varcon , only : sb, capr, cnfac
use clm_varpar , only : nlevsno, nlevsoi
use TridiagonalMod, only : Tridiagonal
```

ARGUMENTS:

```

implicit none
integer , intent(in)  :: lbc, ubc           ! column bounds
integer , intent(in)  :: num_nolakec       ! number of column n
integer , intent(in)  :: filter_nolakec(ubc-lbc+1) ! column filter for
real(r8), intent(out) :: xmf(lbc:ubc)      ! total latent heat
real(r8), intent(out) :: fact(lbc:ubc, -nlevsno+1:nlevsoi) ! used in comp

```

CALLED FROM:

```

subroutine Biogeophysics2 in module Biogeophysics2Mod

```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
12/19/01, Peter Thornton
Changed references for tg to t_grnd, for consistency with the
rest of the code (tg eliminated as redundant)
2/14/02, Peter Thornton: Migrated to new data structures. Added pft loop
in calculation of net ground heat flux.

```

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
integer , pointer :: pgridcell(:)      !pft's gridcell index
integer , pointer :: npfts(:)          !column's number of pfts - ADD
integer , pointer :: pfti(:)          !column's beginning pft index - A
real(r8), pointer :: forc_lwrad(:)     !downward infrared (longwave) rad
integer , pointer :: snl(:)            !number of snow layers
real(r8), pointer :: htvp(:)           !latent heat of vapor of water (o
real(r8), pointer :: emg(:)           !ground emissivity
real(r8), pointer :: cgrnd(:)         !deriv. of soil energy flux wrt t
real(r8), pointer :: dlrads(:)        !downward longwave radiation blow
real(r8), pointer :: sabg(:)          !solar radiation absorbed by grou
integer , pointer :: frac_veg_nosno(:) !fraction of vegetation not cover
real(r8), pointer :: eflx_sh_grnd(:)  !sensible heat flux from ground (
real(r8), pointer :: qflx_evap_soi(:) !soil evaporation (mm H2O/s) (+ =
real(r8), pointer :: wtcol(:)         !weight of pft relative to column

local pointers to original implicit inout arguments
real(r8), pointer :: t_grnd(:)        !ground surface temperature [K]

local pointers to original implicit out arguments
these two are new variables added to clmtype at the pft level
real(r8), pointer :: eflx_gnet(:)     !net ground heat flux into the su
real(r8), pointer :: dgnetdT(:)      !temperature derivative of ground

local pointers to original implicit in arrays
real(r8), pointer:: zi(:, :)         !interface level below a "z" leve
real(r8), pointer:: dz(:, :)         !layer depth (m)
real(r8), pointer:: z(:, :)          !layer thickness (m)
real(r8), pointer:: t_soisno(:, :)   !soil temperature (Kelvin)

```

A.30.2 SoilThermProp

INTERFACE:

```
subroutine SoilThermProp (lbc, ubc, num_nolakec, filter_nolakec, tk, cv)
```

DESCRIPTION:

Calculation of thermal conductivities and heat capacities of snow/soil layers

- (1) The volumetric heat capacity is calculated as a linear combination in terms of the volumetric fraction of the constituent phases.
- (2) The thermal conductivity of soil is computed from the algorithm of Johansen (as reported by Farouki 1981), and of snow is from the formulation used in SNTHERM (Jordan 1991).

The thermal conductivities at the interfaces between two neighboring layers (j, j+1) are derived from an assumption that the flux across the interface is equal to that from the node j to the interface and the flux from the interface to the node j+1.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use clm_varcon , only : denh2o, denice, tfrz, tkwat, tkice, tkair, &
                        cpice, cpliq, istice, istwet
use clm_varpar , only : nlevsno, nlevsoi
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc                ! column bounds
integer , intent(in) :: num_nolakec             ! number of column
integer , intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter f
real(r8), intent(out) :: cv(lbc:ubc,-nlevsno+1:nlevsoi) ! heat capacity [
real(r8), intent(out) :: tk(lbc:ubc,-nlevsno+1:nlevsoi) ! thermal conduct
```

CALLED FROM:

```
subroutine SoilTemperature in this module
```

REVISION HISTORY:

15 September 1999: Yongjiu Dai; Initial code
 15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
 2/13/02, Peter Thornton: migrated to new data structures
 7/01/03, Mariana Vertenstein: migrated to vector code

LOCAL VARIABLES:

```
local pointers to original implicit in scalars
integer , pointer :: clandunit(:) !column's landunit
integer , pointer :: ityplun(:)   !landunit type
integer , pointer :: snl(:)       !number of snow layers
real(r8), pointer :: h2osno(:)    !snow water (mm H2O)
local pointers to original implicit in arrays
real(r8), pointer :: watsat(:, :) !volumetric soil water at saturatio
real(r8), pointer :: tksatu(:, :) !thermal conductivity, saturated so
real(r8), pointer :: tkmg(:, :)  !thermal conductivity, soil mineral
real(r8), pointer :: tkdry(:, :) !thermal conductivity, dry soil (W/
real(r8), pointer :: csol(:, :)  !heat capacity, soil solids (J/m**3
```



```

real(r8), pointer :: dz(:, :)      !layer depth (m)
real(r8), pointer :: zi(:, :)      !interface level below a "z" level
real(r8), pointer :: z(:, :)      !layer thickness (m)
real(r8), pointer :: t_soiso(:, :) !soil temperature (Kelvin)
real(r8), pointer :: h2soi_liq(:, :) !liquid water (kg/m2)
real(r8), pointer :: h2soi_ice(:, :) !ice lens (kg/m2)

```

A.30.3 PhaseChange

INTERFACE:

```

subroutine PhaseChange (lbc, ubc, num_nolakec, filter_nolakec, fact, &
                        brr, hs, dhdsT, xmf)

```

DESCRIPTION:

Calculation of the phase change within snow and soil layers:

- (1) Check the conditions for which the phase change may take place, i.e., the layer temperature is great than the freezing point and the ice mass is not equal to zero (i.e. melting), or the layer temperature is less than the freezing point and the liquid water mass is not equal to zero (i.e. freezing).
- (2) Assess the rate of phase change from the energy excess (or deficit) after setting the layer temperature to freezing point.
- (3) Re-adjust the ice and liquid mass, and the layer temperature

USES:

```

use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use time_manager, only : get_step_size
use clm_varcon , only : tfrz, hfus
use clm_varpar , only : nlevsno, nlevsoi

```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbc, ubc                ! column bou
integer , intent(in) :: num_nolakec             ! number of
integer , intent(in) :: filter_nolakec(ubc-lbc+1) ! column fil
real(r8), intent(in) :: brr (lbc:ubc, -nlevsno+1:nlevsoi) ! temporary
real(r8), intent(in) :: fact (lbc:ubc, -nlevsno+1:nlevsoi) ! temporary
real(r8), intent(in) :: hs (lbc:ubc)           ! net ground
real(r8), intent(in) :: dhdsT (lbc:ubc)        ! temperatur
real(r8), intent(out):: xmf (lbc:ubc)          ! total late

```

CALLED FROM:

```

subroutine SoilTemperature in this module

```

REVISION HISTORY:

```

15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
2/14/02, Peter Thornton: Migrated to new data structures.
7/01/03, Mariana Vertenstein: Migrated to vector code

```

LOCAL VARIABLES:

```

local pointers to original implicit in scalars
  integer , pointer :: snl(:)          !number of snow layers
  real(r8), pointer :: h2osno(:)       !snow water (mm H2O)
local pointers to original implicit inout scalars
  real(r8), pointer :: snowdp(:)      !snow height (m)
local pointers to original implicit out scalars
  real(r8), pointer :: qflx_snomelt(:) !snow melt (mm H2O /s)
  real(r8), pointer :: eflx_snomelt(:) !snow melt heat flux (W/m**2)
local pointers to original implicit in arrays
  real(r8), pointer :: h2osoi_liq(:, :) !liquid water (kg/m2) (new)
  real(r8), pointer :: h2osoi_ice(:, :) !ice lens (kg/m2) (new)
  real(r8), pointer :: tssbef(:, :)     !temperature at previous time step
local pointers to original implicit inout arrays
  real(r8), pointer :: t_soisno(:, :)  !soil temperature (Kelvin)
local pointers to original implicit out arrays
  integer, pointer :: imelt(:, :)      !flag for melting (=1), freezing (=

```


A.31 Module SurfaceAlbedoMod (Source File: SurfaceAlbedoMod.F90)

Performs surface albedo calculations

PUBLIC TYPES:

```
use shr_sys_mod, only : shr_sys_flush
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: SurfaceAlbedo ! Surface albedo and two-stream fluxes
public :: SnowAge       ! Update snow age
!PRIVATE MEMBER FUNCTIONS:
private :: SnowAlbedo   ! Determine snow albedos
private :: SoilAlbedo   ! Determine ground surface albedo
private :: TwoStream    ! Two-stream fluxes for canopy radiative transfer
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.31.1 SurfaceAlbedo

INTERFACE:

```
subroutine SurfaceAlbedo(lbg, ubg, lbc, ubc, lbp, ubp, &
    caldayp1, eccen, obliqr, lambm0, mvelpp)
```

DESCRIPTION:

Surface albedo and two-stream fluxes
 Surface albedos. Also fluxes (per unit incoming direct and diffuse radiation) reflected, transmitted, and absorbed by vegetation.
 Also sunlit fraction of the canopy.
 The calling sequence is:
 -> SurfaceAlbedo: albedos for next time step
 -> SnowAlbedo: snow albedos: direct beam
 -> SnowAlbedo: snow albedos: diffuse
 -> SoilAlbedo: soil/lake/glacier/wetland albedos
 -> TwoStream: absorbed, reflected, transmitted solar fluxes (vis dir,

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use shr_orb_mod
```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbg, ubg ! gridcell bounds
integer , intent(in) :: lbc, ubc ! column bounds
integer , intent(in) :: lbp, ubp ! pft bounds
real(r8), intent(in) :: caldayp1 ! calendar day at Greenwich (1.00, ...,
real(r8), intent(in) :: eccen ! Earth's orbital eccentricity
real(r8), intent(in) :: obliqr ! Earth's obliquity in radians
real(r8), intent(in) :: lambm0 ! Mean longitude of perihelion at the ve
real(r8), intent(in) :: mvelpp ! Earth's moving vernal equinox long. of

```

CALLED FROM:

```

subroutine lpjreset1 in module DGVMMod (only applicable when cpp token DGVM
subroutine driver
subroutine iniTimeVar

```

REVISION HISTORY:

Author: Gordon Bonan
2/1/02, Peter Thornton: Migrate to new data structures
8/20/03, Mariana Vertenstein: Vectorized routine

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: pgridcell(:) ! gridcell of corresponding pft
integer , pointer :: pcolumn(:) ! column of corresponding pft
integer , pointer :: cgridcell(:) ! gridcell of corresponding column
real(r8), pointer :: lat(:) ! gridcell latitude (radians)
real(r8), pointer :: lon(:) ! gridcell longitude (radians)
real(r8), pointer :: elai(:) ! one-sided leaf area index with buryin
real(r8), pointer :: esai(:) ! one-sided stem area index with buryin
real(r8), pointer :: h2osno(:) ! snow water (mm H2O)
real(r8), pointer :: snowage(:) ! non dimensional snow age [-]
real(r8), pointer :: rho1(:, :) ! leaf reflectance: 1=vis, 2=nir
real(r8), pointer :: rho2(:, :) ! stem reflectance: 1=vis, 2=nir
real(r8), pointer :: tau1(:, :) ! leaf transmittance: 1=vis, 2=nir
real(r8), pointer :: tau2(:, :) ! stem transmittance: 1=vis, 2=nir
integer , pointer :: ivt(:) ! pft vegetation type

local pointers to implicit out arguments
real(r8), pointer :: fsun(:) ! sunlit fraction of canopy
real(r8), pointer :: albgrd(:, :) ! ground albedo (direct)
real(r8), pointer :: albgri(:, :) ! ground albedo (diffuse)
real(r8), pointer :: albd(:, :) ! surface albedo (direct)
real(r8), pointer :: albi(:, :) ! surface albedo (diffuse)
real(r8), pointer :: fabd(:, :) ! flux absorbed by veg per unit direct
real(r8), pointer :: fabi(:, :) ! flux absorbed by veg per unit diffuse
real(r8), pointer :: ftdd(:, :) ! down direct flux below veg per unit d
real(r8), pointer :: ftid(:, :) ! down diffuse flux below veg per unit
real(r8), pointer :: ftii(:, :) ! down diffuse flux below veg per unit

```

A.31.2 SnowAlbedo

INTERFACE:

```
subroutine SnowAlbedo (lbc, ubc, coszen, ind, alb)
```

DESCRIPTION:

Determine snow albedos

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
```

ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc           ! column bounds
real(r8), intent(in) :: coszen(lbc:ubc)    ! cosine solar zenith an
integer , intent(in) :: ind               ! 0=direct beam, 1=diffu
real(r8), intent(out):: alb(lbc:ubc,2)     ! snow albedo by waveban
```

CALLED FROM:

```
subroutine SurfaceAlbedo in this module
```

REVISION HISTORY:

Author: Gordon Bonan
 2/5/02, Peter Thornton: Migrated to new data structures. Eliminated
 reference to derived types in this subroutine, and made consistent use
 of the assumption that numrad = 2, with index values: 1=visible,2=NIR
 8/20/03, Mariana Vertenstein: Vectorized routine

LOCAL VARIABLES:

```
local pointers to implicit in arguments
real(r8), pointer :: h2osno(:)    ! snow water (mm H2O)
real(r8), pointer :: snowage(:)   ! non dimensional snow age [-]
```

A.31.3 SoilAlbedo

INTERFACE:

```
subroutine SoilAlbedo (lbc, ubc, coszen, albsnd, albsni)
```

DESCRIPTION:

Determine ground surface albedo, accounting for snow

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varpar, only : numrad
use clm_varcon, only : albsat, albdry, alblak, albice, tfrz, istic, ists
```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbc, ubc          ! column bounds
real(r8), intent(in) :: coszen(lbc:ubc)   ! cos solar zenith angle
real(r8), intent(in) :: albsnd(lbc:ubc,numrad) ! snow albedo (direct)
real(r8), intent(in) :: albsni(lbc:ubc,numrad) ! snow albedo (diffuse)

```

CALLED FROM:

```

subroutine SurfaceAlbedo in this module

```

REVISION HISTORY:

```

Author: Gordon Bonan
2/5/02, Peter Thornton: Migrated to new data structures.
8/20/03, Mariana Vertenstein: Vectorized routine

```

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
integer , pointer :: clandunit(:) ! landunit of corresponding column
integer , pointer :: ltype(:)    ! landunit type
integer , pointer :: isoicol(:)  ! soil color class
real(r8), pointer :: t_grnd(:)   ! ground temperature (Kelvin)
real(r8), pointer :: frac_sno(:) ! fraction of ground covered by snow
real(r8), pointer :: h2soi_vol(:, :) ! volumetric soil water [m3/m3]
local pointers to original implicit out arguments
real(r8), pointer:: albgrd(:, :) ! ground albedo (direct)
real(r8), pointer:: albgrd(:, :) ! ground albedo (diffuse)

```

A.31.4 TwoStream

INTERFACE:

```

subroutine TwoStream (lbc, ubc, lbp, ubp, filter_vegsol, num_vegsol, &
                     coszen, vai, rho, tau, ftdi, gdir)

```

DESCRIPTION:

Two-stream fluxes for canopy radiative transfer
 Use two-stream approximation of Dickinson (1983) Adv Geophysics
 25:305-353 and Sellers (1985) Int J Remote Sensing 6:1335-1372
 to calculate fluxes absorbed by vegetation, reflected by vegetation,
 and transmitted through vegetation for unit incoming direct or diffuse
 flux given an underlying surface with known albedo.

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varpar, only : numrad
use clm_varcon, only : omegas, tfrz, betads, betais

```

ARGUMENTS:

```

implicit none
integer , intent(in)  :: lbc, ubc           ! column bounds
integer , intent(in)  :: lbp, ubp           ! pft bounds
integer , intent(in)  :: filter_vegsol(ubp-lbp+1) ! filter for vegetated
integer , intent(in)  :: num_vegsol        ! number of vegetated p
real(r8), intent(in)  :: coszen(lbp:ubp)   ! cosine solar zenith a
real(r8), intent(in)  :: vai(lbp:ubp)      ! elai+esai
real(r8), intent(in)  :: rho(lbp:ubp,numrad) ! leaf/stem refl weight
real(r8), intent(in)  :: tau(lbp:ubp,numrad) ! leaf/stem tran weight
real(r8), intent(out) :: ftdi(lbp:ubp,numrad) ! down direct flux belo
real(r8), intent(out) :: gdir(lbp:ubp)      ! aver projected leaf/s

```

CALLED FROM:

subroutine SurfaceAlbedo in this module

REVISION HISTORY:

Author: Gordon Bonan
 Modified for speedup: Mariana Vertenstein, 8/26/02
 Vectorized routine: Mariana Vertenstein: 8/20/03

LOCAL VARIABLES:

```

local pointers to implicit in scalars
integer , pointer :: pcolumn(:) ! column of corresponding pft
real(r8), pointer :: albgd(:, :) ! ground albedo (direct) (column-level)
real(r8), pointer :: albgri(:, :) ! ground albedo (diffuse)(column-level)
real(r8), pointer :: t_veg(:) ! vegetation temperature (Kelvin)
real(r8), pointer :: fwet(:) ! fraction of canopy that is wet (0 to
integer , pointer :: ivt(:) ! pft vegetation type
real(r8), pointer :: xl(:) ! ecophys const - leaf/stem orientatio
local pointers to implicit out scalars
real(r8), pointer :: albd(:, :) ! surface albedo (direct)
real(r8), pointer :: albi(:, :) ! surface albedo (diffuse)
real(r8), pointer :: fabd(:, :) ! flux absorbed by veg per unit direct
real(r8), pointer :: fabi(:, :) ! flux absorbed by veg per unit diffus
real(r8), pointer :: ftdd(:, :) ! down direct flux below veg per unit
real(r8), pointer :: ftid(:, :) ! down diffuse flux below veg per unit
real(r8), pointer :: ftii(:, :) ! down diffuse flux below veg per unit

```

A.31.5 SnowAge

INTERFACE:

subroutine SnowAge (lbc, ubc)

DESCRIPTION:

Updates snow age Based on BATS code.

USES:


```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon, only : tfrz
use time_manager, only : get_step_size

```

ARGUMENTS:

```

implicit none
integer , intent(in) :: lbc, ubc ! column bounds

```

CALLED FROM:

REVISION HISTORY:

```

Original Code: Robert Dickinson
15 September 1999: Yongjiu Dai; Integration of code into CLM
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
3/4/02, Peter Thornton: Migrated to new data structures.
8/20/03, Mariana Vertenstein: Vectorized routine

```

LOCAL VARIABLES:

```

local pointers to implicit in arguments
real(r8), pointer :: t_grnd(:)      ! ground temperature (Kelvin)
real(r8), pointer :: h2osno(:)      ! snow water (mm H2O)
real(r8), pointer :: h2osno_old(:) ! snow mass for previous time step (kg)
local pointers to implicit inout arguments
real(r8), pointer :: snow_age(:)    ! non dimensional snow age [-]

```

A.32 Module SurfaceRadiationMod (Source File: SurfaceRadiationMod.F90)

Calculate solar fluxes absorbed by vegetation and ground surface

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: SurfaceRadiation ! Solar fluxes absorbed by veg and ground surfa
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```

A.32.1 SurfaceRadiation

INTERFACE:

```
subroutine SurfaceRadiation(lbp, ubp)
```

DESCRIPTION:

Solar fluxes absorbed by vegetation and ground surface. Note possible problem when land is on different grid than atmosphere. Land may have sun above the horizon ($\coszen < 0$) but atmosphere may have sun below the horizon ($\text{forc_solad} = 0$ and $\text{forc_solai} = 0$). This is okay because all fluxes (absorbed, reflected, transmitted) are multiplied by the incoming flux and all will equal zero. Atmosphere may have sun above horizon ($\text{forc_solad} > 0$ and $\text{forc_solai} > 0$) but land may have sun below horizon. This is okay because fabd , fabi , ftdd , ftid , and ftii all equal zero so that $\text{sabv} = \text{sabg} = \text{fsa} = 0$. Also, albd and albi equal one so that $\text{fsr} = \text{forc_solad} + \text{forc_solai}$. In other words, all the radiation is reflected. However, the way the code is currently implemented, this is only true if $(\text{forc_solad} + \text{forc_solai}) - \text{vis} = (\text{forc_solad} + \text{forc_solai}) - \text{nir}$.

USES:

```
use clmtype
use clm_varpar , only : numrad
use clm_varcon , only : spval
use time_manager, only : get_curr_date, get_step_size
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp      ! pft upper and lower bounds
```

CALLED FROM:

```
subroutine Biogeophysics1 in module Biogeophysics1Mod
subroutine BiogeophysicsLake in module BiogeophysicsLakeMod
```

REVISION HISTORY:

Author: Gordon Bonan

2/18/02, Peter Thornton: Migrated to new data structures. Added a pft loop.

LOCAL VARIABLES:

```

local pointers to original implicit in arguments
  integer , pointer :: pcolumn(:)      ! pft's column index
  integer , pointer :: pgridcell(:)    ! pft's gridcell index
  real(r8), pointer :: fsun(:)         ! sunlit fraction of canopy
  real(r8), pointer :: elai(:)         ! one-sided leaf area index with b
  real(r8), pointer :: esai(:)         ! one-sided stem area index with b
  real(r8), pointer :: londeg(:)       ! longitude (degrees)
  real(r8), pointer :: latdeg(:)       ! latitude (degrees)
  real(r8), pointer :: forc_solad(:, :) ! direct beam radiation (W/m**2)
  real(r8), pointer :: forc_solai(:, :) ! diffuse radiation (W/m**2)
  real(r8), pointer :: fabd(:, :)      ! flux absorbed by veg per unit di
  real(r8), pointer :: fabi(:, :)      ! flux absorbed by veg per unit di
  real(r8), pointer :: ftdd(:, :)      ! down direct flux below veg per u
  real(r8), pointer :: ftid(:, :)      ! down diffuse flux below veg per
  real(r8), pointer :: ftii(:, :)      ! down diffuse flux below veg per
  real(r8), pointer :: albgnd(:, :)    ! ground albedo (direct)
  real(r8), pointer :: albgri(:, :)    ! ground albedo (diffuse)
  real(r8), pointer :: albd(:, :)      ! surface albedo (direct)
  real(r8), pointer :: albi(:, :)      ! surface albedo (diffuse)
local pointers to original implicit out arguments
  real(r8), pointer :: laisun(:)       ! sunlit leaf area
  real(r8), pointer :: laisha(:)       ! shaded leaf area
  real(r8), pointer :: sabg(:)         ! solar radiation absorbed by grou
  real(r8), pointer :: sabv(:)         ! solar radiation absorbed by vege
  real(r8), pointer :: fsa(:)          ! solar radiation absorbed (total)
  real(r8), pointer :: parsun(:)       ! average absorbed PAR for sunlit
  real(r8), pointer :: parsha(:)       ! average absorbed PAR for shaded
  real(r8), pointer :: fsr(:)          ! solar radiation reflected (W/m**
  real(r8), pointer :: fsds_vis_d(:)   ! incident direct beam vis solar r
  real(r8), pointer :: fsds_nir_d(:)   ! incident direct beam nir solar r
  real(r8), pointer :: fsds_vis_i(:)   ! incident diffuse vis solar radia
  real(r8), pointer :: fsds_nir_i(:)   ! incident diffuse nir solar radia
  real(r8), pointer :: fsr_vis_d(:)    ! reflected direct beam vis solar
  real(r8), pointer :: fsr_nir_d(:)    ! reflected direct beam nir solar
  real(r8), pointer :: fsr_vis_i(:)    ! reflected diffuse vis solar radi
  real(r8), pointer :: fsr_nir_i(:)    ! reflected diffuse nir solar radi
  real(r8), pointer :: fsds_vis_d_ln(:) ! incident direct beam vis solar r
  real(r8), pointer :: fsds_nir_d_ln(:) ! incident direct beam nir solar r
  real(r8), pointer :: fsr_vis_d_ln(:) ! reflected direct beam vis solar
  real(r8), pointer :: fsr_nir_d_ln(:) ! reflected direct beam nir solar

```

A.33 Module TridiagonalMod

(Source File: TridiagonalMod.F90)

Tridiagonal matrix solution

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: Tridiagonal
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.33.1 Tridiagonal

INTERFACE:

```
subroutine Tridiagonal (lbc, ubc, lbj, ubj, jtop, numf, filter, &
                        a, b, c, r, u)
```

DESCRIPTION:

Tridiagonal matrix solution

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

ARGUMENTS:

```
implicit none
integer , intent(in)      :: lbc, ubc           ! lbinning and ubing co
integer , intent(in)      :: lbj, ubj           ! lbinning and ubing le
integer , intent(in)      :: jtop(lbc:ubc)      ! top level for each co
integer , intent(in)      :: numf              ! filter dimension
integer , intent(in)      :: filter(1:numf)     ! filter
real(r8), intent(in)      :: a(lbc:ubc, lbj:ubj) ! "a" left off diagonal
real(r8), intent(in)      :: b(lbc:ubc, lbj:ubj) ! "b" diagonal column f
real(r8), intent(in)      :: c(lbc:ubc, lbj:ubj) ! "c" right off diagona
real(r8), intent(in)      :: r(lbc:ubc, lbj:ubj) ! "r" forcing term of t
real(r8), intent(inout)   :: u(lbc:ubc, lbj:ubj) ! solution
```

CALLED FROM:

```
subroutine BiogeophysicsLake in module BiogeophysicsLakeMod
subroutine SoilTemperature in module SoilTemperatureMod
subroutine SoilWater in module HydrologyMod
```

REVISION HISTORY:

```
15 September 1999: Yongjiu Dai; Initial code
15 December 1999: Paul Houser and Jon Radakovich; F90 Revision
1 July 2003: Mariana Vertenstein; modified for vectorization
```


A.34 Module VOCEmissionMod

(Source File: VOCEmissionMod.F90)

Volatile organic compound emission

USES:

use abortutils, only: endrun

PUBLIC TYPES:

implicit none
save

PUBLIC MEMBER FUNCTIONS:

public :: VOCEmission

REVISION HISTORY:

Created by Mariana Vertenstein

A.34.1 VOCEmission

INTERFACE:

subroutine VOCEmission (lbp, ubp, num_nolakep, filter_nolakep)

DESCRIPTION:

Volatile organic compound emission

This code simulates volatile organic compound emissions following the algorithm presented in Guenther, A., 1999: Modeling Biogenic Volatile Organic Compound Emissions to the Atmosphere. In Reactive Hydrocarbons in the Atmosphere, Ch. 3
This model relies on the assumption that 90% of isoprene and monoterpene emissions originate from canopy foliage:

$$E = \epsilon * \gamma * \text{density} * \delta$$

The factor delta (longterm activity factor) applies to isoprene emission from deciduous plants only. We neglect this factor at the present time. This factor is discussed in Guenther (1997).

Subroutine written to operate at the patch level.

IN FINAL IMPLEMENTATION, REMEMBER:

1. may wish to call this routine only as freq. as rad. calculations
 2. may wish to place epsilon values directly in pft-physiology file
- Output: vocflx(nvoc) !VOC flux [ug C m⁻² h⁻¹]

USES:

use shr_kind_mod , only : r8 => shr_kind_r8
use clmtype
use shr_const_mod, only : SHR_CONST_RGAS

ARGUMENTS:

```

implicit none
integer, intent(in) :: lbp, ubp           ! pft bounds
integer, intent(in) :: num_nolakep        ! number of column non
integer, intent(in) :: filter_nolakep(num_nolakep) ! pft filter for non-1

```

CALLED FROM:

REVISION HISTORY:

Author: Sam Levis
 2/1/02, Peter Thornton: migration to new data structure

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: pgridcell(:)      ! gridcell index of corresponding p
integer , pointer :: ivt(:)            ! pft vegetation type for current
real(r8), pointer :: t_veg(:)          ! pft vegetation temperature (Kelvi
real(r8), pointer :: fsun(:)           ! sunlit fraction of canopy
real(r8), pointer :: elai(:)           ! one-sided leaf area index with bu
real(r8), pointer :: forc_solad(:, :)   ! direct beam radiation (visible on
real(r8), pointer :: forc_solai(:, :)   ! diffuse radiation      (visible on
real(r8), pointer :: sla(:)            ! ecophys constant - specific leaf

local pointers to original implicit out arrays
real(r8), pointer :: vocflx(:, :)      ! VOC flux [ug C m-2 h-1]
real(r8), pointer :: vocflx_tot(:)     ! VOC flux [ug C m-2 h-1]
real(r8), pointer :: vocflx_1(:)       ! VOC flux(1) [ug C m-2 h-1]
real(r8), pointer :: vocflx_2(:)       ! VOC flux(2) [ug C m-2 h-1]
real(r8), pointer :: vocflx_3(:)       ! VOC flux(3) [ug C m-2 h-1]
real(r8), pointer :: vocflx_4(:)       ! VOC flux(4) [ug C m-2 h-1]
real(r8), pointer :: vocflx_5(:)       ! VOC flux(5) [ug C m-2 h-1]

```

A.35 Module abortutils (Source File: abortutils.F90)

Abort the model for abnormal termination

REVISION HISTORY:

Author: CCM Core group

A.35.1 endrun

INTERFACE:

```
subroutine endrun(msg)
```

DESCRIPTION:

Abort the model for abnormal termination

USES:

```
#if (defined SPMD || defined COUP_CSM)
  use mpiinc
#endif
  use shr_sys_mod, only: shr_sys_flush
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in), optional :: msg    ! string to be printed
```

REVISION HISTORY:

Author: CCM Core group

A.36 Module accFldsMod

(Source File: accFldsMod.F90)

This module contains subroutines that initialize, update and extract the user-specified fields over user-defined intervals. Each interval and accumulation type is unique to each field processed.

Subroutine [initAccumFlds] defines the fields to be processed and the type of accumulation. Subroutine [updateAccumFlds] does the actual accumulation for a given field. Fields are accumulated by calls to subroutine [update_accum_field]. To accumulate a field, it must first be defined in subroutine [initAccumFlds] and then accumulated by calls to [updateAccumFlds].

Four types of accumulations are possible:

- o average over time interval
- o running mean over time interval
- o running accumulation over time interval

Time average fields are only valid at the end of the averaging interval.

Running means are valid once the length of the simulation exceeds the averaging interval. Accumulated fields are continuously accumulated.

The trigger value "-99999." resets the accumulation to zero.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils,    only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: initAccFlds      ! Initialization accumulator fields
public :: initAccClmtype   ! Initialize clmtype variables obtained from accu
public :: updateAccFlds    ! Update accumulator fields
```

REVISION HISTORY:

Created by M. Vertenstein 03/2003

A.36.1 initAccFlds()

INTERFACE:

```
subroutine initAccFlds()
```

DESCRIPTION:

Initializes accumulator and sets up array of accumulated fields

USES:

```

use accumulMod    , only : init_accum_field, print_accum_fields
use time_manager  , only : get_step_size, get_nstep
use shr_const_mod, only : SHR_CONST_CDAY, SHR_CONST_TKFRZ
use nanMod        , only : bigint

```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by M. Vertenstein 03/2003

A.36.2 updateAccFlds

INTERFACE:

```
subroutine updateAccFlds()
```

DESCRIPTION:

Update and/or extract accumulated fields

USES:

```

use clmtype
use decompMod    , only : get_proc_bounds
use clm_varcon   , only : spval
use pftvarcon    , only : pftpar
use shr_const_mod, only : SHR_CONST_CDAY, SHR_CONST_TKFRZ
use time_manager , only : get_step_size, get_nstep, is_end_curr_day, get_
use accumulMod   , only : update_accum_field, extract_accum_field

```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by M. Vertenstein 03/2003

LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: itype(:)           ! pft vegetation
integer , pointer :: pgridcell(:)       ! index into gridcell level quan
real(r8), pointer :: forc_t(:)          ! atmospheric temperature (Kelvi
real(r8), pointer :: forc_rain(:)       ! rain rate [mm/s]
real(r8), pointer :: forc_snow(:)       ! snow rate [mm/s]
real(r8), pointer :: frmf(:)            ! leaf maintenance respiration
real(r8), pointer :: fpsn(:)            ! photosynthesis (umol CO2 /m**2
real(r8), pointer :: t_ref2m(:)         ! 2 m height surface air tempera
local pointers to implicit out arguments
real(r8), pointer :: t_ref2m_min(:)     ! daily minimum of average 2 m h
real(r8), pointer :: t_ref2m_max(:)     ! daily maximum of average 2 m h

```

```

real(r8), pointer :: t_ref2m_min_inst(:) ! instantaneous daily min of ave
real(r8), pointer :: t_ref2m_max_inst(:) ! instantaneous daily max of ave
real(r8), pointer :: t10(:)             ! 10-day running mean of the 2 m
real(r8), pointer :: t_mo(:)            ! 30-day average temperature (Ke
real(r8), pointer :: t_mo_min(:)        ! annual min of t_mo (Kelvin)
real(r8), pointer :: fnpsn10(:)         ! 10-day running mean net photos
real(r8), pointer :: prec365(:)         ! 365-day running mean of tot. p
real(r8), pointer :: agdd0(:)           ! accumulated growing degree day
real(r8), pointer :: agdd5(:)           ! accumulated growing degree day
real(r8), pointer :: agddtw(:)          ! accumulated growing degree day
real(r8), pointer :: agdd(:)            ! accumulated growing degree day

```

A.36.3 initAccClmtype

INTERFACE:

```
subroutine initAccClmtype
```

DESCRIPTION:

Initialize clmtype variables that are associated with time accumulated fields. This routine is called in an initial run at nstep=0 for cam and csm mode and at nstep=1 for offline mode. This routine is also always called for a restart run and therefore must be called after the restart file is read in and the accumulated fields are obtained.

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use decompMod    , only : get_proc_bounds, get_proc_global
use accumulMod   , only : extract_accum_field
use time_manager, only : get_nstep
use clm_varctl   , only : nsrest
use clm_varcon   , only : spval

```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

REVISION HISTORY:

Created by Mariana Vertenstein

LOCAL VARIABLES:

```

local pointers to implicit out arguments
real(r8), pointer :: t_ref2m_min(:) ! daily minimum of average 2 m h
real(r8), pointer :: t_ref2m_max(:) ! daily maximum of average 2 m h

```

```

real(r8), pointer :: t_ref2m_min_inst(:) ! instantaneous daily min of ave
real(r8), pointer :: t_ref2m_max_inst(:) ! instantaneous daily max of ave
real(r8), pointer :: t10(:)             ! 10-day running mean of the 2 m
real(r8), pointer :: t_mo(:)            ! 30-day average temperature (Ke
real(r8), pointer :: fnpsn10(:)         ! 10-day running mean net photos
real(r8), pointer :: prec365(:)         ! 365-day running mean of tot. p
real(r8), pointer :: agdd0(:)           ! accumulated growing degree day
real(r8), pointer :: agdd5(:)           ! accumulated growing degree day
real(r8), pointer :: agddtw(:)          ! accumulated growing degree day
real(r8), pointer :: agdd(:)            ! accumulated growing degree day

```

LOCAL VARIABLES:

A.37 Module accumulMod

(Source File: accumulMod.F90)

This module contains generic subroutines that can be used to define, accumulate and extract user-specified fields over user-defined intervals. Each interval and accumulation type is unique to each field processed.

Subroutine [init_accumulator] defines the values of the accumulated field data structure. Subroutine [update_accum_field] does the actual accumulation for a given field.

Four types of accumulations are possible:

- Average over time interval. Time average fields are only valid at the end of the averaging interval.
- Running mean over time interval. Running means are valid once the length of the simulation exceeds the
- Running accumulation over time interval. Accumulated fields are continuously accumulated. The trigger value "-99999." resets the accumulation to zero.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils,    only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: init_accum_field      ! Initialize an accumulator field
public :: print_accum_fields    ! Print info about accumulator fields
public :: extract_accum_field   ! Extracts the current value of an accumul
interface extract_accum_field
  module procedure extract_accum_field_sl ! Extract current val of single-
  module procedure extract_accum_field_ml ! Extract current val of multi-l
end interface
public :: update_accum_field
interface update_accum_field      ! Updates the current value of a
  module procedure update_accum_field_sl ! Update single-level accumulato
  module procedure update_accum_field_ml ! Update multi-level accumulator
end interface
public :: restart_accum          ! Write/read restart of accumula
```

REVISION HISTORY:

```
Created by Sam Levis
Updated to clm2.1 data structures by Mariana Vertenstein
Updated to include all subgrid type and multilevel fields, M. Vertenstein 0
```

A.37.1 init_accum_field

INTERFACE:

```
subroutine init_accum_field (name, units, desc, &
    accum_type, accum_period, numlev, subgrid_type, init_value)
```

DESCRIPTION:

```
Initialize accumulation fields. This subroutine sets:
o name of accumulated field
o units of accumulated field
o accumulation type of accumulated field
o description of accumulated fields: accdes
o accumulation period for accumulated field (in iterations)
o initial value of accumulated field
```

USES:

```
use shr_const_mod, only: SHR_CONST_CDAY
use time_manager, only : get_step_size
use decompMod, only : get_proc_bounds, get_proc_global
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name           !field name
character(len=*), intent(in) :: units          !field units
character(len=*), intent(in) :: desc           !field description
character(len=*), intent(in) :: accum_type     !field type: tavg, runm, r
integer , intent(in) :: accum_period           !field accumulation period
character(len=*), intent(in) :: subgrid_type  !["gridcell","landunit","c
integer , intent(in) :: numlev                 !number of vertical levels
real(r8), intent(in) :: init_value             !field initial or reset va
```

REVISION HISTORY:

Created by Mariana Vertenstein 03/2003

A.37.2 print_accum_fields

INTERFACE:

```
subroutine print_accum_fields()
```

DESCRIPTION:

```
Diagnostic printout of accumulated fields
```

USES:

```
use spmdMod, only : masterproc
use nanMod, only : bigint
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by Mariana Vertenstein 03/2003

A.37.3 extract_accum_field_sl

INTERFACE:

```
subroutine extract_accum_field_sl (name, field, nstep)
```

DESCRIPTION:

Extract single-level accumulated field.

This routine extracts the field values from the multi-level accumulation field. It extracts the current value except if the field type is a time average. In this case, an absurd value is assigned to indicate the time average is not yet valid.

USES:

```
use clm_varcon, only : spval
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name      !field name
real(r8), pointer, dimension(:) :: field !field values for current time s
integer, intent(in) :: nstep              !timestep index
```

REVISION HISTORY:

Created by Sam Levis

Updated to clm2.1 data structures by Mariana Vertenstein

Updated to include all subgrid type and multilevel fields, Mariana Vertenst

A.37.4 extract_accum_field_ml

INTERFACE:

```
subroutine extract_accum_field_ml (name, field, nstep)
```

DESCRIPTION:

Extract mutli-level accumulated field.

This routine extracts the field values from the multi-level accumulation field. It extracts the current value except if the field type is a time average. In this case, an absurd value is assigned to indicate the time average is not yet valid.

USES:

```
use clm_varcon, only : spval
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name      !field name
real(r8), pointer, dimension(:, :) :: field !field values for current time
integer, intent(in) :: nstep              !timestep index
```


REVISION HISTORY:

Created by Sam Levis
 Updated to clm2.1 data structures by Mariana Vertenstein
 Updated to include all subgrid type and multilevel fields, M. Vertenstein 0

A.37.5 `update_accum_field_sl`

INTERFACE:

```
subroutine update_accum_field_sl (name, field, nstep)
```

DESCRIPTION:

Accumulate single level field over specified time interval.
 The appropriate field is accumulated in the array [accval].

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name      !field name
real(r8), pointer, dimension(:) :: field !field values for current time s
integer , intent(in) :: nstep             !time step index
```

REVISION HISTORY:

Created by Sam Levis
 Updated to clm2.1 data structures by Mariana Vertenstein
 Updated to include all subgrid type and multilevel fields by M. Vertenstein

A.37.6 `update_accum_field_ml`

INTERFACE:

```
subroutine update_accum_field_ml (name, field, nstep)
```

DESCRIPTION:

Accumulate multi level field over specified time interval.

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name      !field name
real(r8), pointer, dimension(:,:) :: field !field values for current time
integer , intent(in) :: nstep             !time step index
```

REVISION HISTORY:

Created by Sam Levis
 Updated to clm2.1 data structures by Mariana Vertenstein
 Updated to include all subgrid type and multilevel fields by M. Vertenstein

A.37.7 restart_accum

INTERFACE:

```
subroutine restart_accum (nio, flag)
```

DESCRIPTION:

Read/write accumulation restart data

USES:

```
use decompMod, only : get_proc_bounds, get_proc_global
use iobinary
#ifdef SPMD
use spmdMod, only : mpicom, MPI_CHARACTER, MPI_INTEGER
#endif
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: nio          !restart unit
character(len=*), intent(in) :: flag !'read' or 'write'
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.38 Module areaMod

(Source File: areaMod.F90)

Area averaging routines

These routines are used for area-average mapping of a field from one grid to another.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varcon, only : re
use shr_const_mod, only : SHR_CONST_PI
use abortutils,    only: endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: areaini          ! area averaging initialization
public :: areaave          ! area averaging of field from input to output grid
public :: areaini_point    ! area averaging initialization for single grid cell
interface celledge
  module procedure celledge_regional !Southern and western edges of grid
  module procedure celledge_global  !Southern and western edges of grid
end interface
interface cellarea
  module procedure cellarea_regional !Area of grid cells (square kilometers)
  module procedure cellarea_global   !Area of grid cells (square kilometers)
end interface
public :: mkmxovr          ! find maximum number of overlapping cells between
```

REVISION HISTORY:

Created by Sam Levis

Updated to clm2.1 data structures by Mariana Vertenstein

A.38.1 areaini

INTERFACE:

```
subroutine areaini (nlon_i , nlat_i, numlon_i, lon_i, lat_i, area_i, &
                  mask_i , nlon_o , nlat_o, numlon_o, lon_o, lat_o, &
                  area_o, fland_o, mx_ovr , novr_i2o, iovr_i2o, &
                  jovr_i2o, wovr_i2o )
```

DESCRIPTION:

area averaging initialization

This subroutine is used for area-average mapping of a field from one grid to another.

areaini - initializes indices and weights for area-averaging from input grid to output grid
 areamap - called by areaini: finds indices and weights
 areaovr - called by areamap: finds if cells overlap and area of overlap
 areaave - does area-averaging from input grid to output grid

To map from one grid to another, must first call areaini to build the indices and weights (iovr_i2o, jovr_i2o, wovr_i2o). Then must call areaave to get new field on output grid.

Not all grid cells on the input grid will be used in the area-averaging of a field to the output grid. Only input grid cells with [mask_i] = 1 contribute to output grid cell average. If [mask_i] = 0, input grid cell does not contribute to output grid cell. This distinction is not usually required for atm → land mapping, because all cells on the atm grid have data. But when going from land → atm, only land grid cells have data. Non-land grid cells on surface grid do not have data. So if output grid cell overlaps with land and non-land cells (input grid), can only use land grid cells when computing area-average.

o Input and output grids can be ANY resolution BUT:

- a. Grid orientation -- Grids can be oriented south to north (i.e. cell(lat+1) is north of cell(lat)) or from north to south (i.e. cell(lat+1) is south of cell(lat)). Both grids must be oriented from west to east, i.e., cell(lon+1) must be east of cell(lon)
- b. Grid domain -- Grids do not have to be global. Both grids are defined by their north, east, south, and west edges (edge_i and edge_o in this order, i.e., edge_i(1) is north and edge_i(4) is west). For partial grids, northern and southern edges are any latitude between 90 (North Pole) and -90 (South Pole). Western and eastern edges are any longitude between -180 and 180, with longitudes west of Greenwich negative. For global grids, northern and southern edges are 90 (North Pole) and -90 (South Pole). The grids do not have to start at the same longitude, i.e., one grid can start at Dateline and go east; the other grid can start at Greenwich and go east. Longitudes for the western edge of the cells must increase continuously and span 360 degrees. Examples

	West edge	East edge
Dateline	-180 to 180	(negative W of Greenwi
Greenwich (centered):	0 - dx/2 to 360 - dx/2	

- c. Both grids can have variable number of longitude points for each latitude strip. However, the western edge of the first point in each latitude must be the same for all latitudes. Likewise, for the eastern edge of the last point. That is, each latitude strip must span the same longitudes, but the number of points to do this can be differ
 - d. One grid can be a sub-set (i.e., smaller domain) than the other grid. In this way, an atmospheric dataset for the entire globe can be used in a simulation for a region 30N to 50N and 130W to 70W -- the code will extract the appropriate data. The two grids do not have to be the same resolution. Area-averaging will work for full => partial grid but obviously will not work for partial => full grid.
- o Field values fld_i on an input grid with dimensions nlon_i and nlat_i => field values fld_o on an output grid with dimensions nlon_o and nlat_o as
- fld_o(io,jo) =
- fld_i(i_ovr(io,jo, 1),j_ovr(io,jo, 1)) * w_ovr(io,jo, 1)

```

... + ... +
fld_i(i_ovr(io,jo,mx_ovr),j_ovr(io,jo,mx_ovr)) * w_ovr(io,jo,mx_ovr)
o Error checks:
  Overlap weights of input cells sum to 1 for each output cell.
  Global sum of dummy field is conserved for input => output area-average.

```

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlon_i           !input  grid: max n
integer , intent(in) :: nlat_i           !input  grid: numbe
integer , intent(in) :: numlon_i(nlat_i) !input  grid: numbe
real(r8), intent(inout) :: lon_i(nlon_i+1,nlat_i) !input grid: longit
real(r8), intent(in) :: lat_i(nlat_i+1) !input grid: latitu
real(r8), intent(in) :: area_i(nlon_i,nlat_i) !input grid: cell a
real(r8), intent(in) :: mask_i(nlon_i,nlat_i) !input  grid: mask
integer , intent(in) :: nlon_o           !output grid: max n
integer , intent(in) :: nlat_o           !output grid: numbe
integer , intent(in) :: numlon_o(nlat_o) !output grid: numbe
real(r8), intent(in) :: lon_o(nlon_o+1,nlat_o) !output grid: longi
real(r8), intent(in) :: lat_o(nlat_o+1) !output grid: latit
real(r8), intent(in) :: area_o(nlon_o,nlat_o) !output grid: cell
real(r8), intent(in) :: fland_o(nlon_o,nlat_o) !output grid: fract
integer , intent(in) :: mx_ovr           !maximum number of
integer , intent(out):: novr_i2o(nlon_o,nlat_o) !number of overlapp
integer , intent(out):: iovr_i2o(nlon_o,nlat_o,mx_ovr)!lon index of overl
integer , intent(out):: jovr_i2o(nlon_o,nlat_o,mx_ovr)!lat index of overl
real(r8), intent(out):: wovr_i2o(nlon_o,nlat_o,mx_ovr)!weight    of overl

```

REVISION HISTORY:

Created by Gordon Bonan

A.38.2 areaave

INTERFACE:

```

subroutine areaave (nlat_i , nlon_i , numlon_i, fld_i , &
                   nlat_o , nlon_o , numlon_o, fld_o , &
                   i_ovr  , j_ovr  , w_ovr   , nmax )

```

DESCRIPTION:

Area averaging of field from input to output grids

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlat_i           !input grid : number of
integer , intent(in) :: nlon_i           !input grid : max number
integer , intent(in) :: numlon_i(nlat_i) !input grid : number of
real(r8), intent(in) :: fld_i(nlon_i,nlat_i) !input grid : field
integer , intent(in) :: nlat_o           !output grid: number of

```

```

integer ,intent(in) :: nlon_o           !output grid: max number
integer ,intent(in) :: numlon_o(nlat_o) !output grid: number of
real(r8),intent(out):: fld_o(nlon_o,nlat_o) !field for output grid
integer ,intent(in) :: nmax           !input grid : max number
integer ,intent(in) :: i_ovr(nlon_o,nlat_o,nmax) !lon index, overlapping
integer ,intent(in) :: j_ovr(nlon_o,nlat_o,nmax) !lat index, overlapping
real(r8),intent(in) :: w_ovr(nlon_o,nlat_o,nmax) !overlap weights for inp

```

REVISION HISTORY:

Created by Gordon Bonan

A.38.3 areamap

INTERFACE:

```

subroutine areamap (nlon_i  , nlat_i  , nlon_o , nlat_o , &
                   lon_i    , lat_i    , lon_o  , lat_o  , &
                   numlon_i , numlon_o , mask_i , mx_ovr , &
                   n_ovr    , i_ovr    , j_ovr  , w_ovr  , &
                   fland_o  , area_o   )

```

DESCRIPTION:

Weights and indices for area of overlap between grids
 Get indices and weights for area-averaging between input and output grids.
 For each output grid cell find:

- o number of input grid cells that overlap with output grid cell (n_ovr)
- o longitude index (1 <= i_ovr <= nlon_i) of the overlapping input grid c
- o latitude index (1 <= j_ovr <= nlat_i) of the overlapping input grid c
- o fractional overlap of input grid cell (w_ovr)

so that for
 field values fld_i on an input grid with dimensions nlon_i and nlat_i
 field values fld_o on an output grid with dimensions nlon_o and nlat_o are
 fld_o(io,jo) =
 fld_i(i_ovr(io,jo, 1),j_ovr(io,jo, 1)) * w_ovr(io,jo, 1) +
 ... + ... +
 fld_i(i_ovr(io,jo,mx_ovr),j_ovr(io,jo,mx_ovr)) * w_ovr(io,jo,mx_ovr)
 Note: mx_ovr is some number greater than n_ovr. Weights of zero are
 used for the excess points

ARGUMENTS:

```

implicit none
integer ,intent(in) :: nlon_i           !input grid : max numbe
integer ,intent(in) :: nlat_i           !input grid : number of
integer ,intent(in) :: numlon_i(nlat_i) !input grid : number lo
real(r8),intent(inout) :: lon_i(nlon_i+1,nlat_i) !input grid : cell long
real(r8),intent(in) :: lat_i(nlat_i+1) !input grid : cell lati
real(r8),intent(in) :: mask_i(nlon_i,nlat_i) !input grid : mask (0,
integer ,intent(in) :: nlon_o           !output grid: max numbe
integer ,intent(in) :: nlat_o           !output grid: number of

```

```

integer ,intent(in) :: numlon_o(nlat_o)           !output grid: number lo
real(r8),intent(in) :: lon_o(nlon_o+1,nlat_o)     !output grid: cell long
real(r8),intent(in) :: lat_o(nlat_o+1)           !output grid: cell lati
real(r8),intent(in) :: fland_o(nlon_o,nlat_o)     !output grid: fraction
real(r8),intent(in) :: area_o(nlon_o,nlat_o)      !output grid: cell area
integer ,intent(in) :: mx_ovr                     !max num input cells th
integer ,intent(out):: n_ovr(nlon_o,nlat_o)        !number of overlapping
integer ,intent(out):: i_ovr(nlon_o,nlat_o,mx_ovr)!lon index, overlapping
integer ,intent(out):: j_ovr(nlon_o,nlat_o,mx_ovr)!lat index, overlapping
real(r8),intent(out):: w_ovr(nlon_o,nlat_o,mx_ovr)!overlap weights for in

```

REVISION HISTORY:

Created by Gordon Bonan

A.38.4 areaovr

INTERFACE:

```

subroutine areaovr (nlon_i , nlat_i , numlon_i , lon_i , lat_i , &
                   nlon_o , nlat_o , numlon_o , lon_o , lat_o , &
                   mx_ovr , n_ovr , i_ovr , j_ovr , w_ovr )

```

DESCRIPTION:

Find area of overlap between grid cells

For each output grid cell: find overlapping input grid cell and area of input grid cell that overlaps with output grid cell. Cells overlap if:

southern edge of input grid < northern edge of output grid AND

northern edge of input grid > southern edge of output grid

western edge of input grid < eastern edge of output grid AND

eastern edge of input grid > western edge of output grid

```

lon_o(io,jo)      lon_o(io+1,jo)
|                 |
|                 | lat_o(jo+1)
|                 |
|                 |
xxxxxxxxxxxxxxxxx lat_i(ji+1)
x               | x               |
x input   |   x   output         |
x cell    |   x   cell           |
x ii,ji   |   x   io,jo          |
x         |   x                 |
x         |   x                 |
x         |-----x----- lat_o(jo )
x         |
xxxxxxxxxxxxxxxxx lat_i(ji )
x         |
lon_i(ii,ji) lon_i(ii+1,ji)

```

The above diagram assumes both grids are oriented South to North. Other combinations of North to South and South to North grids are possible:

Input Grid Output Grid

```

(1)  S to N      S to N
(2)  N to S      N to S
(3)  S to N      N to S
(4)  N to S      S to N

```

The code has been modified to allow for North to South grids. Verification that these changes work are:

- o (1) and (4) give same results for output grid
- o (2) and (3) give same results for output grid
- o (2) and (4) give same results for output grid when output grid invert

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlon_i           !input grid : max number o
integer , intent(in) :: nlat_i           !input grid : number of la
integer , intent(in) :: numlon_i(nlat_i) !input grid : number of lo
real(r8), intent(in) :: lon_i(nlon_i+1,nlat_i) !input grid : cell longitu
real(r8), intent(in) :: lat_i(nlat_i+1)      !input grid : cell latitud
integer , intent(in) :: nlon_o           !output grid: max number o
integer , intent(in) :: nlat_o           !output grid: number of la
integer , intent(in) :: numlon_o(nlat_o)    !output grid: number of lo
real(r8), intent(in) :: lon_o(nlon_o+1,nlat_o) !output grid: cell longitu
real(r8), intent(in) :: lat_o(nlat_o+1)      !output grid: cell latitud
integer , intent(in) :: mx_ovr           !maximum number of overlap
integer , intent(inout) :: n_ovr(nlon_o,nlat_o) !number of overlap
integer , intent(inout) :: i_ovr(nlon_o,nlat_o,mx_ovr) !lon index, overla
integer , intent(inout) :: j_ovr(nlon_o,nlat_o,mx_ovr) !lat index, overla
real(r8), intent(inout) :: w_ovr(nlon_o,nlat_o,mx_ovr) !area of overlap f

```

REVISION HISTORY:

Created by Gordon Bonan

A.38.5 cellarea_regional

INTERFACE:

```

subroutine cellarea_regional (nlat ,nlon ,numlon, lats ,lonw , &
                             edgen, edgeee, edges , edgew, area)

```

DESCRIPTION:

Area of grid cells (square kilometers) - regional grid
(can become global as special case)

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlat           !dimension: number of latitude
integer , intent(in) :: nlon           !dimension: number of longitud
integer , intent(in) :: numlon(nlat)    !number of grid cells per lati
real(r8), intent(in) :: edgen           !northern edge of grid (degree
real(r8), intent(in) :: edges          !southern edge of grid (degree

```

```

real(r8), intent(in) :: edgew           !western edge of grid (degrees
real(r8), intent(in) :: edgee           !eastern edge of grid (degrees
real(r8), intent(in) :: lats(nlat+1)     !grid cell latitude, southern
real(r8), intent(in) :: lonw(nlon+1,nlat) !grid cell longitude, western
real(r8), intent(out):: area(nlon,nlat)   !cell area (km**2)

```

REVISION HISTORY:

Created by Mariana Vertenstein

A.38.6 cellarea_global

INTERFACE:

```

subroutine cellarea_global (nlat , nlon, numlon, lats, lonw, area)

```

DESCRIPTION:

Area of grid cells (square kilometers)- global grid

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlat           !dimension: number of latitude p
integer , intent(in) :: nlon           !dimension: number of longitude
integer , intent(in) :: numlon(nlat)   !number of grid cells per latitu
real(r8), intent(in) :: lats(nlat+1)   !grid cell latitude, southern ed
real(r8), intent(in) :: lonw(nlon+1,nlat)!grid cell longitude, western ed
real(r8), intent(out):: area(nlon,nlat) !cell area (km**2)

```

REVISION HISTORY:

Created by Mariana Vertenstein

A.38.7 celledge_regional

INTERFACE:

```

subroutine celledge_regional (nlat , nlon , numlon , longxy , &
                             latixy , edgen , edgee , edges , &
                             edgew , lats , lonw )

```

DESCRIPTION:

Southern and western edges of grid cells - regional grid
 (can become global as special case)
 Latitudes -- southern/northern edges for each latitude strip.
 For grids oriented South to North, the southern
 and northern edges of latitude strip [j] are:
 southern = lats(j)

```

        northern = lats(j+1)
For grids oriented North to South: the southern
and northern edges of latitude strip [j] are:
        northern = lats(j )
        southern = lats(j+1)
In both cases, [lats] must be dimensioned lats(lat+1)
Longitudes -- western edges. Longitudes for the western edge of the
cells must increase continuously and span 360 degrees. Assume that
grid starts at Dateline with western edge on Dateline Western edges
correspond to [longxy] (longitude at center of cell) and range from
-180 to 180 with negative longitudes west of Greenwich.
Partial grids that do not span 360 degrees are allowed so long as they
have the convention of Grid 1 with
        western edge of grid: >= -180 and < 180
        eastern edge of grid: > western edge and <= 180
[lonw] must be dimensioned lonw(lon+1,lat) because each latitude
strip can have variable longitudinal resolution

```

ARGUMENTS:

```

implicit none
integer , intent(in) :: nlat           !dimension: number of latitude
integer , intent(in) :: nlon           !dimension: number of longitude
integer , intent(in) :: numlon(nlat)   !number of grid cells per latit
real(r8), intent(in) :: longxy(nlon,nlat) !longitude at center of grid ce
real(r8), intent(in) :: latixy(nlon,nlat) !latitude at center of grid cel
real(r8), intent(in) :: edgen          !northern edge of grid (degrees)
real(r8), intent(in) :: edgee          !eastern edge of grid (degrees)
real(r8), intent(in) :: edges          !southern edge of grid (degrees)
real(r8), intent(in) :: edgew          !western edge of grid (degrees)
real(r8), intent(out):: lats(nlat+1)    !grid cell latitude, southern e
real(r8), intent(out):: lonw(nlon+1,nlat) !grid cell longitude, western e

```

REVISION HISTORY:

Created by Mariana Vertenstein

A.38.8 celledge_global

INTERFACE:

```

subroutine celledge_global (nlat, nlon, numlon, longxy, latixy, &
                           lats, lonw )

```

DESCRIPTION:

Southern and western edges of grid cells - global grid
 Latitudes -- southern/northern edges for each latitude strip.
 For grids oriented South to North, the southern
 and northern edges of latitude strip [j] are:

```

        southern = lats(j )
        northern = lats(j+1)

```

For grids oriented North to South: the southern
and northern edges of latitude strip [j] are:

```
northern = lats(j )
southern = lats(j+1)
```

In both cases, [lats] must be dimensioned lats(lat+1)
Longitudes -- western edges. Longitudes for the western edge of the
cells must increase continuously and span 360 degrees.

ARGUMENTS:

```
implicit none
integer , intent(in) :: nlat           !dimension: number of latitude
integer , intent(in) :: nlon           !dimension: number of longitud
integer , intent(in) :: numlon(nlat)   !number of grid cells per lati
real(r8), intent(in) :: longxy(nlon,nlat) !longitude at center of grid c
real(r8), intent(in) :: latixy(nlon,nlat) !latitude at center of grid ce
real(r8), intent(out):: lats(nlat+1)    !grid cell latitude, southern
real(r8), intent(out):: lonw(nlon+1,nlat) !grid cell longitude, western
```

REVISION HISTORY:

Created by Mariana Vertenstein

A.38.9 areaini_point

INTERFACE:

```
subroutine areaini_point (io      , jo      , nlon_i  , nlat_i  , &
                        numlon_i, lon_i    , lon_i_offset, lat_i    , &
                        area_i   , mask_i   , nlon_o   , nlat_o   , &
                        numlon_o, lon_o    , lat_o     , area_o    , &
                        fland_o  , novr_i2o, iovr_i2o, jovr_i2o , &
                        wovr_i2o, maxovr)
```

DESCRIPTION:

area averaging initialization

This subroutine is used for area-average mapping of a field from one
grid to another.

areaini_point - initializes indices and weights for area-averaging from
input grid to output grid

areamap_point - called by areaini_point: finds indices and weights

areaovr_point - called by areamap_point: finds if cells overlap and are
To map from one grid to another, must first call areaini to build
the indices and weights (iovr_i2o, jovr_i2o, wovr_i2o). Then must
call areaave to get new field on output grid.

Not all grid cells on the input grid will be used in the area-averaging
of a field to the output grid. Only input grid cells with [mask_i] = 1
contribute to output grid cell average. If [mask_i] = 0, input grid cell
does not contribute to output grid cell. This distinction is not usually
required for atm -> land mapping, because all cells on the atm grid have
data. But when going from land -> atm, only land grid cells have data.

Non-land grid cells on surface grid do not have data. So if output grid cell overlaps with land and non-land cells (input grid), can only use land grid cells when computing area-average.

o Input and output grids can be ANY resolution BUT:

- a. Grid orientation -- Grids can be oriented south to north (i.e. cell(lat+1) is north of cell(lat)) or from north to south (i.e. cell(lat+1) is south of cell(lat)). Both grids must be oriented from west to east, i.e., cell(lon+1) must be east of cell(lon)
- b. Grid domain -- Grids do not have to be global. Both grids are defined by their north, east, south, and west edges (edge_i and edge_o in this order, i.e., edge_i(1) is north and edge_i(4) is west). For partial grids, northern and southern edges are any latitude between 90 (North Pole) and -90 (South Pole). Western and eastern edges are any longitude between -180 and 180, with longitudes west of Greenwich negative.

For global grids, northern and southern edges are 90 (North Pole) and -90 (South Pole). The grids do not have to start at the same longitude, i.e., one grid can start at Dateline and go east; the other grid can start at Greenwich and go east. Longitudes for the western edge of the cells must increase continuously and span 360 degrees. Examples

	West edge	East edge
Dateline	-180 to 180	(negative W of Greenwich (centered): 0 - dx/2 to 360 - dx/2)

- | | | |
|----------|-------------|--|
| | ----- | |
| Dateline | -180 to 180 | (negative W of Greenwich (centered): 0 - dx/2 to 360 - dx/2) |
- c. Both grids can have variable number of longitude points for each latitude strip. However, the western edge of the first point in each latitude must be the same for all latitudes. Likewise, for the eastern edge of the last point. That is, each latitude strip must span the same longitudes, but the number of points to do this can be differ
 - d. One grid can be a sub-set (i.e., smaller domain) than the other grid. In this way, an atmospheric dataset for the entire globe can be used in a simulation for a region 30N to 50N and 130W to 70W -- the code will extract the appropriate data. The two grids do not have to be the same resolution. Area-averaging will work for full => partial grid but obviously will not work for partial => full grid.
- o Field values fld_i on an input grid with dimensions nlon_i and nlat_i => field values fld_o on an output grid with dimensions nlon_o and nlat_o as
- ```
fld_o(io,jo) =
 fld_i(i_ovr(io,jo, 1),j_ovr(io,jo, 1)) * w_ovr(io,jo, 1)
 ... + ... +
 fld_i(i_ovr(io,jo,maxovr),j_ovr(io,jo,maxovr)) * w_ovr(io,jo,maxovr)
```
- o Error checks:
- Overlap weights of input cells sum to 1 for each output cell.
- Global sum of dummy field is conserved for input => output area-average.

#### ARGUMENTS:

```
implicit none
integer , intent(in) :: io !output grid longitude
integer , intent(in) :: jo !output grid latitude i
integer , intent(in) :: nlon_i !input grid: max numbe
integer , intent(in) :: nlat_i !input grid: number of
integer , intent(in) :: numlon_i(nlat_i) !input grid: number lo
real(r8) , intent(in) :: lon_i(nlon_i+1,nlat_i) !input grid: longitude,
```

```

real(r8), intent(in) :: lon_i_offset(nlon_i+1,nlat_i) !input grid : ce
real(r8), intent(in) :: lat_i(nlat_i+1) !input grid: latitude,
real(r8), intent(in) :: area_i(nlon_i,nlat_i) !input grid: cell area
real(r8), intent(in) :: mask_i(nlon_i,nlat_i) !input grid: mask (0,
integer , intent(in) :: nlon_o !output grid: max numbe
integer , intent(in) :: nlat_o !output grid: number of
integer , intent(in) :: numlon_o(nlat_o) !output grid: number lo
real(r8), intent(in) :: lon_o(nlon_o+1,nlat_o) !output grid: longitude
real(r8), intent(in) :: lat_o(nlat_o+1) !output grid: latitude,
real(r8), intent(in) :: area_o !output grid: cell area
real(r8), intent(in) :: fland_o !output grid: fraction
integer , intent(out) :: novr_i2o !number of overlapping
integer , intent(in) :: maxovr !maximum number of over
integer , intent(out) :: iovr_i2o(maxovr) !lon index of overlap i
integer , intent(out) :: jovr_i2o(maxovr) !lat index of overlap i
real(r8), intent(out) :: wovr_i2o(maxovr) !weight of overlap i

```

## REVISION HISTORY:

Created by Gordon Bonan

## A.38.10 areamap\_point

## INTERFACE:

```

subroutine areamap_point(io , jo , nlon_i , nlat_i , nlon_o , &
 nlat_o , numlon_i , lon_i , lat_i , mask_i , &
 lon_o , lat_o , fland_o , area_o , n_ovr , &
 i_ovr , j_ovr , w_ovr , lon_i_offset, &
 maxovr)

```

## DESCRIPTION:

weights and indices for area of overlap between grids  
 Get indices and weights for area-averaging between input and output grids.  
 For each output grid cell find:  
   o number of input grid cells that overlap with output grid cell (n\_ovr)  
   o longitude index (1 <= i\_ovr <= nlon\_i) of the overlapping input grid c  
   o latitude index (1 <= j\_ovr <= nlat\_i) of the overlapping input grid c  
   o fractional overlap of input grid cell (w\_ovr)  
 so that for  
 field values fld\_i on an input grid with dimensions nlon\_i and nlat\_i  
 field values fld\_o on an output grid with dimensions nlon\_o and nlat\_o are  
 fld\_o(io,jo) =  
 fld\_i(i\_ovr(io,jo, 1),j\_ovr(io,jo, 1)) \* w\_ovr(io,jo, 1) +  
   ... + ... +  
 fld\_i(i\_ovr(io,jo,maxovr),j\_ovr(io,jo,maxovr)) \* w\_ovr(io,jo,maxovr)  
 Note: maxovr is some number greater than n\_ovr. Weights of zero are  
 used for the excess points

## ARGUMENTS:



```

 x ii,ji | x io,jo |
 x | x |
 x |-----x----- lat_o(jo)
 x | x |
 xxxxxxxxxxxxxxxx lat_i(ji)
 x | x |
lon_i(ii,ji) lon_i(ii+1,ji)

```

The above diagram assumes both grids are oriented South to North. Other combinations of North to South and South to North grids are possible:

|     | Input Grid | Output Grid |
|-----|------------|-------------|
| (1) | S to N     | S to N      |
| (2) | N to S     | N to S      |
| (3) | S to N     | N to S      |
| (4) | N to S     | S to N      |

The code has been modified to allow for North to South grids. Verification that these changes work are:

- o (1) and (4) give same results for output grid
- o (2) and (3) give same results for output grid
- o (2) and (4) give same results for output grid when output grid invert

#### ARGUMENTS:

```

implicit none
integer , intent(in) :: io !output grid lon index
integer , intent(in) :: jo !output grid lat index
integer , intent(in) :: nlon_i !input grid : max number o
integer , intent(in) :: nlat_i !input grid : number of la
integer , intent(in) :: numlon_i(nlat_i) !input grid : number of lo
real(r8), intent(in) :: lon_i(nlon_i+1,nlat_i) !input grid : cell longitu
real(r8), intent(in) :: lat_i(nlat_i+1) !input grid : cell latitud
integer , intent(in) :: nlon_o !output grid: max number o
integer , intent(in) :: nlat_o !output grid: number of la
real(r8), intent(in) :: lon_o(nlon_o+1,nlat_o) !output grid: cell longitu
real(r8), intent(in) :: lat_o(nlat_o+1) !output grid: cell latitud
integer , intent(inout) :: n_ovr !number of overlapping inp
integer , intent(in) :: maxovr !maximum number of overlap
integer , intent(inout) :: i_ovr(maxovr) !lon index, overlapping in
integer , intent(inout) :: j_ovr(maxovr) !lat index, overlapping in
real(r8), intent(inout) :: w_ovr(maxovr) !area of overlap for input

```

#### REVISION HISTORY:

Created by Gordon Bonan

### A.38.12 mkmxovr

#### INTERFACE:

```

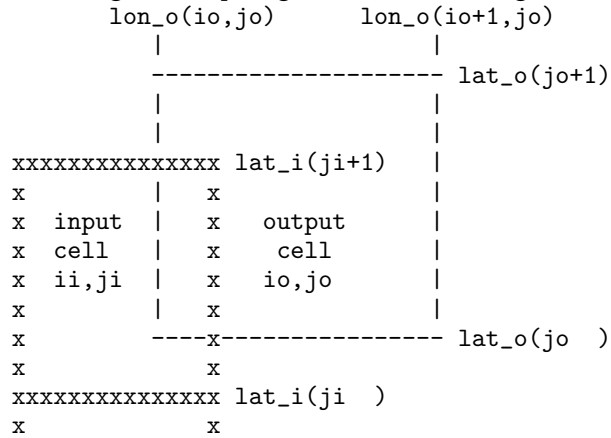
subroutine mkmxovr (nlon_i, nlat_i, numlon_i, lon_i, lat_i, &
 nlon_o, nlat_o, numlon_o, lon_o, lat_o, &
 mxovr , n_ovr)

```



## DESCRIPTION:

find maximum number of overlapping cells  
 For each output grid cell: find overlapping input grid cells that  
 that overlap with output grid cell. Cells overlap if:  
 southern edge of input grid < northern edge of output grid AND  
 northern edge of input grid > southern edge of output grid  
 western edge of input grid < eastern edge of output grid AND  
 eastern edge of input grid > western edge of output grid



lon\_i(ii,ji) lon\_i(ii+1,ji)

The above diagram assumes both grids are oriented South to North. Other combinations of North to South and South to North grids are possible:

|     | Input Grid | Output Grid |
|-----|------------|-------------|
| (1) | S to N     | S to N      |
| (2) | N to S     | N to S      |
| (3) | S to N     | N to S      |
| (4) | N to S     | S to N      |

- 

The code has been modified to allow for North to South grids. Verification that these changes work are:

- o (1) and (4) give same results for output grid
- o (2) and (3) give same results for output grid
- o (2) and (4) give same results for output grid when output grid inverts

## ARGUMENTS:

```

implicit none
integer , intent(in) :: nlon_i !input grid : max number o
integer , intent(in) :: nlat_i !input grid : number of la
integer , intent(in) :: numlon_i(nlat_i) !input grid : number of lo
real(r8), intent(inout) :: lon_i(nlon_i+1,nlat_i) !input grid : cell long
real(r8), intent(in) :: lat_i(nlat_i+1) !input grid : cell latitud
integer , intent(in) :: nlon_o !output grid: max number o
integer , intent(in) :: nlat_o !output grid: number of la
integer , intent(in) :: numlon_o(nlat_o) !output grid: number of lo
real(r8), intent(in) :: lon_o(nlon_o+1,nlat_o) !output grid: cell longitu
real(r8), intent(in) :: lat_o(nlat_o+1) !output grid: cell latitud
integer , intent(out):: n_ovr(nlon_o,nlat_o) !number of overlapping inp
integer , intent(out):: mxovr !maximum number of overlap

```

REVISION HISTORY:

Created by Mariana Vertenstein



## A.39 Module atm\_lndMod

### (Source File: atm\_lndMod.F90)

This module provides the interface between the atmosphere land modules. If running as part of cam, the land surface model must use the same grid as the cam. The land surface model calculates its own net solar radiation and net longwave radiation at the surface. The net longwave radiation at the surface will differ somewhat from that calculated in the atmospheric model because the atm model will use the upward longwave flux (or radiative temperature) from the previous time step whereas the land surface model uses the flux for the current time step. The net solar radiation should equal that calculated in the atmospheric model. If not, there is a problem in how the models are coupled.

USES:

```

use shr_kind_mod , only: r8 => shr_kind_r8
use pmgrid , only: plon, plat
use rgrid , only: nlon
use ppgrid , only: begchunk, endchunk
use phys_grid , only: gather_chunk_to_field
use comsrf , only: snowhland, srfflx_state2d, srfflx_parm2d, &
 srfflx_parm, surface_state, landfrac
use history , only: ctitle, inithist, nhtfrq, mfilt
use filenames , only: caseid
use shr_const_mod, only: SHR_CONST_PI
use abortutils , only: endrun

```

PUBLIC TYPES:

```

implicit none
private ! By default make data private
integer :: landmask(plon,plat) !2d land mask

```

PUBLIC MEMBER FUNCTIONS:

```

public :: atmlnd_ini, atmlnd_drv ! Public interfaces

```

REVISION HISTORY:

Author: Mariana Vertenstein

### A.39.1 atmlnd\_ini

INTERFACE:

```

subroutine atmlnd_ini(srfflx2d)

```

DESCRIPTION:

Initialize land surface model and obtain relevant atmospheric model arrays back from (i.e. albedos, surface temperature and snow cover over land).

USES:

```

 use initializeMod, only : initialize !initialization of clm
 use lp_coupling , only : alltoall_clump_to_chunk_init
 #if (defined SPMD)
 use mpishorthand
 #endif
 use commap
 use time_manager, only: get_nstep
 use filenames, only: mss_irt
 #include <comsol.h>
 #include <comctl.h>

```

#### ARGUMENTS:

```

 type(srfflx_parm),intent(inout),dimension(begchunk:endchunk) :: srfflx2d

```

#### LOCAL VARIABLES:

```

 integer :: i,lat !indices
 integer :: nstep !current timestep number
 integer :: ier !returned error code
 real(r8) :: oro_glob(plon,plat) !global oro field
 real(r8) :: lsmlandfrac(plon,plat) !2d fractional land
 real(r8) :: latixy(plon,plat) !2d latitude grid (degrees)
 real(r8) :: longxy(plon,plat) !2d longitude grid (degrees)
 real(r8) :: pi

```

#### REVISION HISTORY:

Author: Mariana Vertenstein

### A.39.2 atmlnd\_drv

#### INTERFACE:

```

 subroutine atmlnd_drv(nstep, iradsw, eccen, obliqr, lambm0, mvelpp, &
 srf_state, srfflx2d)

```

#### DESCRIPTION:

Pack data to be sent to land model into a single array. Send data to land model and call land model driver. Receive data back from land model in a single array. Unpack this data into component arrays. NOTE: component arrays are contained in module comsrf. When coupling to an atmospheric model: solar radiation depends on surface albedos from the previous time step (based on current surface conditions and solar zenith angle for next time step). Longwave radiation depends on upward longwave flux from previous time step.

#### USES:

```

 #if (defined SPMD)
 use mpishorthand
 #endif
 use comsrf , only : surface_state
 use lp_coupling, only : alltoall_chunk_to_clump, alltoall_clump_to_chunk

```

#### ARGUMENTS:

```

implicit none
integer , intent(in) :: nstep !Current time index
integer , intent(in) :: iradsw !Iteration frequency for shortwave
 !radiation
real(r8), intent(in) :: eccen !Earth's orbital eccentricity
real(r8), intent(in) :: obliqr !Earth's obliquity in radians
real(r8), intent(in) :: lambm0 !Mean longitude of perihelion at the
 !vernal equinox (radians)
real(r8), intent(in) :: mvelpp !Earth's moving vernal equinox
 !longitude of perihelion + pi (radians)
type(srfflx_parm),intent(inout),dimension(begchunk:endchunk) :: srfflx2d
type(surface_state),intent(inout),dimension(begchunk:endchunk) &
:: srf_state

```

## LOCAL VARIABLES:

```

integer :: i,lat,m,n,lchnk,ncols !indices
integer :: ier !returned error code
logical doalb !true if surface albedo calculation time step

```

## REVISION HISTORY:

Author: Mariana Vertenstein



## A.40 Module atmdrvMod

### (Source File: atmdrvMod.F90)

Read and generate atmospheric grid data at model resolution

USES:

```

use shr_kind_mod , only : r8 => shr_kind_r8
use shr_const_mod, only : SHR_CONST_TKFRZ, SHR_CONST_PSTD
use clm_varpar , only : lsmlon, lsmlat
use abortutils , only: endrun
#if (defined SPMD)
 use spmdMod , only : masterproc, mpicom, MPI_REAL8, MPI_INTEGER
#else
 use spmdMod , only : masterproc
#endif

```

PUBLIC TYPES:

```

implicit none
save

```

PUBLIC MEMBER FUNCTIONS:

```

public :: atmdrv ! read atmospheric data
public :: atm_getgrid ! read atmospheric grid

```

REVISION HISTORY:

Created by Gordon Bonan, Sam Levis and Mariana Vertenstein

#### A.40.1 atmdrv

INTERFACE:

```

subroutine atmdrv(nstep)

```

DESCRIPTION:

This code reads in atmospheric fields from an input file and generates the required atmospheric forcing. These data files have [atmmin] minute average data for each month. Input data files are named in month-year format (e.g., 09-0001 contains 240 3-hour time slices of data, 30\*8, for September of year one). The model will cycle through however many full years of data are available [pyr]. At least one full year of data is necessary for cycling. The model may start on any base date, as long as this date corresponds to an existing data file. A run need not be an exact multiple of a year.

```

=====
Possible atmospheric fields:
=====

```

| Name | Description | Required/Optional |
|------|-------------|-------------------|
|------|-------------|-------------------|

-----



|          |                                          |                           |
|----------|------------------------------------------|---------------------------|
| TBOT     | temperature (K)                          | Required                  |
| WIND     | wind:sqrt(u**2+v**2) (m/s)               | Required                  |
| QBOT     | specific humidity (kg/kg)                | Required                  |
| Tdew     | dewpoint temperature (K)                 | Alternative to Q          |
| RH       | relative humidity (percent)              | Alternative to Q          |
| ZBOT     | reference height (m)                     | optional                  |
| PSRF     | surface pressure (Pa)                    | optional                  |
| FSDS     | total incident solar radiation (W/m**2)  | Required                  |
| FSDSdir  | direct incident solar radiation (W/m**2) | optional (replaces FSDS)  |
| FSDSdif  | diffuse incident solar rad (W/m**2)      | optional (replaces FSDS)  |
| FLDS     | incident longwave radiation (W/m**2)     | optional                  |
| PRECTmms | total precipitation (mm H2O / sec)       | Required                  |
| PRECCmms | convective precipitation (mm H2O / sec)  | optional (replaces PRECT) |
| PRECLmms | large-scale precipitation (mm H2O / sec) | optional (replaces PRECT) |

=====  
Data format:  
=====

Data format is netCDF with dimensions longitude x latitude  
for each time slice and field. Variable names can be as in above list  
or can be reset to desired names using [fldlst] in code below.

=====  
Namelist input:  
=====

character\*256 offline\_atmdir = directory for input atm data files (can be M

#### USES:

```

use nanMod
use clmtype
use decompMod , only : get_proc_bounds
use clm_varctl , only : offline_atmdir, pertlim
use clm_varcon , only : rair, cpair, po2, pco2, tcrit
use time_manager, only : get_step_size, get_curr_calday, get_curr_date
use fileutils , only : getfil

```

#### ARGUMENTS:

```

implicit none
integer, intent(in) :: nstep !current time step

```

#### REVISION HISTORY:

Created by Sam Levis

### A.40.2 atm\_getgrid

#### INTERFACE:

```

subroutine atm_getgrid()

```

#### DESCRIPTION:

Read atmospheric grid

## USES:

```

use nanMod
use clm_varctl , only : offline_atmdir
use clm_varcon , only : rair, cpair, po2, pco2
use clm_varsur , only : numlon, longxy, latixy, lsmedge
use fileutils , only : getfil
use time_manager, only : get_curr_date
use ncio

```

## ARGUMENTS:

```

implicit none

```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.40.3 atm\_openfile

## INTERFACE:

```

subroutine atm_openfile (kda, kmo, kyr, locfn, itim, atmmin)

```

## DESCRIPTION:

Open atmospheric forcing netCDF file

## USES:

```

use clm_varctl, only : offline_atmdir
use fileutils , only : getfil
use ncio

```

## ARGUMENTS:

```

implicit none
integer, intent(in) :: kda !day (1 -> 31)
integer, intent(in) :: kmo !month (1 -> 12)
integer, intent(in) :: kyr !year (0 -> ...)
character(len=*), intent(inout) :: locfn !history file to open and read
integer, intent(out) :: itim !time index used in atmrd
integer, intent(out) :: atmmin !temporal resolution of atm data

```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.40.4 atm\_readdata

## INTERFACE:

```
subroutine atm_readdata (fname, kmo, itim)
```

## DESCRIPTION:

```
read atmospheric forcing single level fields from netCDF file
this is done at every subsequent call to atmrd
this includes a second call at the 1st tstep of the run or of the month
Use nf_get_vara_real to read data for variable referenced by
variable id = [i]
nf_get_vara_real (ncid, i, beg, len, varval)
integer beg - a vector of integers specifying the index in the
 variable where the first data value is read.
 must be dimensioned same as the variable's dimension
 and a starting value must be given for each dimension
integer len - a vector of integers specifying the number of data
 values, for each of the variable's dimensions, to read.
 must be dimensioned same as the variable's dimension
 and a length must be given for each dimension
```

## USES:

```
use clm_varcon, only : sb
use fileutils , only : getfil
use ncio
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fname !history file to open and
integer, intent(in) :: kmo, itim !current month and time i
```

## REVISION HISTORY:

Created by Sam Levis

---

## A.40.5 interpa2si

## INTERFACE:

```
subroutine interpa2si()
```

## DESCRIPTION:

```
Initialize variables for atm->land model surface interp
```

## USES:

```
use clm_varsur, only : numlon, longxy, latixy, lsmedge, lonw, lats, area
use areaMod
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Gordon Bonan

---

### A.40.6 interpa2s

#### INTERFACE:

```

subroutine interpa2s (forc_t_a , forc_t_s , zgcm_a , zgcm_s , &
 forc_u_a , forc_u_s , forc_v_a , forc_v_s , &
 forc_q_a , forc_q_s , prc_a , prc_s , &
 prl_a , prl_s , flwds_a , flwds_s , &
 forc_sols_a, forc_sols_s , forc_soll_a , &
 forc_soll_s, forc_solsd_a , forc_solsd_s , &
 forc_solld_a, forc_solld_s, forc_pbot_a , &
 forc_pbot_s , forc_psrfa_a , forc_psrfa_s)

```

#### DESCRIPTION:

Area average fields from atmosphere grid to surface grid

#### USES:

```

use clm_varsur, only : numlon, longxy, latixy, lsmedge
use areaMod

```

#### ARGUMENTS:

```

implicit none
real(r8), intent(in) :: forc_t_a(atmlon,atmlat) !atm bottom level t
real(r8), intent(in) :: zgcm_a(atmlon,atmlat) !atm bottom level h
real(r8), intent(in) :: forc_u_a(atmlon,atmlat) !atm bottom level z
real(r8), intent(in) :: forc_v_a(atmlon,atmlat) !atm bottom level m
real(r8), intent(in) :: forc_q_a(atmlon,atmlat) !atm bottom level s
real(r8), intent(in) :: prc_a(atmlon,atmlat) !convective precipi
real(r8), intent(in) :: prl_a(atmlon,atmlat) !large-scale precip
real(r8), intent(in) :: flwds_a(atmlon,atmlat) !downward longwave
real(r8), intent(in) :: forc_sols_a(atmlon,atmlat) !vis direct beam so
real(r8), intent(in) :: forc_soll_a(atmlon,atmlat) !nir direct beam so
real(r8), intent(in) :: forc_solsd_a(atmlon,atmlat) !vis diffuse solar
real(r8), intent(in) :: forc_solld_a(atmlon,atmlat) !nir diffuse solar
real(r8), intent(in) :: forc_pbot_a(atmlon,atmlat) !atm bottom level p
real(r8), intent(in) :: forc_psrfa_a(atmlon,atmlat) !atm surface pressu

real(r8), intent(out) :: forc_t_s(lsmlon,lsmlat) !atm bottom level t
real(r8), intent(out) :: zgcm_s(lsmlon,lsmlat) !atm bottom level h
real(r8), intent(out) :: forc_u_s(lsmlon,lsmlat) !atm bottom level z
real(r8), intent(out) :: forc_v_s(lsmlon,lsmlat) !atm bottom level m
real(r8), intent(out) :: forc_q_s(lsmlon,lsmlat) !atm bottom level s
real(r8), intent(out) :: prc_s(lsmlon,lsmlat) !convective precipi
real(r8), intent(out) :: prl_s(lsmlon,lsmlat) !large-scale precip
real(r8), intent(out) :: flwds_s(lsmlon,lsmlat) !downward longwave
real(r8), intent(out) :: forc_sols_s(lsmlon,lsmlat) !vis direct beam so
real(r8), intent(out) :: forc_soll_s(lsmlon,lsmlat) !nir direct beam so
real(r8), intent(out) :: forc_solsd_s(lsmlon,lsmlat) !vis diffuse solar
real(r8), intent(out) :: forc_solld_s(lsmlon,lsmlat) !nir diffuse solar
real(r8), intent(out) :: forc_pbot_s(lsmlon,lsmlat) !atm bottom level p
real(r8), intent(out) :: forc_psrfa_s(lsmlon,lsmlat) !atm surface pressu

```

#### REVISION HISTORY:

Created by Gordon Bonan



## A.41 Module clm\_csmMod

### (Source File: clm\_csmMod.F90)

Set of routines that define communication between the land model and flux coupler. The order of sends/receives is:

- 1) receive orbital data from coupler
- 2) send control data (grids and masks) to coupler  
land grid does not have valid data, runoff grid does
- 3) receive valid land grid from flux coupler
- 4) send compressed runoff information to flux coupler
- 5) start normal send/recv communication pattern

USES:

```

 use shr_kind_mod, only: r8 => shr_kind_r8
 use nanMod
 use clm_varpar
 #if (defined SPMD)
 use spmdMod , only : masterproc, mpicom
 use spmdGathScatMod, only : gather_data_to_master
 #else
 use spmdMod , only : masterproc
 #endif
 use mpiinc
 use cpl_fields_mod
 use cpl_contract_mod
 use cpl_interface_mod
 use RunoffMod , only : runoff
 use shr_sys_mod , only : shr_sys_irtc, shr_sys_flush ! csm_share syste
 use system_messages , only : allocation_err ! allocation erro
 use abortutils , only : endrun

```

PUBLIC TYPES:

```

 implicit none
 save

```

PUBLIC MEMBER FUNCTIONS:

```

 public :: csm_setup ! Setup, mpi_init
 public :: csm_shutdown ! Shutdown, mpi_finalize
 public :: csm_initialize ! Initialize contracts, etc
 public :: csm_recvgrid ! Receive grid and land mask
 public :: csm_dosndrcv ! Logic for determining if send/recv
 public :: csm_recv ! Receive data from flux coupler
 public :: csm_send ! Send data to flux coupler
 public :: csm_sendalb ! Send initial albedos, surface temp and snow
 public :: csm_flxave ! Flux averaging routine
 public :: restart_coupler ! Restart code
 public :: compat_check_spval ! Checks that data sent from the coupler is va
 public :: csm_compat ! Checks compatibility of messages send/receiv

```

REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

03.01.15 T. Craig Update for cpl6  
 03.04.27 M. Vertenstein, added qref\_2m to communication and  
 generalized global sums to include all fields

---

### A.41.1 csm\_setup

INTERFACE:

```
subroutine csm_setup(mpicom)
```

DESCRIPTION:

Initialize csm coupling, return the communicator group to the application.

ARGUMENTS:

```
implicit none
integer, intent(out) :: mpicom !MPI group communicator
```

REVISION HISTORY:

03.01.15 T. Craig: initial version

---

### A.41.2 csm\_shutdown

INTERFACE:

```
subroutine csm_shutdown
```

DESCRIPTION:

Finalize csm coupling

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

03.01.15 T. Craig: initial version

---

### A.41.3 csm\_initialize

INTERFACE:

```
subroutine csm_initialize(irad, eccen, obliqr, lambm0, mvelpp)
```

## DESCRIPTION:

Send first control data to flux coupler and "invalid" grid containing special value data.  
 The coupler treats points where the mask is nonzero as points where you could possibly do a calculation (in the case of the runoff, this corresponds to all rtm ocean points). The coupler then defines a "key" as points where the model can give you valid data (in the case of runoff, this corresponds to points where the land model will give you valid compressed data points). The key can be 0 where the mask is 1. However, the key cannot be 1 where the mask is 0 unless the data is also zero. In the case of runoff, the key the coupler builds is time invariant.  
 Send first control data to flux coupler and "invalid" grid containing special value data

## USES:

```

use clmtype
use decompMod , only : get_proc_bounds, get_proc_global
use RunoffMod , only : get_proc_rof_bounds, runoff
use clm_varctl , only : csm_doflxave, nsrest
use RtmMod , only : area_r, longxy_r, latixy_r, mask_r
use clm_varcon , only : re
use time_manager , only : get_step_size
use shr_const_mod , only : SHR_CONST_CDAY

```

## ARGUMENTS:

```

implicit none
integer , intent(in) :: irad ! frequency of radiation computation
real(r8), intent(out) :: eccen ! Earth's eccentricity of orbit
real(r8), intent(out) :: obliqr ! Earth's obliquity in radians
real(r8), intent(out) :: lambm0 ! Mean longitude of perihelion at the ver
real(r8), intent(out) :: mvelpp ! Earth's moving vernal equinox long of p

```

## REVISION HISTORY:

```

02.09.17 Mariana Vertenstein Updated to clm2_1 data structures
03.01.15 T.Craig Update for cpl6

```

## A.41.4 csm\_recvgrid

## INTERFACE:

```

subroutine csm_recvgrid(cam_longxy, cam_latixy, cam_numlon, &
 cam_landfrac, cam_landmask)

```

## DESCRIPTION:

Receive valid land grid and land mask from coupler

## ARGUMENTS:



```

implicit none
integer , intent(out) :: cam_numlon(lsmlat) !cam number of long
real(r8), intent(out) :: cam_longxy(lsmlon,lsmlat) !cam lon values
real(r8), intent(out) :: cam_latixy(lsmlon,lsmlat) !cam lat values
real(r8), intent(out) :: cam_landfrac(lsmlon,lsmlat) !cam fractional lan
integer , intent(out) :: cam_landmask(lsmlon,lsmlat) !cam land mask

```

#### REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

### A.41.5 csm\_sendalb

#### INTERFACE:

```
subroutine csm_sendalb()
```

#### DESCRIPTION:

Send initial albedos, surface temperature and snow data to the flux coupler

#### USES:

```

use clmtype
use clm_varsur
use clm_varctl , only : csm_doflxave, nsrest
use clm_varcon , only : sb
use time_manager, only : get_curr_date, get_prev_date, get_nstep
use lnd2atmMod , only : lnd2atm

```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

### A.41.6 csm\_dosndrcv

#### INTERFACE:

```
subroutine csm_dosndrcv(doalb)
```

#### DESCRIPTION:

Determine when to send and receive messages to/from the flux coupler on this time-step.  
Determine if send/receive information to/from flux coupler  
Send msgs (land state and fluxes) to the flux coupler only when

doalb is true (i.e. on time steps before the atm does a solar radiation computation). Receive msgs (atm state) from the flux coupler only when dorad is true (i.e. on time steps when the atm does a solar radiation computation).  
The fluxes are then averaged between the send and receive calls.

## USES:

```
use clm_varctl , only : csm_doflxave
use time_manager , only : get_step_size, get_nstep
use shr_const_mod, only : SHR_CONST_CDAY
```

## ARGUMENTS:

```
implicit none
logical, intent(in) :: doalb !true=>next timestep a radiation time step
```

## REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

**A.41.7 csm\_recv**

## INTERFACE:

```
subroutine csm_recv()
```

## DESCRIPTION:

Receive and map data from flux coupler

## USES:

```
use clmtype
use clm_varcon, only : rair, po2, pco2
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

**A.41.8 csm\_send**

## INTERFACE:

```
subroutine csm_send()
```

## DESCRIPTION:

Send data to the flux coupler

USES:

```
use clmtype
use clm_varctl , only : csm_doflxave
use clm_varsur , only : landmask
use clm_varcon , only : sb
use time_manager, only : get_curr_date, get_nstep
use lnd2atmMod , only : lnd2atm
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

### A.41.9 csm\_flxave

INTERFACE:

```
subroutine csm_flxave()
```

DESCRIPTION:

Average output fluxes for flux coupler  
Add land surface model output fluxes to accumulators every time step.  
When icnt==ncnt, compute the average flux over the time interval.

USES:

```
use clmtype
use clm_varctl , only : irad
use time_manager, only : get_nstep
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

---

### A.41.10 compat\_check

INTERFACE:

```
subroutine compat_check_spval(spval, data, string)
```

DESCRIPTION:

Check that the given piece of real data sent from the coupler is valid data and not the couplers special data flag. This ensures that the data you expect is actually being sent by the coupler.

#### ARGUMENTS:

```
implicit none
real(r8), intent(in) :: spval
real(r8), intent(in) :: data
character(len=*), intent(in) :: string
```

#### REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

### A.41.11 csm\_compat

#### INTERFACE:

```
subroutine csm_compat(cpl_maj_vers, cpl_min_vers, expect_maj_vers, &
 expect_min_vers)
```

#### DESCRIPTION:

Checks that the message recieved from the coupler is compatible with the type of message that I expect to receive. If the minor version numbers differ I print a warning message. If the major numbers differ I abort since that means that the change is drastic enough that I can't run with the differences.

Original Author: Erik Kluzek Dec/97

#### PARAMETERS:

```
implicit none
integer, intent(in) :: cpl_maj_vers ! major version from coupler initi
integer, intent(in) :: cpl_min_vers ! minor version from coupler initi
integer, intent(in) :: expect_maj_vers ! major version of the coupler I'm
integer, intent(in) :: expect_min_vers ! minor version of the coupler I'm
```

#### REVISION HISTORY:

02.09.17 Mariana Vertenstein Updated to clm2\_1 data structures

### A.41.12 restart\_coupler

#### INTERFACE:

```
subroutine restart_coupler(nio, flag)
```

#### DESCRIPTION:

Read/write restart data needed for running in flux coupled mode

USES:

```
use clm_varctl, only : csm_doflxave
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: nio !restart unit
character(len=*), intent(in) :: flag !"read" or "write"
```

REVISION HISTORY:

02.09.17 Mariana Vertenstein: moved code to be part of ccs module

---

### A.41.13 `global_sum_fld2d`

INTERFACE:

```
real(r8) function global_sum_fld2d(array, spval)
```

DESCRIPTION:

Performs a global sum on an input 2d grid array

USES:

```
use clm_varsur, only : area !km^2
```

ARGUMENTS:

```
implicit none
real(r8), intent(in) :: array(lsm1on,lsm1at) !W/m2, Kg/m2-s or N/m2
real(r8), intent(in) :: spval !points to not include in gl
```

REVISION HISTORY:

---

### A.41.14 `global_sum_fld1d`

INTERFACE:

```
real(r8) function global_sum_fld1d(array)
```

DESCRIPTION:

Performs a global sum on an input flux array

USES:

```
use clmtype
use clm_varsur, only : area
```

ARGUMENTS:

```
implicit none
real(r8), intent(in) :: array(:) !W/m2, Kg/m2-s or N/m2
```

REVISION HISTORY:



## A.42 Module `clm_varcon` (Source File: `clm_varcon.F90`)

Module containing various model constants

USES:

```
use shr_kind_mod , only: r8 => shr_kind_r8
use shr_const_mod, only: SHR_CONST_G,SHR_CONST_STEBOL,SHR_CONST_KARMAN,
 SHR_CONST_RWV,SHR_CONST_RDAIR,SHR_CONST_CPFW,
 SHR_CONST_CPICE,SHR_CONST_CPDAIR,SHR_CONST_LATVAP,
 SHR_CONST_LATSUB,SHR_CONST_LATICE,SHR_CONST_RHOFW,
 SHR_CONST_RHOICE,SHR_CONST_TKFRZ,SHR_CONST_REARTH
use clm_varpar , only: numcol, numrad, nlevsoi, nlevlak
```

PUBLIC TYPES:

```
implicit none
save
```

REVISION HISTORY:

Created by Mariana Vertenstein





## A.43 Module clm\_varctl

### (Source File: clm\_varctl.F90)

Module containing run control variables

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
Run control variables
character(len=256) :: caseid ! case id
character(len=256) :: ctitle ! case title
integer :: nsrest ! 0: initial run. 1: restart:
logical, public :: brnch_retain_casename = .false. ! true => allow case nam
 ! by default this is not

Initial file variables
character(len= 8) :: hist_crtinic ! if set to 'MONTHLY' or 'YEA
Long term archive variables
character(len=256) :: archive_dir ! long term archive directory
character(len= 8) :: mss_wpass ! mass store write password f
integer :: mss_irt ! mass store retention period

Run input files
character(len=256) :: finidat ! initial conditions file nam
character(len=256) :: fsurdat ! surface data file name
character(len=256) :: fpftcon ! ASCII data file with PFT ph
character(len=256) :: nrevsn ! restart data file name for
character(len=256) :: frivinp_rtm ! RTM input data file name
character(len=256) :: offline_atmdir ! directory for input offline

Files and logical variables for generating surface dataset
logical :: mksrf_all_pfts ! true => surface dataset wi
real(r8) :: mksrf_offline_edgen ! northern edge of grid (deg
real(r8) :: mksrf_offline_edgee ! eastern edge of grid (degr
real(r8) :: mksrf_offline_edges ! southern edge of grid (deg
real(r8) :: mksrf_offline_edgew ! western edge of grid (degr
character(len=256) :: mksrf_offline_fgrid ! land grid file name to use
character(len=256) :: mksrf_offline_fnavyoro ! directory for 20 min navy
character(len=256) :: mksrf_fvegtyp ! when making [fsurdat]: veg
character(len=256) :: mksrf_fsoitex ! when making [fsurdat]: soi
character(len=256) :: mksrf_fsoicol ! when making [fsurdat]: soi
character(len=256) :: mksrf_flanwat ! when making [fsurdat]: inl
character(len=256) :: mksrf_furban ! when making [fsurdat]: urb
character(len=256) :: mksrf_fglacier ! when making [fsurdat]: gla
character(len=256) :: mksrf_flai ! when making [fsurdat]: lai

Physics
integer :: irad ! solar radiation frequency (iterations)
logical :: wrtdia ! true => write global average diagnostics to std o
logical :: csm_doflxave ! true => only communicate with flux coupler on alb
Rtm control variables
integer :: rtm_nsteps ! if > 1, average rtm over rtm_nsteps time steps
Derived variables (run, history and restart file)
character(len=256) :: rpntdir ! directory name for local restart p
```

```
character(len=256) :: rpntfil ! file name for local restart pointe
character(len=256) :: version ! model version number
Error growth perturbation limit
real(r8) :: pertlim ! perturbation limit when doing erro
```

## REVISION HISTORY:

Created by Mariana Vertenstein and Gordon Bonan

## A.44 Module clm\_varpar (Source File: clm\_varpar.F90)

Module containing CLM parameters

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
Define land surface 2-d grid. This sets the model resolution according
to cpp directives LSMLON and LSMLAT in preproc.h.
integer, parameter :: lsmlon = LSMLON !maximum number of longitude points
integer, parameter :: lsmlat = LSMLAT !number of latitude points on lsm gr
Define indices used in surface file read
maxpatch_pft = max number of vegetated pfts in naturally vegetated landunit
maxpatch_crop = max number of crop pfts in crop landunit
#if (defined DGVM)
integer, parameter :: maxpatch_pft = 10
#else
integer, parameter :: maxpatch_pft = 4
#endif
integer, parameter :: maxpatch_cft = 2
integer, parameter :: npatch_urban = maxpatch_pft + 1
integer, parameter :: npatch_lake = npatch_urban + 1
integer, parameter :: npatch_wet = npatch_lake + 1
integer, parameter :: npatch_glacier = npatch_wet + 1
integer, parameter :: npatch_crop = npatch_glacier + maxpatch_cft
integer, parameter :: maxpatch = npatch_crop
Determine maximums in subgrid hierarchy
integer, parameter :: max_pft_per_col = maxpatch_pft
#if (defined NOCOMPETE)
integer, parameter :: max_col_per_lunit = maxpatch_pft
#else
integer, parameter :: max_col_per_lunit = 1
#endif
integer, parameter :: max_lunit_per_gcell = 5 ! (soil,urban,lake,
Define number of levels
integer, parameter :: nlevsoi = 10 !number of soil layers
integer, parameter :: nlevlak = 10 !number of lake layers
integer, parameter :: nlevsno = 5 !maximum number of snow layers
Define miscellaneous parameters
integer, parameter :: numwat = 5 !number of water types (soil, ice
integer, parameter :: numpft = 16 !number of plant types
integer, parameter :: npftpar = 32 !number of pft parameters (in LPJ
integer, parameter :: numcol = 8 !number of soil color types
integer, parameter :: numrad = 2 !number of solar radiation bands:
integer, parameter :: ndst = 4 !number of dust size classes (BGC
integer, parameter :: dst_src_nbr = 3 !number of size distns in src soi
integer, parameter :: sz_nbr = 200 !number of sub-grid bins in large
integer, parameter :: nvoc = 5 !number of voc categories (BGC on
Define parameters for RTM river routing model
```

```
integer, parameter :: rtmlon = 720 !number of rtm longitudes
integer, parameter :: rtmlat = 360 !number of rtm latitudes
```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.45 Module clm\_varsur

### (Source File: clm\_varsur.F90)

Module containing 2-d surface boundary data information

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar, only : lsmlon, lsmlat, nlevsoi
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
land model grid
integer :: numlon(lsmlat) !longitude points for each latitude s
real(r8):: latixy(lsmlon,lsmlat) !latitude of grid cell (degrees)
real(r8):: longxy(lsmlon,lsmlat) !longitude of grid cell (degrees)
real(r8):: area(lsmlon,lsmlat) !grid cell area (km**2)
real(r8):: landarea !total land area for all gridcells (k
real(r8):: lats(lsmlat+1) !grid cell latitude, southern edge (d
real(r8):: lonw(lsmlon+1,lsmlat) !grid cell longitude, western edge (d
real(r8):: lsmedge(4) !North,East,South,West edges of grid
logical :: pole_points !true => grid has pole points
logical :: fullgrid = .true. !true => no grid reduction towards po
logical :: offline_rdgrid !true => read offline grid rather tha
fractional land and mask
integer landmask(lsmlon,lsmlat) !land mask: 1 = land. 0 = ocean
real(r8) landfrac(lsmlon,lsmlat) !fractional land
surface boundary data
integer , allocatable :: soic2d(:,) !soil color
real(r8), allocatable :: sand3d(:,,:) !soil texture: percent sand
real(r8), allocatable :: clay3d(:,,:) !soil texture: percent clay
real(r8), allocatable :: pctgla(:,) !percent of grid cell that is glacie
real(r8), allocatable :: pctlak(:,) !percent of grid cell that is lake
real(r8), allocatable :: pctwet(:,) !percent of grid cell that is wetlan
real(r8), allocatable :: pcturb(:,) !percent of grid cell that is urbani
logical, public :: all_pfts_on_srfdat = .false. ! old format dataset is us
```

PUBLIC MEMBER FUNCTIONS:

```
public :: varsur_alloc !allocates 2d surface data needed for initializat
public :: varsur_dealloc !deallocates 2d surface data needed for initializ
```

REVISION HISTORY:

Created by Mariana Vertenstein

#### A.45.1 varsur\_alloc

INTERFACE:

```
subroutine varsur_alloc
```

## DESCRIPTION:

Allocate dynamic memory for module variables

## ARGUMENTS:

implicit none

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.45.2 varsur\_dealloc**

## INTERFACE:

subroutine varsur\_dealloc

## DESCRIPTION:

Deallocate dynamic memory for module variables

## ARGUMENTS:

implicit none

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.46 Module clmtype

### (Source File: clmtype.F90)

Define derived type hierarchy. Includes declaration of the clm derived type and id mapping arrays.

```
1 => default
landunits types can have values of (see clm_varcon.F90)
 (note shallow lakes not currently implemented)
1 => (istsoil) soil (vegetated or bare soil landunit)
2 => (istice) land ice
3 => (istdlak) deep lake
5 => (istwet) wetland
6 => (isturb) urban
column types can have values of
1 => in compete mode
pft type values (see below) => in non-compete mode
pft types can have values of
0 => not vegetated
1 => needleleaf evergreen temperate tree
2 => needleleaf evergreen boreal tree
3 => needleleaf deciduous boreal tree
4 => broadleaf evergreen tropical tree
5 => broadleaf evergreen temperate tree
6 => broadleaf deciduous tropical tree
7 => broadleaf deciduous temperate tree
8 => broadleaf deciduous boreal tree
9 => broadleaf evergreen shrub
10 => broadleaf deciduous temperate shrub
11 => broadleaf deciduous boreal shrub
12 => c3 arctic grass
13 => c3 non-arctic grass
14 => c4 grass
15 => corn
16 => wheat
```

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar
```

PUBLIC TYPES:

```
implicit none
```

REVISION HISTORY:

Created by Peter Thornton and Mariana Vertenstein

\*\*\*\*\*

-----  
Begin definition of conservation check structures  
-----

energy balance structure  
-----

```
type energy_balance_type
 real(r8), pointer :: errsoi(:) !soil/lake energy conservation error
 real(r8), pointer :: errseb(:) !surface energy conservation error (
```



```

 real(r8), pointer :: errsol(:) !solar radiation conservation error
 real(r8), pointer :: errlon(:) !longwave radiation conservation err
end type energy_balance_type

```

```

water balance structure

```

```

type water_balance_type
 real(r8), pointer :: begwb(:) !water mass begining of the time ste
 real(r8), pointer :: endwb(:) !water mass end of the time step
 real(r8), pointer :: errh2o(:) !water conservation error (mm H2O)
end type water_balance_type

```

```

carbon balance structure

```

```

type carbon_balance_type
 real(r8), pointer :: dummy_entry(:)
end type carbon_balance_type

```

```

nitrogen balance structure

```

```

type nitrogen_balance_type
 real(r8), pointer :: dummy_entry(:)
end type nitrogen_balance_type

```

```

End definition of conservation check structures

```

```

```

```

```

```

Begin definition of structures defined at the pft_type level

```

```

pft physical state variables structure

```

```

type pft_pstate_type
 integer , pointer :: frac_veg_nosno(:) !fraction of vegetation not c
 integer , pointer :: frac_veg_nosno_alb(:) !fraction of vegetation not c
 real(r8), pointer :: rsw(:) !soil water content for root
 real(r8), pointer :: emv(:) !vegetation emissivity
 real(r8), pointer :: z0mv(:) !roughness length over vegeta
 real(r8), pointer :: z0hv(:) !roughness length over vegeta
 real(r8), pointer :: z0qv(:) !roughness length over vegeta
 real(r8), pointer :: rootfr(:, :) !fraction of roots in each so
 real(r8), pointer :: rootr(:, :) !effective fraction of roots
 real(r8), pointer :: dewmx(:) !Maximum allowed dew [mm]
 real(r8), pointer :: rssun(:) !sunlit stomatal resistance (
 real(r8), pointer :: rsha(:) !shaded stomatal resistance (
 real(r8), pointer :: laisun(:) !sunlit leaf area
 real(r8), pointer :: laisha(:) !shaded leaf area
 real(r8), pointer :: btran(:) !transpiration wetness factor
 real(r8), pointer :: fsun(:) !sunlit fraction of canopy

```

```

real(r8), pointer :: tlai(:) !one-sided leaf area index, n
real(r8), pointer :: tsai(:) !one-sided stem area index, n
real(r8), pointer :: elai(:) !one-sided leaf area index wi
real(r8), pointer :: esai(:) !one-sided stem area index wi
real(r8), pointer :: igs(:) !growing season index (0=off,
real(r8), pointer :: stembio(:) !stem biomass (kg /m**2)
real(r8), pointer :: rootbio(:) !root biomass (kg /m**2)
real(r8), pointer :: fwet(:) !fraction of canopy that is w
real(r8), pointer :: fdry(:) !fraction of foliage that is
real(r8), pointer :: dt_veg(:) !change in t_veg, last iterat
real(r8), pointer :: htop(:) !canopy top (m)
real(r8), pointer :: hbot(:) !canopy bottom (m)
real(r8), pointer :: z0m(:) !momentum roughness length (m)
real(r8), pointer :: displa(:) !displacement height (m)
real(r8), pointer :: albd(:, :) !surface albedo (direct)
real(r8), pointer :: albi(:, :) !surface albedo (diffuse)
real(r8), pointer :: fabd(:, :) !flux absorbed by veg per uni
real(r8), pointer :: fabi(:, :) !flux absorbed by veg per uni
real(r8), pointer :: ftdd(:, :) !down direct flux below veg p
real(r8), pointer :: ftid(:, :) !down diffuse flux below veg
real(r8), pointer :: ftii(:, :) !down diffuse flux below veg
real(r8), pointer :: u10(:) !10-m wind (m/s) (for dust mo
real(r8), pointer :: ram1(:) !aerodynamical resistance (s/
real(r8), pointer :: fv(:) !friction velocity (m/s) (for
end type pft_pstate_type

```

```

pft ecophysiological constants structure

```

```

type pft_epc_type
integer , pointer :: ncorn(:) !value for corn
integer , pointer :: nwheat(:) !value for wheat
integer , pointer :: novveg(:) !value for not vegetated
integer , pointer :: ntree(:) !value for last type of tree
real(r8), pointer :: smpmax(:) !Wilting point potential in m
real(r8), pointer :: foln(:) !foliage nitrogen (%)
real(r8), pointer :: dleaf(:) !characteristic leaf dimensio
real(r8), pointer :: c3psn(:) !photosynthetic pathway: 0. =
real(r8), pointer :: vcmx25(:) !max rate of carboxylation at
real(r8), pointer :: mp(:) !slope of conductance-to-phot
real(r8), pointer :: qe25(:) !quantum efficiency at 25C (u
real(r8), pointer :: xl(:) !leaf/stem orientation index
real(r8), pointer :: rho1(:, :) !leaf reflectance: 1=vis, 2=n
real(r8), pointer :: rho2(:, :) !stem reflectance: 1=vis, 2=n
real(r8), pointer :: tau1(:, :) !leaf transmittance: 1=vis, 2
real(r8), pointer :: tau2(:, :) !stem transmittance: 1=vis, 2
real(r8), pointer :: z0mr(:) !ratio of momentum roughness
real(r8), pointer :: displar(:) !ratio of displacement height
real(r8), pointer :: roota_par(:) !CLM rooting distribution par
real(r8), pointer :: rootb_par(:) !CLM rooting distribution par
real(r8), pointer :: sla(:) !specific leaf area [m2 leaf
end type pft_epc_type

```

---

pft DGVM-specific ecophysiological constants structure

---

```

type pft_dgvepc_type
 real(r8), pointer :: respcoeff(:) !maintenance respiration coefficie
 real(r8), pointer :: flam(:) !flammability threshold [units?]
 real(r8), pointer :: resist(:) !fire resistance index [units?]
 real(r8), pointer :: l_turn(:) !leaf turnover period [years]
 real(r8), pointer :: l_long(:) !leaf longevity [years]
 real(r8), pointer :: s_turn(:) !sapwood turnover period [years]
 real(r8), pointer :: r_turn(:) !root turnover period [years]
 real(r8), pointer :: l_pton(:) !leaf C:N (mass ratio)
 real(r8), pointer :: s_pton(:) !sapwood C:N (mass ratio)
 real(r8), pointer :: r_pton(:) !root C:N (mass ratio)
 real(r8), pointer :: l_morph(:) !leaf morphology: 1=broad, 2=needle
 real(r8), pointer :: l_phen(:) !leaf phenology: 1=everg, 2=summer
 real(r8), pointer :: lmtorm(:) !leaf:root ratio under non-water s
 real(r8), pointer :: crownarea_max(:) !tree maximum crown area [m2]
 real(r8), pointer :: init_lai(:) !sapling (or initial grass) LAI [-
 real(r8), pointer :: x(:) !sapling: (heart+sapwood)/sapwood
 real(r8), pointer :: tcmin(:) !minimum coldest monthly mean temp
 real(r8), pointer :: tcmax(:) !maximum coldest monthly mean temp
 real(r8), pointer :: gddmin(:) !minimum growing degree days (at o
 real(r8), pointer :: twmax(:) !upper limit of temperature of the
 real(r8), pointer :: lm_sapl(:)
 real(r8), pointer :: sm_sapl(:)
 real(r8), pointer :: hm_sapl(:)
 real(r8), pointer :: rm_sapl(:)
 logical , pointer :: tree(:)
 logical , pointer :: summergreen(:)
 logical , pointer :: raingreen(:)
 real(r8), pointer :: reinickerp(:) !parameter in allometric equation
 real(r8), pointer :: wooddens(:) !wood density (gC/m3)
 real(r8), pointer :: latosa(:) !ratio of leaf area to sapwood cro
 real(r8), pointer :: allom1(:) !parameter in allometric
 real(r8), pointer :: allom2(:) !parameter in allometric
 real(r8), pointer :: allom3(:) !parameter in allometric
end type pft_dgvepc_type

```

---

pft energy state variables structure

---

```

type pft_estate_type
 real(r8), pointer :: t_ref2m(:) !2 m height surface air temperatu
 real(r8), pointer :: t_ref2m_min(:) !daily minimum of average 2 m hei
 real(r8), pointer :: t_ref2m_max(:) !daily maximum of average 2 m hei
 real(r8), pointer :: t_ref2m_min_inst(:) !instantaneous daily min of avera
 real(r8), pointer :: t_ref2m_max_inst(:) !instantaneous daily max of avera
 real(r8), pointer :: q_ref2m(:) !2 m height surface specific humi
 real(r8), pointer :: t_veg(:) !vegetation temperature (Kelvin)
end type pft_estate_type

```

---

pft water state variables structure

---

```

type pft_wstate_type
 real(r8), pointer :: h2ocan(:) !canopy water (mm H2O)
end type pft_wstate_type

 pft carbon state variables structure

type pft_cstate_type
 real(r8), pointer :: dummy_entry(:)
end type pft_cstate_type

 pft nitrogen state variables structure

type pft_nstate_type
 real(r8), pointer :: dummy_entry(:)
end type pft_nstate_type

 pft VOC state variables structure

type pft_vstate_type
 real(r8), pointer :: dummy_entry(:)
end type pft_vstate_type

 pft DGVM state variables structure

type pft_dgvstate_type
 real(r8), pointer :: agdd0(:) !accumulated growing degree da
 real(r8), pointer :: agdd5(:) !accumulated growing degree da
 real(r8), pointer :: agddtw(:) !accumulated growing degree da
 real(r8), pointer :: agdd(:) !accumulated growing degree da
 real(r8), pointer :: t10(:) !10-day running mean of the 2
 real(r8), pointer :: t_mo(:) !30-day average temperature (K
 real(r8), pointer :: t_mo_min(:) !annual min of t_mo (Kelvin)
 real(r8), pointer :: fnpsn10(:) !10-day running mean net photo
 real(r8), pointer :: prec365(:) !365-day running mean of tot.
 real(r8), pointer :: agdd20(:) !20-yr running mean of agdd
 real(r8), pointer :: tmomin20(:) !20-yr running mean of tmomin
 real(r8), pointer :: t10min(:) !annual minimum of 10-day runn
 real(r8), pointer :: tsoi25(:) !soil temperature to 0.25 m (K
 real(r8), pointer :: annpsn(:) !annual photosynthesis (umol C
 real(r8), pointer :: annpsnpot(:) !annual potential photosynthes
 logical , pointer :: present(:) !whether PFT present in patch
 real(r8), pointer :: dphen(:) !phenology [0 to 1]
 real(r8), pointer :: leafon(:) !leafon days
 real(r8), pointer :: leafof(:) !leafoff days
 real(r8), pointer :: nind(:) !number of individuals (#/m**2
 real(r8), pointer :: lm_ind(:) !individual leaf mass
 real(r8), pointer :: sm_ind(:) !individual sapwood mass
 real(r8), pointer :: hm_ind(:) !individual heartwood mass
 real(r8), pointer :: rm_ind(:) !individual root mass
 real(r8), pointer :: lai_ind(:) !LAI per individual

```

```

real(r8), pointer :: fpcinc(:) !foliar projective cover incre
real(r8), pointer :: fpcgrid(:) !foliar projective cover on gr
real(r8), pointer :: crownarea(:) !area that each individual tre
real(r8), pointer :: bm_inc(:) !biomass increment
real(r8), pointer :: afmicr(:) !annual microbial respiration
real(r8), pointer :: firelength(:) !fire season in days
real(r8), pointer :: litterag(:) !above ground litter
real(r8), pointer :: litterbg(:) !below ground litter
real(r8), pointer :: cpool_fast(:) !fast carbon pool
real(r8), pointer :: cpool_slow(:) !slow carbon pool
real(r8), pointer :: k_fast_ave(:) !decomposition rate
real(r8), pointer :: k_slow_ave(:) !decomposition rate
real(r8), pointer :: litter_decom_ave(:) !decomposition rate
real(r8), pointer :: turnover_ind(:) !
end type pft_dgvstate_type

```

---

pft energy flux variables structure

---

```

type pft_eflux_type
 real(r8), pointer :: sabg(:) !solar radiation absorbed by gro
 real(r8), pointer :: sabv(:) !solar radiation absorbed by veg
 real(r8), pointer :: fsa(:) !solar radiation absorbed (total
 real(r8), pointer :: fsr(:) !solar radiation reflected (W/m*
 real(r8), pointer :: parsun(:) !average absorbed PAR for sunlit
 real(r8), pointer :: parsha(:) !average absorbed PAR for shaded
 real(r8), pointer :: dlrاد(:) !downward longwave radiation bel
 real(r8), pointer :: ulrad(:) !upward longwave radiation above
 real(r8), pointer :: eflx_lh_tot(:) !total latent heat flux (W/m8*2)
 real(r8), pointer :: eflx_lh_grnd(:) !ground evaporation heat flux (W
 real(r8), pointer :: eflx_soil_grnd(:) !soil heat flux (W/m**2) [+ = in
 real(r8), pointer :: eflx_sh_tot(:) !total sensible heat flux (W/m**
 real(r8), pointer :: eflx_sh_grnd(:) !sensible heat flux from ground
 real(r8), pointer :: eflx_sh_veg(:) !sensible heat flux from leaves
 real(r8), pointer :: eflx_lh_vege(:) !veg evaporation heat flux (W/m*
 real(r8), pointer :: eflx_lh_vegt(:) !veg transpiration heat flux (W/
 real(r8), pointer :: cgrnd(:) !deriv. of soil energy flux wrt
 real(r8), pointer :: cgrndl(:) !deriv. of soil latent heat flux
 real(r8), pointer :: cgrnds(:) !deriv. of soil sensible heat fl
 real(r8), pointer :: eflx_gnet(:) !net heat flux into ground (W/m*
 real(r8), pointer :: dgnetdT(:) !derivative of net ground heat f
 real(r8), pointer :: eflx_lwrad_out(:) !emitted infrared (longwave) rad
 real(r8), pointer :: eflx_lwrad_net(:) !net infrared (longwave) rad (W/
 real(r8), pointer :: fsds_vis_d(:) !incident direct beam vis solar
 real(r8), pointer :: fsds_nir_d(:) !incident direct beam nir solar
 real(r8), pointer :: fsds_vis_i(:) !incident diffuse vis solar radi
 real(r8), pointer :: fsds_nir_i(:) !incident diffuse nir solar radi
 real(r8), pointer :: fsr_vis_d(:) !reflected direct beam vis solar
 real(r8), pointer :: fsr_nir_d(:) !reflected direct beam nir solar
 real(r8), pointer :: fsr_vis_i(:) !reflected diffuse vis solar rad
 real(r8), pointer :: fsr_nir_i(:) !reflected diffuse nir solar rad
 real(r8), pointer :: fsds_vis_d_ln(:) !incident direct beam vis solar
 real(r8), pointer :: fsds_nir_d_ln(:) !incident direct beam nir solar
 real(r8), pointer :: fsr_vis_d_ln(:) !reflected direct beam vis solar

```

```

 real(r8), pointer :: fsr_nir_d_ln(:) !reflected direct beam nir solar
end type pft_eflux_type

```

```

pft momentum flux variables structure

```

```

type pft_mflux_type
 real(r8), pointer :: tau_x(:) !wind (shear) stress: e-w (kg/m/s**
 real(r8), pointer :: tau_y(:) !wind (shear) stress: n-s (kg/m/s**
end type pft_mflux_type

```

```

pft water flux variables structure

```

```

type pft_wflux_type
 real(r8), pointer :: qflx_prec_intr(:) !interception of precipitation [mm/
 real(r8), pointer :: qflx_prec_grnd(:) !water onto ground including canopy
 real(r8), pointer :: qflx_rain_grnd(:) !rain on ground after interception
 real(r8), pointer :: qflx_snow_grnd(:) !snow on ground after interception
 real(r8), pointer :: qflx_snowcap(:) !excess precipitation due to snow c
 real(r8), pointer :: qflx_evap_veg(:) !vegetation evaporation (mm H2O/s)
 real(r8), pointer :: qflx_tran_veg(:) !vegetation transpiration (mm H2O/s)
 real(r8), pointer :: qflx_evap_can(:) !evaporation from leaves and stems
 real(r8), pointer :: qflx_evap_soi(:) !soil evaporation (mm H2O/s) (+ = t
 real(r8), pointer :: qflx_evap_tot(:) !qflx_evap_soi + qflx_evap_veg + qf
 real(r8), pointer :: qflx_evap_grnd(:) !ground surface evaporation rate (m
 real(r8), pointer :: qflx_dew_grnd(:) !ground surface dew formation (mm H
 real(r8), pointer :: qflx_sub_snow(:) !sublimation rate from snow pack (m
 real(r8), pointer :: qflx_dew_snow(:) !surface dew added to snow pack (mm
end type pft_wflux_type

```

```

pft carbon flux variables structure

```

```

type pft_cflux_type
 real(r8), pointer :: psnsun(:) !sunlit leaf photosynthesis (umol C
 real(r8), pointer :: psnsha(:) !shaded leaf photosynthesis (umol C
 real(r8), pointer :: fpsn(:) !photosynthesis (umol CO2 /m**2 /s)
 real(r8), pointer :: frm(:) !total maintenance respiration (umo
 real(r8), pointer :: frmf(:) !leaf maintenance respiration (umo
 real(r8), pointer :: frms(:) !stem maintenance respiration (umo
 real(r8), pointer :: frmr(:) !root maintenance respiration (umo
 real(r8), pointer :: frg(:) !growth respiration (umol CO2 /m**2
 real(r8), pointer :: dmi(:) !total dry matter production (ug /m
 real(r8), pointer :: fco2(:) !net CO2 flux (umol CO2 /m**2 /s) [
 real(r8), pointer :: fmicr(:) !microbial respiration (umol CO2 /m
end type pft_cflux_type

```

```

pft nitrogen flux variables structure

```

```

type pft_nflux_type
 real(r8), pointer :: dummy_entry(:)
end type pft_nflux_type

```

```

pft VOC flux variables structure

type pft_vflux_type
 real(r8), pointer :: vocflx_tot(:) !total VOC flux into atmosphere [ug
 real(r8), pointer :: vocflx(:, :) !(nvoc) VOC flux [ug C m-2 h-1]
 real(r8), pointer :: vocflx_1(:) !vocflx(1) (for history output) [ug
 real(r8), pointer :: vocflx_2(:) !vocflx(2) (for history output) [ug
 real(r8), pointer :: vocflx_3(:) !vocflx(3) (for history output) [ug
 real(r8), pointer :: vocflx_4(:) !vocflx(4) (for history output) [ug
 real(r8), pointer :: vocflx_5(:) !vocflx(5) (for history output) [ug
end type pft_vflux_type

pft dust flux variables structure

type pft_dflux_type
 real(r8), pointer :: flx_mss_vrt_dst(:, :) ! (ndst) !surface dust emissi
 real(r8), pointer :: flx_mss_vrt_dst_tot(:) !total dust flux into atmosph
 real(r8), pointer :: vlc_trb(:, :) ! (ndst) turbulent deposition
 real(r8), pointer :: vlc_trb_1(:) !turbulent deposition velocit
 real(r8), pointer :: vlc_trb_2(:) !turbulent deposition velocit
 real(r8), pointer :: vlc_trb_3(:) !turbulent deposition velocit
 real(r8), pointer :: vlc_trb_4(:) !turbulent deposition velocit
end type pft_dflux_type

End definition of structures defined at the pft_type level

Begin definition of structures defined at the column_type level

column physical state variables structure

type column_pstate_type
 type(pft_pstate_type) :: pps_a !pft-level pstate variables aver
 integer , pointer :: snl(:) !number of snow layers
 integer , pointer :: isoicol(:) !soil color class
 real(r8), pointer :: bsw(:, :) !Clapp and Hornberger "b" (nlev
 real(r8), pointer :: watsat(:, :) !volumetric soil water at satur
 real(r8), pointer :: hksat(:, :) !hydraulic conductivity at satu
 real(r8), pointer :: sucsat(:, :) !minimum soil suction (mm) (nle
 real(r8), pointer :: csol(:, :) !heat capacity, soil solids (J/
 real(r8), pointer :: tkmg(:, :) !thermal conductivity, soil min
 real(r8), pointer :: tkdry(:, :) !thermal conductivity, dry soil
 real(r8), pointer :: tksatu(:, :) !thermal conductivity, saturate
 real(r8), pointer :: smpmin(:) !restriction for min of soil po
 real(r8), pointer :: gwc_thr(:) !threshold soil moisture based
 real(r8), pointer :: mss_frc_cly_vld(:) ![frc] Mass fraction clay limit

```

```

real(r8), pointer :: mbl_bsn_fct(:) !??
logical , pointer :: do_capsnow(:) !true => do snow capping
real(r8), pointer :: snowdp(:) !snow height (m)
real(r8), pointer :: snowage(:) !non dimensional snow age [-] (
real(r8), pointer :: frac_sno(:) !fraction of ground covered by
real(r8), pointer :: zi(:, :) !interface level below a "z" le
real(r8), pointer :: dz(:, :) !layer thickness (m) (-nlevsno
real(r8), pointer :: z(:, :) !layer depth (m) (-nlevsno+1:nl
real(r8), pointer :: frac_iceold(:, :) !fraction of ice relative to th
integer , pointer :: imelt(:, :) !flag for melting (=1), freezin
real(r8), pointer :: eff_porosity(:, :) !effective porosity = porosity
real(r8), pointer :: sfact(:) !term for implicit correction t
real(r8), pointer :: sfactmax(:) !maximim of "sfact"
real(r8), pointer :: emg(:) !ground emissivity
real(r8), pointer :: zOmg(:) !roughness length over ground,
real(r8), pointer :: zOhg(:) !roughness length over ground,
real(r8), pointer :: zOqg(:) !roughness length over ground,
real(r8), pointer :: htvp(:) !latent heat of vapor of water
real(r8), pointer :: beta(:) !coefficient of convective velo
real(r8), pointer :: zii(:) !convective boundary height [m]
real(r8), pointer :: albgrd(:, :) !ground albedo (direct) (numrad
real(r8), pointer :: albgrl(:, :) !ground albedo (diffuse) (numra
real(r8), pointer :: rootr_column(:, :) !effective fraction of roots in
real(r8), pointer :: wf(:) !soil water as frac. of whc for
end type column_pstate_type

```

---

column energy state variables structure

---

```

type column_estate_type
 type(pft_estate_type):: pes_a !pft-level energy state variabl
 real(r8), pointer :: t_grnd(:) !ground temperature (Kelvin)
 real(r8), pointer :: dt_grnd(:) !change in t_grnd, last iterati
 real(r8), pointer :: t_soisno(:, :) !soil temperature (Kelvin) (-n
 real(r8), pointer :: t_lake(:, :) !lake temperature (Kelvin) (1:
 real(r8), pointer :: tssbef(:, :) !soil/snow temperature before u
 real(r8), pointer :: t_snow(:) !vertically averaged snow tempe
 real(r8), pointer :: thv(:) !virtual potential temperature
 real(r8), pointer :: thm(:) !intermediate variable (forc_t+
end type column_estate_type

```

---

column water state variables structure

---

```

type column_wstate_type
 type(pft_wstate_type):: pws_a !pft-level water state variables
 real(r8), pointer :: h2osno(:) !snow water (mm H2O)
 real(r8), pointer :: h2osoi_liq(:, :) !liquid water (kg/m2) (new) (-n
 real(r8), pointer :: h2osoi_ice(:, :) !ice lens (kg/m2) (new) (-nlevs
 real(r8), pointer :: h2osoi_vol(:, :) !volumetric soil water (0<=h2os
 real(r8), pointer :: h2osno_old(:) !snow mass for previous time st
 real(r8), pointer :: qg(:) !ground specific humidity [kg/k
 real(r8), pointer :: dqgdT(:) !d(qg)/dT
 real(r8), pointer :: snowice(:) !average snow ice lens

```



```

 real(r8), pointer :: snowliq(:) !average snow liquid water
end type column_wstate_type

column carbon state variables structure

type column_cstate_type
 type(pft_cstate_type):: pcs_a !pft-level carbon state variables ave
 real(r8), pointer :: soilc(:) !soil carbon (kg C /m**2)
end type column_cstate_type

column nitrogen state variables structure

type column_nstate_type
 type(pft_nstate_type):: pns_a !pft-level nitrogen state variables a
end type column_nstate_type

column VOC state variables structure

type column_vstate_type
 type(pft_vstate_type):: pvs_a !pft-level VOC state variables averag
end type column_vstate_type

column DGVM state variables structure

type column_dgvstate_type
 type(pft_dgvstate_type):: pdgvs_a
end type column_dgvstate_type

column dust state variables structure

type column_dstate_type
 real(r8), pointer :: dummy_entry(:)
end type column_dstate_type

column energy flux variables structure

type column_eflux_type
 type(pft_eflux_type):: pef_a !pft-level energy flux variables averaged to
 real(r8), pointer :: eflx_snomelt(:) !snow melt heat flux (W/m**2)
 real(r8), pointer :: eflx_impsoil(:) !implicit evaporation for soil temper
end type column_eflux_type

column momentum flux variables structure

type column_mflux_type
 type(pft_mflux_type):: pmf_a !pft-level momentum flux variables averag
end type column_mflux_type

```

```

column water flux variables structure

type column_wflux_type
 type(pft_wflux_type):: pwf_a !pft-level water flux variables averaged to t
 real(r8), pointer :: qflx_infl(:) !infiltration (mm H2O /s)
 real(r8), pointer :: qflx_surf(:) !surface runoff (mm H2O /s)
 real(r8), pointer :: qflx_drain(:) !sub-surface runoff (mm H2O /s)
 real(r8), pointer :: qflx_top_soil(:) !net water input into soil from top (
 real(r8), pointer :: qflx_snomelt(:) !snow melt (mm H2O /s)
 real(r8), pointer :: qflx_qrgwl(:) !qflx_surf at glaciers, wetlands, lak
 real(r8), pointer :: qmelt(:) !snow melt [mm/s]
end type column_wflux_type

column carbon flux variables structure

type column_cflux_type
 type(pft_cflux_type):: pcf_a !pft-level carbon flux variables aver
end type column_cflux_type

column nitrogen flux variables structure

type column_nflux_type
 type(pft_nflux_type):: pnf_a !pft-level nitrogen flux variables av
end type column_nflux_type

column VOC flux variables structure

type column_vflux_type
 type(pft_vflux_type):: pvf_a !pft-level VOC flux variables average
end type column_vflux_type

column dust flux variables structure

type column_dflux_type
 type(pft_dflux_type):: pdf_a !pft-level dust flux variables averag
end type column_dflux_type

End definition of structures defined at the column_type level

Begin definition of structures defined at the landunit_type level

landunit physical state variables structure
note - landunit type can be vegetated (includes bare soil), deep lake,

```

```

 shallow lake, wetland, glacier or urban

type landunit_pstate_type
 type(column_pstate_type):: cps_a !column-level physical state
end type landunit_pstate_type

 landunit energy state variables structure

type landunit_estate_type
 type(column_estate_type):: ces_a !column-level energy state varia
end type landunit_estate_type

 landunit water state variables structure

type landunit_wstate_type
 type(column_wstate_type):: cws_a !column-level water state varia
end type landunit_wstate_type

 landunit carbon state variables structure

type landunit_cstate_type
 type(column_cstate_type):: ccs_a !column-level carbon state vari
end type landunit_cstate_type

 landunit nitrogen state variables structure

type landunit_nstate_type
 type(column_nstate_type):: cns_a !column-level nitrogen state va
end type landunit_nstate_type

 landunit VOC state variables structure

type landunit_vstate_type
 type(column_vstate_type):: cvs_a !column-level VOC state variabl
end type landunit_vstate_type

 landunit DGVM state variables structure

type landunit_dgvstate_type
 real(r8):: dummy_entry
end type landunit_dgvstate_type

 landunit dust state variables structure

type landunit_dstate_type
 type(column_dstate_type):: cds_a !column-level dust state vari
end type landunit_dstate_type

```

```

 landunit energy flux variables structure

type landunit_eflux_type
 type(column_eflux_type):: cef_a !column-level energy flux var
end type landunit_eflux_type

 landunit momentum flux variables structure

type landunit_mflux_type
 type(pft_mflux_type):: pmf_a !pft-level momentum flux vari
end type landunit_mflux_type

 landunit water flux variables structure

type landunit_wflux_type
 type(column_wflux_type):: cwf_a !column-level water flux vari
end type landunit_wflux_type

 landunit carbon flux variables structure

type landunit_cflux_type
 type(column_cflux_type):: ccf_a !column-level carbon flux va
end type landunit_cflux_type

 landunit nitrogen flux variables structure

type landunit_nflux_type
 type(column_nflux_type):: cnf_a !column-level nitrogen flux v
end type landunit_nflux_type

 landunit VOC flux variables structure

type landunit_vflux_type
 type(pft_vflux_type):: pvf_a !pft-level VOC flux variable
end type landunit_vflux_type

 landunit dust flux variables structure

type landunit_dflux_type
 type(pft_dflux_type):: pdf_a !pft-level dust flux variabl
end type landunit_dflux_type

 End definition of structures defined at the landunit_type level


```

```


Begin definition of structures defined at the gridcell_type level

gridcell physical state variables structure

type gridcell_pstate_type
 type(column_pstate_type):: cps_a !column-level physical state variables
 real(r8), pointer :: wtfact(:) !Fraction of model area with high water
end type gridcell_pstate_type

atmosphere -> land state variables structure

type atm2lnd_state_type
#if (defined OFFLINE)
 real(r8), pointer :: flfall(:) !fraction of liquid water within fall
#endif
 real(r8), pointer :: forc_t(:) !atmospheric temperature (Kelvin)
 real(r8), pointer :: forc_u(:) !atmospheric wind speed in east direc
 real(r8), pointer :: forc_v(:) !atmospheric wind speed in north dire
 real(r8), pointer :: forc_wind(:) !atmospheric wind speed
 real(r8), pointer :: forc_q(:) !atmospheric specific humidity (kg/kg)
 real(r8), pointer :: forc_hgt(:) !atmospheric reference height (m)
 real(r8), pointer :: forc_hgt_u(:) !observational height of wind [m] (ne
 real(r8), pointer :: forc_hgt_t(:) !observational height of temperature
 real(r8), pointer :: forc_hgt_q(:) !observational height of humidity [m]
 real(r8), pointer :: forc_pbot(:) !atmospheric pressure (Pa)
 real(r8), pointer :: forc_th(:) !atmospheric potential temperature (K
 real(r8), pointer :: forc_vp(:) !atmospheric vapor pressure (Pa)
 real(r8), pointer :: forc_rho(:) !density (kg/m**3)
 real(r8), pointer :: forc_co2(:) !atmospheric CO2 concentration (Pa)
 real(r8), pointer :: forc_o2(:) !atmospheric O2 concentration (Pa)
 real(r8), pointer :: forc_psrp(:) !surface pressure (Pa)
end type atm2lnd_state_type

land -> atmosphere state variables structure

type lnd2atm_state_type
 real(r8), pointer :: t_rad(:) !radiative temperature (Kelvin)
 real(r8), pointer :: t_ref2m(:) !2 m height surface air temperature (
 real(r8), pointer :: q_ref2m(:) !2 m height surface specific humidity
 real(r8), pointer :: h2osno(:) !snow water (mm H2O)
 real(r8), pointer :: albd(:, :) !(numrad) surface albedo (direct)
 real(r8), pointer :: albi(:, :) !(numrad) surface albedo (diffuse)
end type lnd2atm_state_type

gridcell energy state variables structure

type gridcell_estate_type
 type(column_estate_type):: ces_a !column-level energy state variables

```

```
end type gridcell_estate_type
```

```

 gridcell water state variables structure

```

```
type gridcell_wstate_type
 type(column_wstate_type):: cws_a !column-level water state variables a
end type gridcell_wstate_type
```

```

 gridcell carbon state variables structure

```

```
type gridcell_cstate_type
 type(column_cstate_type):: ccs_a !column-level carbon state variables
end type gridcell_cstate_type
```

```

 gridcell nitrogen state variables structure

```

```
type gridcell_nstate_type
 type(column_nstate_type):: cns_a !column-level nitrogen state variable
end type gridcell_nstate_type
```

```

 gridcell VOC state variables structure

```

```
type gridcell_vstate_type
 type(column_vstate_type):: cvs_a !column-level VOC state variables ave
end type gridcell_vstate_type
```

```

 gridcell dust state variables structure

```

```
type gridcell_dstate_type
 type(column_dstate_type):: cds_a !column-level dust state variables av
end type gridcell_dstate_type
```

```

 gridcell DGVM state variables structure

```

```
type gridcell_dgvstate_type
 real(r8), pointer :: afirefrac(:) ! fraction of gridcell affected by fir
 real(r8), pointer :: acfluxfire(:) ! C flux to atmosphere from biomass bu
 real(r8), pointer :: bmfm(:, :) ! biomass (NPP) for each naturally-veg
 real(r8), pointer :: afmicr(:, :) ! microbial respiration (Rh) for each
 real(r8), pointer :: begwater(:)
 real(r8), pointer :: endwater(:)
 real(r8), pointer :: begenergy(:)
 real(r8), pointer :: endenergy(:)
end type gridcell_dgvstate_type
```

```

 atmosphere -> land flux variables structure

```

```

type atm2lnd_flux_type
 real(r8), pointer :: forc_lwrad(:) !downward infrared (longwave)
 real(r8), pointer :: forc_solad(:, :) !direct beam radiation (vis=f
 real(r8), pointer :: forc_solai(:, :) !diffuse radiation (vis=f
 real(r8), pointer :: forc_solar(:) !incident solar radiation
 real(r8), pointer :: forc_rain(:) !rain rate [mm/s]
 real(r8), pointer :: forc_snow(:) !snow rate [mm/s]
end type atm2lnd_flux_type

```

```

land -> atmosphere flux variables structure

```

```

type lnd2atm_flux_type
 real(r8), pointer :: taux(:) !wind stress: e-w (kg/m/s**2)
 real(r8), pointer :: tauy(:) !wind stress: n-s (kg/m/s**2)
 real(r8), pointer :: eflx_lh_tot(:) !total latent heat flux (W/m8
 real(r8), pointer :: eflx_sh_tot(:) !total sensible heat flux (W/
 real(r8), pointer :: eflx_lwrad_out(:) !emitted infrared (longwave)
 real(r8), pointer :: qflx_evap_tot(:) !qflx_evap_soi + qflx_evap_ve
 real(r8), pointer :: fsa(:) !solar radiation absorbed (to
end type lnd2atm_flux_type

```

```

gridcell energy flux variables structure

```

```

type gridcell_eflux_type
 type(column_eflux_type):: cef_a !column-level energy flux var
end type gridcell_eflux_type

```

```

gridcell momentum flux variables structure

```

```

type gridcell_mflux_type
 type(pft_mflux_type):: pmf_a !pft-level momentum flux vari
end type gridcell_mflux_type

```

```

gridcell water flux variables structure

```

```

type gridcell_wflux_type
 FT0
 real(r8), pointer :: qchan2(:) !history file RTM river (chan
 real(r8), pointer :: qchocn2(:) !history file RTM river (chan
 FT0
 type(column_wflux_type):: cwf_a !column-level water flux vari
end type gridcell_wflux_type

```

```

gridcell carbon flux variables structure

```

```

type gridcell_cflux_type
 type(column_cflux_type):: ccf_a !column-level carbon flux var
end type gridcell_cflux_type

```

```

 gridcell nitrogen flux variables structure

type gridcell_nflux_type
 type(column_nflux_type):: cnf_a !column-level nitrogen flux v
end type gridcell_nflux_type

 gridcell VOC flux variables structure

type gridcell_vflux_type
 type(pft_vflux_type):: pvf_a !pft-level VOC flux variables
end type gridcell_vflux_type

 gridcell dust flux variables structure

type gridcell_dflux_type
 type(pft_dflux_type):: pdf_a !pft-level dust flux variable
end type gridcell_dflux_type

 End definition of structures defined at the gridcell_type level

 Begin definition of structures defined at the CLM level

 CLM physical state variables structure

type model_pstate_type
 type(column_pstate_type) :: cps_a !column-level physical state variable
end type model_pstate_type

 CLM energy state variables structure

type model_estate_type
 type(column_estate_type):: ces_a !column-level energy state va
end type model_estate_type

 CLM water state variables structure

type model_wstate_type
 type(column_wstate_type):: cws_a !column-level water state var
end type model_wstate_type

 CLM carbon state variables structure

type model_cstate_type

```



```

 type(column_cstate_type):: ccs_a !column-level carbon state va
end type model_cstate_type

CLM nitrogen state variables structure

type model_nstate_type
 type(column_nstate_type):: cns_a !column-level nitrogen state
end type model_nstate_type

CLM VOC state variables structure

type model_vstate_type
 type(column_vstate_type):: cvs_a !column-level VOC state varia
end type model_vstate_type

CLM dust state variables structure

type model_dstate_type
 type(column_dstate_type):: cds_a !column-level dust state vari
end type model_dstate_type

CLM energy flux variables structure

type model_eflux_type
 type(column_eflux_type):: cef_a !column-level energy flux var
end type model_eflux_type

CLM momentum flux variables structure

type model_mflux_type
 type(pft_mflux_type):: pmf_a !pft-level momentum flux vari
end type model_mflux_type

CLM water flux variables structure

type model_wflux_type
 type(column_wflux_type):: cwf_a !column-level water flux vari
end type model_wflux_type

CLM carbon flux variables structure

type model_cflux_type
 type(column_cflux_type):: ccf_a !column-level carbon flux var
end type model_cflux_type

CLM nitrogen flux variables structure

```

```

type model_nflux_type
 type(column_nflux_type):: cnf_a !column-level nitrogen flux v
end type model_nflux_type

CLM VOC flux variables structure

type model_vflux_type
 type(pft_vflux_type):: pvf_a !pft-level VOC flux variables
end type model_vflux_type

CLM dust flux variables structure

type model_dflux_type
 type(pft_dflux_type):: pdf_a !pft-level dust flux variable
end type model_dflux_type

End definition of structures defined at the model_type level

Begin definition of spatial scaling hierarchy

define the pft structure

type pft_type
 ! indices into higher levels in hierarchy
 integer, pointer :: column(:) !index into column level quantities
 integer, pointer :: landunit(:) !index into landunit level quantities
 integer, pointer :: gridcell(:) !index into gridcell level quantities

 ! topological mapping functionality
 integer , pointer :: itype(:) !pft vegetation
 real(r8), pointer :: area(:) !total land area for this pft (km^2)
 real(r8), pointer :: wtcol(:) !weight (relative to column) for this
 real(r8), pointer :: wtlunit(:) !weight (relative to landunit) for th
 real(r8), pointer :: wtgcell(:) !weight (relative to gridcell) for th
 integer , pointer :: ixy(:) !xy lon index
 integer , pointer :: jxy(:) !xy lat index
 integer , pointer :: mxy(:) !m index for laixy(i,j,m),etc.
 integer , pointer :: snindex(:) !corresponding pft index in s->n and
 real(r8), pointer :: latdeg(:) !latitude (degrees)
 real(r8), pointer :: londeg(:) !longitude (degrees)

 ! conservation check structures for the pft level

```

```

type(energy_balance_type) :: pebal !energy balance structure
type(water_balance_type) :: pwbal !water balance structure
type(carbon_balance_type) :: pcbal !carbon balance structure
type(nitrogen_balance_type) :: pnbal !nitrogen balance structure

! DGVM state variables
type(pft_dgvstate_type) :: pdgvs !pft DGVM state variables

! state variables defined at the pft level
type(pft_pstate_type) :: pps !physical state variables
type(pft_estate_type) :: pes !pft energy state
type(pft_wstate_type) :: pws !pft water state
type(pft_cstate_type) :: pcs !pft carbon state
type(pft_nstate_type) :: pns !pft nitrogen state
type(pft_vstate_type) :: pvs !pft VOC state

! flux variables defined at the pft level
type(pft_eflux_type) :: pef !pft energy flux
type(pft_mflux_type) :: pmf !pft momentum flux
type(pft_wflux_type) :: pwf !pft water flux
type(pft_cflux_type) :: pcf !pft carbon flux
type(pft_nflux_type) :: pnf !pft nitrogen flux
type(pft_vflux_type) :: pvf !pft VOC flux
type(pft_dflux_type) :: pdf !pft dust flux
end type pft_type

define the column structure

type column_type
! lower levels in hierarchy
type(pft_type) :: p !plant functional type (pft) data str
integer , pointer :: pfti(:) !beginning pft index for each column
integer , pointer :: pftf(:) !ending pft index for each column
integer , pointer :: npfts(:) !number of pfts for each column

! higher level in hierarchy
integer , pointer :: landunit(:) !index into landunit level quantities
integer , pointer :: gridcell(:) !index into gridcell level quantities

! topological mapping functionality
integer , pointer :: itype(:) !column type
real(r8), pointer :: area(:) !total land area for this column (km^
real(r8), pointer :: wtgcell(:) !weight (relative to gridcell) for th
real(r8), pointer :: wtlunit(:) !weight (relative to landunit) for th
integer , pointer :: ixy(:) !xy lon index
integer , pointer :: jxy(:) !xy lat index
integer , pointer :: snindex(:) !corresponding column index in s->n a
real(r8), pointer :: latdeg(:) !latitude (degrees)
real(r8), pointer :: londeg(:) !longitude (degrees)

! conservation check structures for the column level
type(energy_balance_type) :: cebal !energy balance structure

```

```

type(water_balance_type) :: cwbal !water balance structure
type(carbon_balance_type) :: ccbal !carbon balance structure
type(nitrogen_balance_type) :: cnbal !nitrogen balance structure

! state variables defined at the column level
type(column_pstate_type) :: cps !column physical state variables
type(column_estate_type) :: ces !column energy state
type(column_wstate_type) :: cws !column water state
type(column_cstate_type) :: ccs !column carbon state
type(column_nstate_type) :: cns !column nitrogen state
type(column_vstate_type) :: cvs !column VOC state
type(column_dstate_type) :: cds !column dust state

! flux variables defined at the column level
type(column_elflux_type) :: cef !column energy flux
type(column_mflux_type) :: cmf !column momentum flux
type(column_wflux_type) :: cwf !column water flux
type(column_cflux_type) :: ccf !column carbon flux
type(column_nflux_type) :: cnf !column nitrogen flux
type(column_vflux_type) :: cvf !column VOC flux
type(column_dflux_type) :: cdf !column dust flux

! dgvm variables defined at the column level
type(column_dgvstate_type) :: cdgvs !column DGVM structure
end type column_type

define the geomorphological land unit structure

type landunit_type
! lower levels in hierarchy
type(column_type) :: c !column data structure (soil/snow/ca
integer, pointer :: coli(:) !beginning column index for each lan
integer, pointer :: colf(:) !ending column index for each landun
integer, pointer :: ncolumns(:) !number of columns for each landunit
integer, pointer :: pfti(:) !beginning pft index for each landun
integer, pointer :: pftf(:) !ending pft index for each landunit
integer, pointer :: npfts(:) !number of pfts for each landunit

! higher level in hierarchy
integer, pointer :: gridcell(:) !index into gridcell level quantitie

! topological mapping functionality
integer , pointer :: itype(:) !landunit type
real(r8), pointer :: area(:) !total land area for this landunit (
real(r8), pointer :: wtgcell(:) !weight (relative to gridcell) for t
integer , pointer :: ixy(:) !xy lon index
integer , pointer :: jxy(:) !xy lat index
integer , pointer :: snindex(:) !corresponding landunit index in s->
real(r8), pointer :: latdeg(:) !latitude (degrees)
real(r8), pointer :: londeg(:) !longitude (degrees)
logical , pointer :: ifspecial(:) !BOOL: true=>landunit is not vegetat
logical , pointer :: lakpoi(:) !BOOL: true=>lake point

```

```

! conservation check structures for the landunit level
type(energy_balance_type) :: lebal !energy balance structure
type(water_balance_type) :: lwbal !water balance structure
type(carbon_balance_type) :: lcbal !carbon balance structure
type(nitrogen_balance_type) :: lnbal !nitrogen balance structure

! state variables defined at the land unit level
type(landunit_pstate_type) :: lps !land unit physical state variables
type(landunit_estate_type) :: les !average of energy states over all c
type(landunit_wstate_type) :: lws !average of water states over all co
type(landunit_cstate_type) :: lcs !average of carbon states over all c
type(landunit_nstate_type) :: lns !average of nitrogen states over all
type(landunit_vstate_type) :: lvs !average of VOC states over all colu
type(landunit_dstate_type) :: lds !average of dust states over all col

! flux variables defined at the landunit level
type(landunit_elflux_type) :: lef !average of energy fluxes over all c
type(landunit_mflux_type) :: lmf !average of momentum fluxes over all
type(landunit_wflux_type) :: lwf !average of water fluxes over all co
type(landunit_cflux_type) :: lcf !average of carbon fluxes over all c
type(landunit_nflux_type) :: lnf !average of nitrogen fluxes over all
type(landunit_vflux_type) :: lvf !average of VOC fluxes over all colu
type(landunit_dflux_type) :: ldf !average of dust fluxes over all col
end type landunit_type

```

```

define the gridcell structure

```

```

type gridcell_type
! lower level in hierarchy
type(landunit_type) :: l !geomorphological landunits
integer, pointer :: luni(:) !beginning landunit index for each
integer, pointer :: lunf(:) !ending landunit index for each gri
integer, pointer :: nlandunits(:) !number of landunit for each gridce
integer, pointer :: coli(:) !beginning column index for each gr
integer, pointer :: colf(:) !ending column index for each gridc
integer, pointer :: ncolumns(:) !number of columns for each gridcel
integer, pointer :: pfti(:) !beginning pft index for each gridc
integer, pointer :: pftf(:) !ending pft index for each gridcell
integer, pointer :: npfts(:) !number of pfts for each gridcell

! topological mapping functionality
integer , pointer :: itype(:) !gridcell type
real(r8), pointer :: area(:) !total land area for this gridcell
real(r8), pointer :: wtglob(:) !weight for this gridcell relative
integer , pointer :: ixy(:) !xy lon index
integer , pointer :: jxy(:) !xy lat index
integer , pointer :: snindex(:) !corresponding gridcell index in s-
real(r8), pointer :: lat(:) !latitude (radians)
real(r8), pointer :: lon(:) !longitude (radians)
real(r8), pointer :: latdeg(:) !latitude (degrees)
real(r8), pointer :: londeg(:) !longitude (degrees)

```

```

real(r8), pointer :: landfrac(:) !fractional land for this gridcell

! conservation check structures for the gridcell level
type(energy_balance_type) :: gebal !energy balance structure
type(water_balance_type) :: gwbal !water balance structure
type(carbon_balance_type) :: gcbal !carbon balance structure
type(nitrogen_balance_type):: gnbal !nitrogen balance structure

! dgvm variables defined at the gridcell level
type(gridcell_dgvstate_type):: gdgvs !gridcell DGVM structure

! state variables defined at the gridcell level
type(gridcell_pstate_type) :: gps !gridcell physical state variables
type(gridcell_estate_type) :: ges !average of energy states over all l
type(gridcell_wstate_type) :: gws !average of water states over all la
type(gridcell_cstate_type) :: gcs !average of carbon states over all l
type(gridcell_nstate_type) :: gns !average of nitrogen states over all
type(gridcell_vstate_type) :: gvs !average of VOC states over all land
type(gridcell_dstate_type) :: gds !average of dust states over all lan
type(atm2lnd_state_type) :: a2ls !atmospheric state variables require
type(lnd2atm_state_type) :: l2as !land state variables required by th

! flux variables defined at the gridcell level
type(gridcell_eflux_type) :: gef !average of energy fluxes over all
type(gridcell_wflux_type) :: gwf !average of water fluxes over all l
type(gridcell_cflux_type) :: gcf !average of carbon fluxes over all
type(gridcell_nflux_type) :: gnf !average of nitrogen fluxes over al
type(gridcell_vflux_type) :: gvf !average of VOC fluxes over all lan
type(gridcell_dflux_type) :: gdf !average of dust fluxes over all la
type(atm2lnd_flux_type) :: a2lf !atmospheric flux variables require
type(lnd2atm_flux_type) :: l2af !land flux variables required by th
end type gridcell_type

define the top-level (model) structure

type model_type
! lower level in hierarch
type(gridcell_type) :: g !gridicell data structure
integer :: ngridcells !number of gridcells allocated for
real(r8) :: area !total land area for all gridcells

! conservation check structures for the clm (global) level
type(energy_balance_type) :: mebal !energy balance structure
type(water_balance_type) :: mwbal !water balance structure
type(carbon_balance_type) :: mcbal !carbon balnace structure
type(nitrogen_balance_type):: mnbal !nitrogen balance structure

! globally average state variables
type(model_pstate_type) :: mps !clm physical state variables
type(model_estate_type) :: mes !average of energy states over all g
type(model_wstate_type) :: mws !average of water states over all gr
type(model_cstate_type) :: mcs !average of carbon states over all g

```

```

type(model_nstate_type) :: mns !average of nitrogen states over all
type(model_vstate_type) :: mvs !average of VOC states over all grid
type(model_dstate_type) :: mds !average of dust states over all gri

! globally averaged flux variables
type(model_eflux_type) :: mef !average of energy fluxes over all g
type(model_wflux_type) :: mwf !average of water fluxes over all gr
type(model_cflux_type) :: mcf !average of carbon fluxes over all g
type(model_nflux_type) :: mnf !average of nitrogen fluxes over all
type(model_vflux_type) :: mvf !average of VOC fluxes over all grid
type(model_dflux_type) :: mdf !average of dust fluxes over all gri
end type model_type

End definition of spatial scaling hierarchy

Declare single instance of clmtype

type(model_type), target, save :: clm3

Declare single instance of array of ecophysiological constant types

type(pft_epc_type), target, save :: pftcon

Declare single instance of array of dgvm ecophysiological constant types

type(pft_dgvepc_type), target, save :: dgvm_pftcon

character(len=8), parameter :: nameg = 'gridcell' ! name of gridcells
character(len=8), parameter :: namel = 'landunit' ! name of landunits
character(len=8), parameter :: namec = 'column' ! name of columns
character(len=8), parameter :: namep = 'pft' ! name of pfts
character(len=8), parameter :: ocnrof = 'ocnrof' ! name of river routing o
character(len=8), parameter :: lndrof = 'lndrof' ! name of river routing l

```

## A.47 Module clmtypeInitMod

### (Source File: clmtypeInitMod.F90)

Allocate clmtype components and initialize them to signaling NaN.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use nanMod
use clmtype
use clm_varpar, only: maxpatch_pft
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: initClmtype
```

REVISION HISTORY:

Created by Peter Thornton and Mariana Vertenstein

!PRIVATE MEMBER FUNCTIONS:

```
private :: init_pft_type
private :: init_column_type
private :: init_landunit_type
private :: init_gridcell_type
private :: init_energy_balance_type
private :: init_water_balance_type
private :: init_pft_pstate_type
private :: init_pft_ecophys_constants
private :: init_pft_DGVMecophys_constants
private :: init_pft_estate_type
private :: init_pft_wstate_type
private :: init_pft_pdgvstate_type
private :: init_pft_eflux_type
private :: init_pft_mflux_type
private :: init_pft_wflux_type
private :: init_pft_cflux_type
private :: init_pft_vflux_type
private :: init_pft_dflux_type
private :: init_column_pstate_type
private :: init_column_estate_type
private :: init_column_wstate_type
private :: init_column_cstate_type
private :: init_column_eflux_type
private :: init_column_wflux_type
private :: init_landunit_pstate_type
private :: init_gridcell_pstate_type
private :: init_atm2lnd_state_type
private :: init_lnd2atm_state_type
private :: init_atm2lnd_flux_type
private :: init_lnd2atm_flux_type
private :: init_gridcell_wflux_type
```

---



### A.47.1 `initClmtype`

#### INTERFACE:

```
subroutine initClmtype()
```

#### DESCRIPTION:

```
Initialize clmtype components to signaling nan
The following clmtype components should NOT be initialized here
since they are set in routine clm_map which is called before this
routine is invoked
 %area, %wt, %wtlnd, %wtxy, %ixy, %jxy, %mxy, %snindex
 %ifspecial, %ityplun, %itype
 %pfti, %pftf, %pftn
 %coli, %colf, %coln
 %luni, %lunf, %lunn
```

#### USES:

```
use decompMod, only : get_proc_bounds, get_proc_global
```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

```
Created by Mariana Vertenstein
```

---

### A.47.2 `init_pft_type`

#### INTERFACE:

```
subroutine init_pft_type (beg, end, p)
```

#### DESCRIPTION:

```
Initialize components of pft_type structure
```

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(pft_type), intent(inout):: p
```

#### REVISION HISTORY:

```
Created by Mariana Vertenstein
```

---

### A.47.3 init\_column\_type

INTERFACE:

```
subroutine init_column_type (beg, end, c)
```

DESCRIPTION:

Initialize components of column\_type structure

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(column_type), intent(inout):: c
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.4 init\_landunit\_type

INTERFACE:

```
subroutine init_landunit_type (beg, end,l)
```

DESCRIPTION:

Initialize components of landunit\_type structure

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(landunit_type), intent(inout):: l
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.5 init\_gridcell\_type

INTERFACE:

```
subroutine init_gridcell_type (beg, end,g)
```

DESCRIPTION:

Initialize components of gridcell\_type structure

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(gridcell_type), intent(inout):: g
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.6 init\_energy\_balance\_type**

## INTERFACE:

```
subroutine init_energy_balance_type(beg, end, ebal)
```

## DESCRIPTION:

Initialize energy balance variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(energy_balance_type), intent(inout):: ebal
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.7 init\_water\_balance\_type**

## INTERFACE:

```
subroutine init_water_balance_type(beg, end, wbal)
```

## DESCRIPTION:

Initialize water balance variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type(water_balance_type), intent(inout):: wbal
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.8 init\_pft\_pstate\_type

INTERFACE:

```
subroutine init_pft_pstate_type(beg, end, pps)
```

DESCRIPTION:

Initialize pft physical state

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_pstate_type), intent(inout):: pps
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.9 init\_pft\_ecophys\_constants

INTERFACE:

```
subroutine init_pft_ecophys_constants()
```

DESCRIPTION:

Initialize pft physical state

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.10 init\_pft\_DGVMecophys\_constants

INTERFACE:

```
subroutine init_pft_DGVMecophys_constants()
```

DESCRIPTION:

Initialize pft physical state

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.11 `init_pft_energy_type`

INTERFACE:

```
subroutine init_pft_estate_type(beg, end, pes)
```

DESCRIPTION:

Initialize pft energy state

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_estate_type), intent(inout):: pes
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.12 `init_pft_wstate_type`

INTERFACE:

```
subroutine init_pft_wstate_type(beg, end, pws)
```

DESCRIPTION:

Initialize pft water state

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_wstate_type), intent(inout):: pws !pft water state
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.13 `init_pft_pdgvstate_type`

INTERFACE:

```
subroutine init_pft_pdgvstate_type(beg, end, pdgvs)
```

DESCRIPTION:

Initialize pft DGVM state variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_dgvstate_type), intent(inout):: pdgvs
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.14 init\_pft\_elflux\_type**

## INTERFACE:

```
subroutine init_pft_elflux_type(beg, end, pef)
```

## DESCRIPTION:

Initialize pft energy flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_elflux_type), intent(inout):: pef
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.15 init\_pft\_mflux\_type**

## INTERFACE:

```
subroutine init_pft_mflux_type(beg, end, pmf)
```

## DESCRIPTION:

Initialize pft momentum flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_mflux_type), intent(inout) :: pmf
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.16 `init_pft_wflux_type`

INTERFACE:

```
subroutine init_pft_wflux_type(beg, end, pwf)
```

DESCRIPTION:

Initialize pft water flux variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_wflux_type), intent(inout) :: pwf
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.17 `init_pft_cflux_type`

INTERFACE:

```
subroutine init_pft_cflux_type(beg, end, pcf)
```

DESCRIPTION:

Initialize pft carbon flux variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_cflux_type), intent(inout) :: pcf
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.18 `init_pft_vflux_type`

INTERFACE:

```
subroutine init_pft_vflux_type(beg, end, pvf)
```

DESCRIPTION:

Initialize pft VOC flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_vflux_type), intent(inout) :: pvf
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.19 init\_pft\_dflux\_type**

## INTERFACE:

```
subroutine init_pft_dflux_type(beg, end, pdf)
```

## DESCRIPTION:

Initialize pft dust flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (pft_dflux_type), intent(inout):: pdf
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.20 init\_column\_pstate\_type**

## INTERFACE:

```
subroutine init_column_pstate_type(beg, end, cps)
```

## DESCRIPTION:

Initialize column physical state variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_pstate_type), intent(inout):: cps
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---



### A.47.21 `init_column_estate_type`

INTERFACE:

```
subroutine init_column_estate_type(beg, end, ces)
```

DESCRIPTION:

Initialize column energy state variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_estate_type), intent(inout):: ces
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.22 `init_column_wstate_type`

INTERFACE:

```
subroutine init_column_wstate_type(beg, end, cws)
```

DESCRIPTION:

Initialize column water state variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_wstate_type), intent(inout):: cws !column water state
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.23 `init_column_cstate_type`

INTERFACE:

```
subroutine init_column_cstate_type(beg, end, ccs)
```

DESCRIPTION:

Initialize column carbon state variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_cstate_type), intent(inout):: ccs
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.24 init\_column\_elflux\_type**

## INTERFACE:

```
subroutine init_column_elflux_type(beg, end, cef)
```

## DESCRIPTION:

Initialize column energy flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_elflux_type), intent(inout):: cef
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.25 init\_column\_wflux\_type**

## INTERFACE:

```
subroutine init_column_wflux_type(beg, end, cwf)
```

## DESCRIPTION:

Initialize column water flux variables

## ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (column_wflux_type), intent(inout):: cwf
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.26 init\_landunit\_pstate\_type**

INTERFACE:

```
subroutine init_landunit_pstate_type(beg, end, lps)
```

DESCRIPTION:

Initialize landunit physical state variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (landunit_pstate_type), intent(inout):: lps
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.27 init\_gridcell\_dgvstate\_type**

INTERFACE:

```
subroutine init_gridcell_dgvstate_type(beg, end, gps)
```

DESCRIPTION:

Initialize gridcell DGVM variables

ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (gridcell_dgvstate_type), intent(inout):: gps
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.28 init\_gridcell\_pstate\_type**

INTERFACE:

```
subroutine init_gridcell_pstate_type(beg, end, gps)
```

DESCRIPTION:

Initialize gridcell physical state variables

## ARGUMENTS:

```

implicit none
integer, intent(in) :: beg, end
type (gridcell_pstate_type), intent(inout):: gps

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.29 init\_atm2lnd\_state\_type**

## INTERFACE:

```

subroutine init_atm2lnd_state_type(beg, end, a2ls)

```

## DESCRIPTION:

Initialize atmospheric state variables required by the land

## ARGUMENTS:

```

implicit none
integer, intent(in) :: beg, end
type (atm2lnd_state_type), intent(inout):: a2ls

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.47.30 init\_lnd2atm\_state\_type**

## INTERFACE:

```

subroutine init_lnd2atm_state_type(beg, end, l2as)

```

## DESCRIPTION:

Initialize land state variables required by the atmosphere

## ARGUMENTS:

```

implicit none
integer, intent(in) :: beg, end
type (lnd2atm_state_type), intent(inout):: l2as

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.31 `init_atm2lnd_flux_type`

#### INTERFACE:

```
subroutine init_atm2lnd_flux_type(beg, end, a2lf)
```

#### DESCRIPTION:

Initialize atmospheric fluxes required by the land

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (atm2lnd_flux_type), intent(inout):: a2lf
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.32 `init_lnd2atm_flux_type`

#### INTERFACE:

```
subroutine init_lnd2atm_flux_type(beg, end, l2af)
```

#### DESCRIPTION:

Initialize land fluxes required by the atmosphere

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (lnd2atm_flux_type), intent(inout):: l2af
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.47.33 `init_gridcell_wflux_type`

#### INTERFACE:

```
subroutine init_gridcell_wflux_type(beg, end, gwf)
```

#### DESCRIPTION:

Initialize gridcell water flux variables

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: beg, end
type (gridcell_wflux_type), intent(inout):: gwf
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

## A.48 Module controlMod

### (Source File: controlMod.F90)

Module which initializes run control variables. The following possible namelist variables are set default values and possibly read in on startup

```

=== define run =====
 o caseid = 256 character case name
 o ctitle = 256 character case title
 o nsrest = integer flag. 0: initial run. 1: restart: 3: branch
=== model time =====
 o dtime = integer model time step (s)
 o calendar = Calendar to use in date calculations.
 'no_leap' (default) or 'gregorian'
 o start_ymd = Starting date for run encoded in yearmdd format.
 Default value is read from initial conditions file.
 o start_tod = Starting time of day for run in seconds since 0Z.
 Default value is read from initial conditions file.
 o stop_ymd = Stopping date for run encoded in yearmdd format.
 No default.
 o stop_tod = Stopping time of day for run in seconds since 0Z.
 Default: 0.
 o nelapse = nnn, Specify the ending time for the run as an interval
 starting at the current time in either timesteps
 (if positive) or days (if negative).
 Either nestep or (stop_ymd,stop_tod) take precedence.
 o nestep = nnnn, Specify the ending time for the run as an interval
 starting at (start_ymd,start_tod) in either timesteps
 (if positive) or days (if negative).
 (stop_ymd,stop_tod) takes precedence if set.
 o ref_ymd = Reference date for time coordinate encoded in yearmdd fo
 Default value is start_ymd.
 o ref_tod = Reference time of day for time coordinate in seconds sinc
 Default value is start_tod.
=== input data ===
 o finidat = 256 character initial conditions file name
 o fsurdatt = 256 character surface data file name
 o fpftcon = 256 character data file with PFT physiological const
 o frivinp_rtm = 256 character input data file for rtm
 o nreversn = 256 character restart file name for use with branch
=== offline forcing data ===
 o offline_atmdir = 256 character directory for input atm data files (ca
=== input data when making surface data [fsurdatt] ===
 o mksrf_offline_fgrid = offline - land grid dataset to use instead of
 o mksrf_offline_fnavyoro = offline - 20 min navy orography dataset
 o mksrf_offline_edgen = offline - northern edge of grid (degrees): >
 o mksrf_offline_edgee = offline - eastern edge of grid (degrees): see
 o mksrf_offline_edges = offline - southern edge of grid (degrees): >=
 o mksrf_offline_edgew = offline - western edge of grid (degrees): see
 o mksrf_fvegtyp = 256 character vegetation type data file name
 o mksrf_fsoitex = 256 character soil texture data file name
 o mksrf_fsoicol = 256 character soil color data file name
 o mksrf_flanwat = 256 character inland water data file name
 o mksrf_furban = 256 character urban data file name
 o mksrf_fglacier = 256 character glacier data file name

```

```

 o mksrf_flai = 256 character lai data file file name
=== history and restart files ===
 o hist_ndens = integer, can have value of 1 (nc_double) or 2 (nf_floa
 o hist_dov2xy = true if want grid-average history field (false = vecto
 o hist_nhtfrq = integer history interval (+ = iterations, - = hours,
 o hist_mfilt = integer number of time samples per history file
 o hist_fincl1 = 10 character name of fields for first auxillary histo
 o hist_fincl2 = 10 character name of fields for second auxillary histo
 o hist_fincl3 = 10 character name of fields for first auxillary histo
 o hist_fincl4 = 10 character name of fields for second auxillary histo
 o hist_fincl5 = 10 character name of fields for first auxillary histo
 o hist_fincl6 = 10 character name of fields for second auxillary histo
 o hist_fexcl1 = 8 character name of fields for first auxillary histo
 o hist_fexcl2 = 8 character name of fields for second auxillary histo
 o hist_fexcl3 = 8 character name of fields for first auxillary histo
 o hist_fexcl4 = 8 character name of fields for second auxillary histo
 o hist_fexcl5 = 8 character name of fields for first auxillary histo
 o hist_fexcl6 = 8 character name of fields for second auxillary histo
 o hist_crtinic = 8 character frequency to generate initial dataset
 ['6-HOURLY', 'DAILY', 'MONTHLY', 'YEARLY', 'NONE']
 o rpntpath = 256 character full UNIX pathname of the local restart
 This file must exist when the model is restarted.
 This file is overwritten every time new restart data f
=== long term archiving =====
 o archive_dir = 256 character long term archive directory (can be MSS di
 o mss_irt = integer mass store retention period (days)
 o mss_wpass = 8 character mass store write password for output data se
=== model physics ===
 o irad = integer solar radiation frequency (+ = iteration. - = h
 o wrtdia = true if want output written
 o csm_doflxave = true => flux averaging is to be performed (only used fo
=== rtm control variables ===
 o rtm_nsteps = if > 1, average rtm over rtm_nsteps time steps
When coupled to CAM: base calendar info, nstep, nestep, nsrest, and time
step are input to the land model from CAM. The values in the clmexp namelis
are not used. The minimum namelist parameters are:
 o fsurdat
 o finidat
 o fpftcon
When running in offline mode, the minimum namelist parameters are:
 o nsrest
 o nestep or nelapse
 o fsurdat
 o finidat
 o dtime

```

## USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varctl
use spmdMod
use shr_sys_mod, only : shr_sys_getenv
use decompMod , only : clump_pproc
use histFileMod, only : max_tapes, max_namlen, &
 hist_empty_htapes, hist_dov2xy, &

```

```

 hist_avgflag_pertape, hist_typeid_pertape, &
 hist_nhtfrq, hist_ndens, hist_mfilt, &
 hist_fincl1, hist_fincl2, hist_fincl3, &
 hist_fincl4, hist_fincl5, hist_fincl6, &
 hist_fexcl1, hist_fexcl2, hist_fexcl3, &
 hist_fexcl4, hist_fexcl5, hist_fexcl6
 use shr_const_mod, only : SHR_CONST_CDAY
 use abortutils, only : endrun
#if (defined COUP_CSM)
 use clm_csmMod, only : csm_dtime
#endif
#if (defined SCAM)
 use scamMod, only : lsmsurffile, lsminifile
#endif

```

## PUBLIC TYPES:

```

 implicit none
 save

```

## PUBLIC MEMBER FUNCTIONS:

```

 public :: control_init ! initial run control information
 public :: control_print ! print run control information

```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.48.1 control\_init

## INTERFACE:

```

 subroutine control_init (cam_caseid , cam_ctitle, cam_irad , cam_nsrest, &
 cam_crtinic, cam_nhtfrq, cam_mfilt, cam_irt)

```

## DESCRIPTION:

Initialize CLM run control information

## USES:

```

#if (defined OFFLINE) || (defined COUP_CSM)
 use time_manager, only : calendar, dtime, nestep, nelapse, start_ymd, &
 start_tod, stop_ymd, stop_tod, ref_ymd, ref_tod
#else
 use time_manager, only : get_step_size, is_perpetual
#endif

```

## ARGUMENTS:

```

 implicit none
 include 'netcdf.inc'

```



```

character(len=*), optional, intent(in) :: cam_caseid ! cam caseid
character(len=*), optional, intent(in) :: cam_ctitle ! cam title
integer , optional, intent(in) :: cam_irad ! cam radiation f
integer , optional, intent(in) :: cam_nsrest ! cam run type
character(len=*), optional, intent(in) :: cam_crtinic ! cam initial dat
integer , optional, intent(in) :: cam_nhtfrq ! cam history wri
integer , optional, intent(in) :: cam_mfilt ! cam number of f
integer , optional, intent(in) :: cam_irt ! cam mss retenti

```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.48.2 control\_spmd

#### INTERFACE:

```
subroutine control_spmd()
```

#### DESCRIPTION:

Distribute namelist data all processors. The cpp SPMD definition provides for the funnelling of all program i/o through the master processor. Processor 0 either reads restart/history data from the disk and distributes it to all processors, or collects data from all processors and writes it to disk.

#### USES:

```

#if (defined OFFLINE) || (defined COUP_CSM)
 use time_manager, only : calendar, dtime, nestep, nelapse, start_ymd, &
 start_tod, stop_ymd, stop_tod, ref_ymd, ref_tod
#endif
 use spmdMod, only : mpicom

```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.48.3 control\_print

#### INTERFACE:

```
subroutine control_print ()
```

#### DESCRIPTION:

Write out run control variables

ARGUMENTS:

implicit none

REVISION HISTORY:

Created by Mariana Vertenstein



## A.49 Module decompMod

### (Source File: decompMod.F90)

USES:

```

 use shr_kind_mod, only : r8 => shr_kind_r8
 use clm_varpar , only : lsmlon, lsmlat, maxpatch, maxpatch_pft, &
 npatch_crop, npatch_urban, npatch_glacier
 use clm_varsur , only : numlon, landmask, all_pfts_on_srfdat
#ifdef SPMD
#ifdef COUP_CAM
 use pmgrid , only : masterproc, iam
 use spmd_dyn , only : npes
#else
 use spmdMod , only : masterproc, iam, npes
#endif
#else
 use spmdMod , only : masterproc, iam, npes
#endif
 use shr_sys_mod , only : shr_sys_flush
 use abortutils, only : endrun

```

PUBLIC TYPES:

```

 implicit none
 integer, public :: clump_pproc ! number of clumps per MPI process

```

PUBLIC MEMBER FUNCTIONS:

```

 public initDecomp ! initializes land surface decomposition
 ! into clumps and processors
 public get_gcell_info ! updates gridcell, landunits, columns and
 ! pfts counters
 public get_gcell_xyind ! returns ixy and jxy for each grid cell
 public get_nclumps ! returns the number of clumps defined
 public get_clump_cell_id_coord ! returns clump/cell ids based on lon/lat
 public get_clump_owner_id ! returns clump owner based on clump id
 public get_clump_ncells_proc ! returns number of cells for process
 public get_clump_ncells_id ! returns number of cells in clump
 public get_clump_coord_id ! returns lon/lat coordinates based on id
 public get_clump_gcell_info ! returns id gridcell index
 public get_clump_bounds ! beg and end gridcell, landunit, column,
 ! pft indices for clump
 public get_proc_clumps ! number of clumps for this processor
 public get_proc_bounds ! beg and end gridcell, landunit, column,
 ! pft indices for this processor
 public get_proc_total ! total number of gridcells, landunits,
 ! columns and pfts for any processor
 public get_proc_global ! total gridcells, landunits, columns, pfts
 ! across all processors
 public get_sn_landid ! returns s->n gridcell indices for each
 ! s->n landunit
 public get_sn_colsid ! returns s->n gridcell or landunit indices
 ! for each s->n column
 public get_sn_pftsid ! returns s->n gridcell, landunit or column

```

```

public get_sn_index ! indiceces for s->n pft column
 ! returns s->n gridcell, landunit or column
 ! index given the corresponding dc index

save

```

## DESCRIPTION:

Module provides a descomposition into a clumped data structure which can be mapped back to atmosphere physics chunks.

## REVISION HISTORY:

2002.09.11 Forrest Hoffman Creation.

**A.49.1 initDecomp**

## INTERFACE:

```
subroutine initDecomp(wtxy)
```

## DESCRIPTION:

This subroutine initializes the land surface decomposition into a clump data structure.

## USES:

```
use clmtype
```

## ARGUMENTS:

```

implicit none
real(r8), intent(in) :: wtxy(lsm lon, lsmlat, maxpatch) ! subgrid patch
 ! weights

```

## LOCAL VARIABLES:

```

integer :: ppc ! min number of pfts per clump
integer :: lpc ! min number of landunits per clump
integer :: ppclump ! min pfts per clump
integer :: i,j,cid,pid ! indices
integer :: gi,li,ci,pi ! indices
integer :: gf,lf,cf,pf ! indices
integer :: g,l,c,p,n,m ! indices
integer :: gdc,gsn ! indices
integer :: nzero ! first clump with zero gridcells
integer :: ncells ! total gridcells
integer :: nlunits ! total landunits
integer :: ncols ! total columns
integer :: npfts ! total pfts
integer :: nveg ! number of pfts in vegetated landunit
integer :: numg ! total number of gridcells across all
 ! processors
integer :: numl ! total number of landunits across all
 ! processors

```

```

integer :: numc ! total number of columns across all
 ! processors
integer :: nump ! total number of pfts across all
 ! processors
logical, pointer :: clumpfull(:) ! true => clump is full
logical :: validclump ! temporary for finding full clump
logical :: error = .false. ! temporary for finding full clump
integer :: clumpcount ! temporary for finding full clump
integer :: ilunits, icols, ipfts ! temporaries
integer :: ng ! temporaries
integer :: nl ! temporaries
integer :: nc ! temporaries
integer :: np ! temporaries
integer :: ier ! error code

```

CALLED FROM:

```
subroutine initialize
```

REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
```

---

## A.49.2 get\_nclumps

INTERFACE:

```
integer function get_nclumps()
```

DESCRIPTION:

This function returns the number of clumps on the land model grid.

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine lp_coupling_init() in module lp_coupling (lp_coupling.F90)
```

REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
```

---

## A.49.3 get\_clump\_cell\_id\_coord

INTERFACE:

```
subroutine get_clump_cell_id_coord(lon, lat, cid, cell)
```

## DESCRIPTION:

This subroutine returns the id of the clump and cell corresponding to the longitude/latitude indices provided.

## ARGUMENTS:

```

 implicit none
 integer, intent(in) :: lon, lat ! longitude/latitude indices
 integer, intent(out) :: cid, cell ! clump and cell id

```

## CALLED FROM:

Unused.

## REVISION HISTORY:

2002.09.11 Forrest Hoffman Creation.

**A.49.4 get\_clump\_owner\_id**

## INTERFACE:

```

 integer function get_clump_owner_id(cid)

```

## DESCRIPTION:

This function returns the MPI process id (rank) responsible for the clump identified by the clump id cid.

## ARGUMENTS:

```

 implicit none
 integer, intent(in) :: cid ! clump id

```

## CALLED FROM:

```

 subroutine lp_coupling_init() in module lp_coupling (lp_coupling.F90)

```

## REVISION HISTORY:

2002.09.11 Forrest Hoffman Creation.

**A.49.5 get\_clump\_ncells\_proc**

## INTERFACE:

```

 integer function get_clump_ncells_proc(pid)

```

## DESCRIPTION:

This function returns the number of cells contained within clumps owned by process pid.

## ARGUMENTS:

```

 implicit none
 integer, intent(in) :: pid ! process id
LOCAL VARIABLES:
 integer :: cid ! loop indices
 integer :: ncells ! cell counter
CALLED FROM:
 subroutine clm_map() (clm_map.F90)
REVISION HISTORY:
 2002.09.11 Forrest Hoffman Creation.

```

---

### A.49.6 get\_clump\_ncells\_id

```

INTERFACE:
 integer function get_clump_ncells_id(cid)
REVISION HISTORY:
 2002.09.11 Forrest Hoffman Creation.
ARGUMENTS:
 implicit none
 integer, intent(in) :: cid ! clump id
CALLED FROM:
 subroutine lp_coupling_init() in module lp_coupling (lp_coupling.F90)
DESCRIPTION:

```

This function returns the number of cells contained within the clump identified by the clump id cid.

---

### A.49.7 get\_clump\_coord\_id

```

INTERFACE:
 subroutine get_clump_coord_id(cid, ncells, lons, lats)
DESCRIPTION:
 This subroutine returns the first ncells longitude/latitude indices for the clump identified by the clump id cid. In practice, this ncells is always the number of cells in clump cid: clumps(cid)
ARGUMENTS:
 implicit none
 integer, intent(in) :: cid ! clump id
 integer, intent(in) :: ncells ! number of grid cells
 integer, intent(out) :: lons(ncells) ! longitude indices
 integer, intent(out) :: lats(ncells) ! latitude indices

```



LOCAL VARIABLES:

```
integer :: i ! loop index
```

CALLED FROM:

```
subroutine lp_coupling_init() in module lp_coupling (lp_coupling.F90)
```

REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
```

---

### A.49.8 get\_gcell\_info

INTERFACE:

```
subroutine get_gcell_info (i, j, wtxy, nlunits, ncols, npfts, &
 nveg, wtveg, ncrop, wtcrop)
```

DESCRIPTION:

Obtain gridcell properties.

ARGUMENTS:

```
implicit none
integer , intent(in) :: i ! longitude index
integer , intent(in) :: j ! latitude index
real(r8), intent(in) :: wtxy(lsmlon, lsmlat, maxpatch) ! subgrid pft
 ! weights

integer , optional, intent(out) :: nlunits ! number of landunits
integer , optional, intent(out) :: ncols ! number of columns
integer , optional, intent(out) :: npfts ! number of pfts
integer , optional, intent(out) :: nveg ! number of vegetated pfts
 ! in naturally vegetated
 ! landunit

real(r8), optional, intent(out) :: wtveg ! weight (relative to
 ! gridcell) of naturally
 ! vegetated landunit

integer , optional, intent(out) :: ncrop ! number of crop pfts in
 ! crop landunit

real(r8), optional, intent(out) :: wtcrop ! weight (relative to
 ! gridcell) of crop landunit
```

CALLED FROM:

```
subroutines initDecomp
```

REVISION HISTORY:

```
2002.09.11 Mariana Vertenstein Creation.
```

---

**A.49.9 get\_clump\_gcell\_info**

## INTERFACE:

```
subroutine get_clump_gcell_info(cid, cell, gi)
```

## DESCRIPTION:

Determine beginning grid cell index

## ARGUMENTS:

```
implicit none
integer, intent(in) :: cid ! clump id
integer, intent(in) :: cell ! clump cell id
integer, intent(out) :: gi ! 1d gridcell index
```

## CALLED FROM:

subroutines alltoall\_clump\_to\_chunk\_init(), alltoall\_clump\_to\_chunk(), and  
alltoall\_chunk\_to\_clump() in module lp\_coupling (lp\_coupling.F90)

## REVISION HISTORY:

2002.11.17 Mariana Vertenstein Creation.

**A.49.10 get\_clump\_bounds**

## INTERFACE:

```
subroutine get_clump_bounds (n, begg, endg, begl, endl, begc, endc, &
 begp, endp)
```

## ARGUMENTS:

```
implicit none
integer, intent(in) :: n ! proc clump index
integer, intent(out) :: begp, endp ! clump beginning and ending
 ! pft indices
integer, intent(out) :: begc, endc ! clump beginning and ending
 ! column indices
integer, intent(out) :: begl, endl ! clump beginning and ending
 ! landunit indices
integer, intent(out) :: begg, endg ! clump beginning and ending
 ! gridcell indices
```

## DESCRIPTION:

Determine clump beginning and ending pft, column, landunit and gridcell indices.

## REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

**A.49.11 get\_proc\_bounds**

## INTERFACE:

```
subroutine get_proc_bounds (begg, endg, begl, endl, begc, endc, &
 begp, endp)
```

## ARGUMENTS:

```
implicit none
integer, intent(out) :: begp, endp ! proc beginning and ending
 ! pft indices
integer, intent(out) :: begc, endc ! proc beginning and ending
 ! column indices
integer, intent(out) :: begl, endl ! proc beginning and ending
 ! landunit indices
integer, intent(out) :: begg, endg ! proc beginning and ending
 ! gridcell indices
```

## DESCRIPTION:

Retrieve gridcell, landunit, column, and pft bounds for process.

## REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

**A.49.12 get\_proc\_total**

## INTERFACE:

```
subroutine get_proc_total(pid, ncells, nlunits, ncols, npfts)
```

## DESCRIPTION:

Count up gridcells, landunits, columns, and pfts on process.

## ARGUMENTS:

```
implicit none
integer, intent(in) :: pid ! proc id
integer, intent(out) :: ncells ! total number of gridcells
 ! on the processor
integer, intent(out) :: nlunits ! total number of landunits
 ! on the processor
integer, intent(out) :: ncols ! total number of columns
 ! on the processor
integer, intent(out) :: npfts ! total number of pfts
 ! on the processor
```

## REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

**A.49.13 get\_proc\_global**

INTERFACE:

```
subroutine get_proc_global(numg, numl, numc, nump)
```

DESCRIPTION:

Return number of gridcells, landunits, columns, and pfts across all processes.

ARGUMENTS:

```
implicit none
integer, intent(out) :: numg ! total number of gridcells
 ! across all processors
integer, intent(out) :: numl ! total number of landunits
 ! across all processors
integer, intent(out) :: numc ! total number of columns
 ! across all processors
integer, intent(out) :: nump ! total number of pfts
 ! across all processors
```

REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

---

**A.49.14 get\_proc\_clumps**

INTERFACE:

```
integer function get_proc_clumps()
```

DESCRIPTION:

Return the number of clumps.

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

---

**A.49.15 get\_gcell\_xyind**

INTERFACE:

```
subroutine get_gcell_xyind(lbg, ubg, ixy, jxy)
```

## DESCRIPTION:

Retrieve x,y indices of a gridcell.

## ARGUMENTS:

```
implicit none
integer, intent(in) :: lbg
integer, intent(in) :: ubg
integer, pointer :: ixy(:)
integer, pointer :: jxy(:)
```

## REVISION HISTORY:

2003.09.12 Mariana Vertenstein Creation.

---

**A.49.16 map\_dc2sn\_sl\_real**

## INTERFACE:

```
subroutine map_dc2sn_sl_real(arraydc, arrays, type1d)
```

## DESCRIPTION:

Maps a decomposition single level real array into a south- $\hat{z}$ north single level real array

## USES:

```
use clmtype, only : nameg, namel, namec, namep
```

## ARGUMENTS:

```
implicit none
real(r8), pointer :: arraydc(:)
real(r8), pointer :: arrays(:)
character(len=*), intent(in) :: type1d
```

## REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

**A.49.17 map\_dc2sn\_sl\_int**

## INTERFACE:

```
subroutine map_dc2sn_sl_int(arraydc, arrays, type1d)
```

## DESCRIPTION:

Maps a decomposition single level integer array into a south- $\hat{z}$ north single level real array

## USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
integer, pointer :: arraydc(:)
integer, pointer :: arraysnc(:)
character(len=*), intent(in) :: type1d
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.18 map\_dc2sn\_ml1\_real

#### INTERFACE:

```
subroutine map_dc2sn_ml1_real(arraydc, arraysnc, type1d, lb1, ub1, revord)
```

#### DESCRIPTION:

Maps a decomposition (dc) multilevel real array into a south-*l*north (sn) multilevel level real array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
real(r8), pointer :: arraydc(:, :)
real(r8), pointer :: arraysnc(:, :)
character(len=*), intent(in) :: type1d
integer, intent(in) :: lb1
integer, intent(in) :: ub1
logical, intent(in), optional :: revord
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.19 map\_dc2sn\_ml1\_int

#### INTERFACE:

```
subroutine map_dc2sn_ml1_int(arraydc, arraysnc, type1d, lb1, ub1, revord)
```

#### DESCRIPTION:

Maps a decomposition (dc) multilevel integer array into a south-*l*north (sn) multilevel level integer array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
integer, pointer :: arraydc(:, :)
integer, pointer :: arraysnc(:, :)
character(len=*), intent(in) :: type1d
integer, intent(in) :: lb1
integer, intent(in) :: ub1
logical, intent(in), optional :: revord
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.20 `map_sn2dc_sl_real`

#### INTERFACE:

```
subroutine map_sn2dc_sl_real(arraysn, arraydc, type1d)
```

#### DESCRIPTION:

Maps a south- $\hat{z}$ north single level real array into a decomposition single level real array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
real(r8), pointer :: arraysnc(:)
real(r8), pointer :: arraydc(:)
character(len=*), intent(in) :: type1d
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.21 `map_sn2dc_sl_int`

#### INTERFACE:

```
subroutine map_sn2dc_sl_int(arraysn, arraydc, type1d)
```

#### DESCRIPTION:

Maps a south- $\hat{z}$ north single level integer array into a decomposition single level integer array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
integer, pointer :: arraysn(:)
integer, pointer :: arraydc(:)
character(len=*), intent(in) :: type1d
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.22 map\_sn2dc\_ml1\_real

#### INTERFACE:

```
subroutine map_sn2dc_ml1_real(arraysn, arraydc, type1d, lb1, ub1)
```

#### DESCRIPTION:

Maps a south-¿north multi level real array into a decomposition multi level real array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```

#### ARGUMENTS:

```
implicit none
real(r8), pointer :: arraysn(:, :)
real(r8), pointer :: arraydc(:, :)
character(len=*), intent(in) :: type1d
integer, intent(in) :: lb1
integer, intent(in) :: ub1
```

#### REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.23 map\_sn2dc\_ml1\_int

#### INTERFACE:

```
subroutine map_sn2dc_ml1_int(arraysn, arraydc, type1d, lb1, ub1)
```

#### DESCRIPTION:

Maps a south-¿north multi level integer array into a decomposition multi level integer array

#### USES:

```
use clmtype, only : nameg, namel, namec, namep
```



## ARGUMENTS:

```

 implicit none
 integer, pointer :: arrays(:,:)
 integer, pointer :: arraydc(:,:)
 character(len=*), intent(in) :: typeid
 integer, intent(in) :: lb1
 integer, intent(in) :: ub1

```

## REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

**A.49.24** `get_sn_land1d`

## INTERFACE:

```

 subroutine get_sn_land1d(snindex, typeid)

```

## DESCRIPTION:

Creates south- $\hat{z}$ north indices at a given clm level

## USES:

```

 use clmtype

```

## ARGUMENTS:

```

 implicit none
 integer, pointer :: snindex(:)
 character(len=*), intent(in) :: typeid

```

## REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

**A.49.25** `get_sn_cols1d`

## INTERFACE:

```

 subroutine get_sn_cols1d(snindex, typeid)

```

## DESCRIPTION:

Creates south- $\hat{z}$ north indices at a given clm level

## USES:

```

 use clmtype

```

## ARGUMENTS:

```

 implicit none
 integer, pointer :: snindex(:)
 character(len=*), intent(in) :: typeid

```

## REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.

---

### A.49.26 get\_sn\_pfts1d

INTERFACE:

```
subroutine get_sn_pfts1d(snindex, type1d)
```

DESCRIPTION:

Creates south- and north indices at a given clm level

USES:

```
use clmtype
```

ARGUMENTS:

```
implicit none
integer, pointer :: snindex(:)
character(len=*), intent(in) :: type1d
```

REVISION HISTORY:

2003.12.01 Mariana Vertenstein Creation.



**A.50 do\_close\_dispose (Source File: do\_close\_dispose.F90)****INTERFACE:**

```
subroutine do_close_dispose (ntapes, hist_ntimes, hist_mfilt, &
 if_stop, if_disphist, if_remvhist)
```

**DESCRIPTION:**

Determine logic for closeing and/or disposing history file  
 Sets values for if\_disphist, if\_stop, if\_remvhist (arguments)  
 Remove history files unless this is end of run or  
 history file is not full.

**USES:**

```
use shr_kind_mod, only : r8 => shr_kind_r8
#if (defined COUP_CSM)
 use clm_csmMod , only : csmstop_next, csmrstrt
#else
 use time_manager, only : is_last_step
#endif
```

**ARGUMENTS:**

```
implicit none
integer, intent(in) :: ntapes !actual number of history tapes
integer, intent(in) :: hist_ntimes(ntapes) !current numbers of time sample
integer, intent(in) :: hist_mfilt(ntapes) !maximum number of time samples
logical, intent(out) :: if_stop !true => last time step of run
logical, intent(out) :: if_disphist(ntapes) !true => save and dispose histo
logical, intent(out) :: if_remvhist(ntapes) !true => remove local history f
```

**REVISION HISTORY:**

Created by Mariana Vertenstein



## A.51 do\_restore (Source File: do\_restore.F90)

INTERFACE:

```
logical function do_restore()
```

DESCRIPTION:

```
Determine if restart dataset is to be written at this time step
```

USES:

```
#if (defined COUP_CSM)
 use clm_csmMod , only : csmstop_next, csmrstrt
#else
 use time_manager, only : is_last_step
 use histFileMod , only : if_writrest
#endif
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```



## A.52 driver (Source File: driver.F90)

INTERFACE:

```
subroutine driver (doalb, eccen, obliqr, lambm0, mvelpp)
```

DESCRIPTION:

This subroutine provides the main CLM driver calling sequence. Most computations occurs over “clumps” of gridcells (and associated subgrid scale entities) assigned to each MPI process. Computation is further parallelized by looping over clumps on each process using shared memory OpenMP or Cray Streaming Directives.

The main CLM driver calling sequence is as follows:

```
* Communicate with flux coupler [COUP_CSM]
+ interpMonthlyVeg interpolate monthly vegetation data [!DGVM]
+ readMonthlyVegetation read vegetation data for two months [!DGVM]
==== Begin Loop 1 over clumps ====
-> DriverInit save of variables from previous time step
-> Hydrology1 canopy interception and precip on ground
 -> FracWet fraction of wet vegetated surface and dry elai
-> SurfaceRadiation surface solar radiation
-> Biogeophysics1 leaf temperature and surface fluxes
-> BareGroundFluxes surface fluxes for bare soil or snow-covered
 vegetation patches
 -> MoninObukIni first-guess Monin-Obukhov length and wind speed
 -> FrictionVelocity friction velocity and potential temperature and
 humidity profiles
-> CanopyFluxes leaf temperature and surface fluxes for vegetated
 patches
 -> QSat saturated vapor pressure, specific humidity, &
 derivatives at leaf surface
 -> MoninObukIni first-guess Monin-Obukhov length and wind speed
 -> FrictionVelocity friction velocity and potential temperature and
 humidity profiles
 -> Stomata stomatal resistance and photosynthesis for
 sunlit leaves
 -> Stomata stomatal resistance and photosynthesis for
 shaded leaves
 -> QSat recalculation of saturated vapor pressure,
 specific humidity, & derivatives at leaf surface
-> Biogeophysics_Lake lake temperature and surface fluxes
+ VOCEmission compute VOC emission [VOC]
+ DGVMRespiration CO2 respriation and plant production [DGVM]
+ DGVMEcosystemDyn DGVM ecosystem dynamics: vegetation phenology [!DGV
-> EcosystemDyn "static" ecosystem dynamics: vegetation phenology
 and soil carbon [!DGVM]
-> SurfaceAlbedo albedos for next time step
-> Biogeophysics2 soil/snow & ground temp and update surface fluxes
-> pft2col Average from PFT level to column level
==== End Loop 1 over clumps ====
* Average fluxes over time interval and send to flux coupler [COUP_CSM]
==== Begin Loop 2 over clumps ====
-> Hydrology2 surface and soil hydrology
-> Hydrology_Lake lake hydrology
```



```

-> SnowAge update snow age for surface albedo calculation
-> BalanceCheck check for errors in energy and water balances
==== End Loop 2 over clumps ====
-> write_diagnostic output diagnostic if appropriate
+ Rtmriverflux calls RTM river routing model [RTM]
-> updateAccFlds update accumulated fields
-> update_hbuf accumulate history fields for time interval
Begin DGVM calculations at end of model year [DGVM]
==== Begin Loop over clumps ====
+ lpj LPJ ecosystem dynamics: reproduction, turnover,
 kill, allocation, light, mortality, fire
+ lpjreset1 reset variables & initialize for next year
==== End Loop over clumps ====
End DGVM calculations at end of model year [DGVM]
-> htapes_wrapup write history tapes if appropriate
Begin DGVM calculations at end of model year [DGVM]
==== Begin Loop over clumps ====
+ lpjreset2 reset variables and patch weights
==== End Loop over clumps ====
End DGVM calculations at end of model year [DGVM]
-> restart write restart file if appropriate
-> inicfile write initial file if appropriate

```

Optional subroutines are denoted by an plus (+) with the associated CPP variable in brackets at the end of the line. Coupler communication when coupled with CCSM components is denoted by an asterisk (\*).

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
#if (defined COUP_CSM)
use clm_varctl , only : wrtdia, fsurdat, csm_doflxave
#else
use clm_varctl , only : wrtdia, fsurdat, nsrest
#endif
use spmdMod , only : masterproc
use decompMod , only : get_proc_clumps, get_clump_bounds
use filterMod , only : filter
use clm_varcon , only : zlnd
#if (defined COUP_CAM)
use time_manager , only : get_step_size, get_curr_calday, &
 get_curr_date, get_ref_date, get_nstep, &
 is_perpetual
#else
use time_manager , only : get_step_size, get_curr_calday, &
 get_curr_date, get_ref_date, get_nstep
#endif
use histFileMod , only : update_hbuf, htapes_wrapup
use restFileMod , only : restart
#if (defined COUP_CAM)
use inicFileMod , only : inicfile, do_inicwrite, inicperp
#else
use inicFileMod , only : inicfile, do_inicwrite
#endif

```

```

 use DriverInitMod , only : DriverInit
 use BalanceCheckMod , only : BalanceCheck
 use SurfaceRadiationMod , only : SurfaceRadiation
 use Hydrology1Mod , only : Hydrology1
 use Hydrology2Mod , only : Hydrology2
 use HydrologyLakeMod , only : HydrologyLake
 use Biogeophysics1Mod , only : Biogeophysics1
 use BareGroundFluxesMod , only : BareGroundFluxes
 use CanopyFluxesMod , only : CanopyFluxes
 use Biogeophysics2Mod , only : Biogeophysics2
 use BiogeophysicsLakeMod , only : BiogeophysicsLake
 use SurfaceAlbedoMod , only : SurfaceAlbedo, Snowage
 use pft2colMod , only : pft2col
 use accFldsMod , only : updateAccFlds
#if (defined DGVM)
 use DGVM_EcosystemDynMod , only : DGVM_EcosystemDyn, DGVM_Respiration
 use DGVMMod , only : lpj, lpjreset1, lpjreset2, &
 gatherWeightsDGVM, histDGVM
#else
 use STATICEcosysDynMod , only : EcosystemDyn, interpMonthlyVeg
#endif
#if (defined VOC)
 use VOCEmissionMod , only : VOCEmission
#endif
#if (defined RTM)
 use RtmMod , only : Rtmriverflux
#endif
#if (defined COUP_CSM)
 use clm_csmMod , only : csm_dosndrcv, csm_rcv, csm_send, &
 csm_flxave, dorecv, dosend, csmstop_now
#endif
 use lnd2atmMod , only : lnd2atm
 use abortutils , only : endrun

```

## ARGUMENTS:

```

implicit none
logical , intent(in) :: doalb !true if time for surface albedo
 !calculation
real(r8), intent(in) :: eccen !Earth's orbital eccentricity
real(r8), intent(in) :: obliqr !Earth's obliquity in radians
real(r8), intent(in) :: lambm0 !Mean longitude of perihelion at the
 !vernal equinox (radians)
real(r8), intent(in) :: mvelpp !Earth's moving vernal equinox longitude
 !of perihelion + pi (radians)

```

## CALLED FROM:

```

program program_off (if COUP_OFFLINE cpp variable is defined)
program program_csm (if COUP_CSM cpp variable is defined)
subroutine atm_lnddrv in module atm_lndMod (if COUP_CAM cpp variable
 is defined)

```

## REVISION HISTORY:

2002.10.01 Mariana Vertenstein latest update to new data structures

---

**A.52.1 write\_diagnostic**

## INTERFACE:

```
subroutine write_diagnostic (wrtDia, nstep)
```

## DESCRIPTION:

Write diagnostic surface temperature output each timestep. Written to be fast but not bit-for-bit because order of summations can change each timestep.

## USES:

```
use clmtype
use decompMod , only : get_proc_bounds, get_proc_global
#if (defined SPMD)
use spmdMod , only : masterproc, npes, MPI_REAL8, MPI_ANY_SOURCE, &
 MPI_STATUS_SIZE, mpicom
#else
use spmdMod , only : masterproc
#endif
use shr_sys_mod, only : shr_sys_flush
use abortutils , only : endrun
```

## ARGUMENTS:

```
implicit none
logical, intent(in) :: wrtDia !true => write diagnostic
integer, intent(in) :: nstep !model time step
```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.53 Module fileutils (Source File: fileutils.F90)

Module containing file I/O utilities

USES:

```
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
logical, public :: lsmiou(99) !I/O file unit numbers (1 to 99)
```

PUBLIC MEMBER FUNCTIONS:

```
public :: get_filename !Returns filename given full pathname
public :: set_filename !Set remote full path filename
public :: opnfil !Open local unformatted or formatted file
public :: getfil !Obtain local copy of file
public :: putfil !Dispose file to Mass Store
public :: relavu !Close and release Fortran unit no longer in use
public :: getavu !Get next available Fortran unit number
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.53.1 get\_filename

INTERFACE:

```
character(len=256) function get_filename (fulpath)
```

DESCRIPTION:

Returns filename given full pathname

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fulpath !full pathname
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.53.2 set\_filename**

## INTERFACE:

```
character(len=256) function set_filename (rem_dir, loc_fn)
```

## DESCRIPTION:

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: rem_dir !remote directory
character(len=*), intent(in) :: loc_fn !local full path filename
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.53.3 getfil**

## INTERFACE:

```
subroutine getfil (fulpath, locfn, iflag)
```

## DESCRIPTION:

```
Obtain local copy of file
First check current working directory
Next check full pathname[fulpath] on disk
Finally check full pathname[fulpath] on mass store
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fulpath !MSS or permanent disk full pat
character(len=*), intent(out) :: locfn !output local file name
integer, optional, intent(in) :: iflag !0=>abort if file not found 1=>
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.53.4 putfil**

## INTERFACE:

```
subroutine putfil(locfn, mssfpn, pass, irt, lremov)
```

## DESCRIPTION:

Dispose to Mass Store only if nonzero retention period.  
 Put mswrite command in background for asynchronous behavior.  
 The string put into 'cmd' below needs to be changed to  
 the appropriate archival command for the users system  
 if a shell command 'mswrite' does not exist.

## ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: locfn ! Local filename
character(len=*), intent(in) :: mssfpn ! Mass Store full pathname
character(len=*), intent(in) :: pass ! write password
integer, intent(in) :: irt ! Mass Store retention time
logical, intent(in) :: lremov ! true=>remove local file

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.53.5 opnfil**

## INTERFACE:

```

subroutine opnfil (locfn, iun, form)

```

## DESCRIPTION:

Open file locfn in unformatted or formatted form on unit iun

## ARGUMENTS:

```

implicit none
character(len=*), intent(in):: locfn !file name
integer, intent(in):: iun !fortran unit number
character(len=1), intent(in):: form !file format: u = unformatted,
 !f = formatted

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.53.6 getavu**

## INTERFACE:

```

integer function getavu()

```

## DESCRIPTION:

Get next available Fortran unit number.  
 If COUP\_CSM or OFFLINE is defined, get next available Fortran unit number itst. Set lsmiou(itst). If COUP\_CAM is defined, use CAM function navu to get available unit number, in which case lsmiou is not needed.

## USES:

```
#if (defined COUP_CAM)
 use units !CAM units module
#endif
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Gordon Bonan  
 Modified for clm2 by Mariana Vertenstein

---

## A.53.7 relavu

## INTERFACE:

```
subroutine relavu (iunit)
```

## DESCRIPTION:

Close and release Fortran unit no longer in use!  
 If COUP\_CSM or OFFLINE is defined, close and release Fortran unit number iunit and set lsmiou(iunit) to false.  
 If COUP\_CAM is defined, use CAM function relunit to close/release unit number.

## USES:

```
#if (defined COUP_CAM)
 use units !CAM units module
#endif
```

## ARGUMENTS:

```
implicit none
integer, intent(in) :: iunit !Fortran unit number
```

## REVISION HISTORY:

Created by Gordon Bonan

---

### A.53.8 shell\_cmd

#### INTERFACE:

```
subroutine shell_cmd(text, ier)
```

#### DESCRIPTION:

Invoke shell command

#### ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: text
integer , intent(out):: ier
```

#### REVISION HISTORY:

Created by CAM core group  
Modified for clm2 by Mariana Vertenstein





## A.54 Module filterMod

### (Source File: filterMod.F90)

Module of filters used for processing columns and pfts of particular types, including lake, non-lake, soil, snow, non-snow, and naturally-vegetated patches.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save

type clumpfilter
#ifdef DGVM
 integer, pointer :: natvegp(:) ! DGVM naturally-vegetated (present)
 ! filter (pfts)
 integer :: num_natvegp ! number of pfts in naturally-vegetated
 ! filter
#endif
 integer, pointer :: lakep(:) ! lake filter (pfts)
 integer :: num_lakep ! number of pfts in lake filter
 integer, pointer :: nolakep(:) ! non-lake filter (pfts)
 integer :: num_nolakep ! number of pfts in non-lake filter
 integer, pointer :: lakec(:) ! lake filter (columns)
 integer :: num_lakec ! number of columns in lake filter
 integer, pointer :: nolakec(:) ! non-lake filter (columns)
 integer :: num_nolakec ! number of columns in non-lake filter
 integer, pointer :: soilc(:) ! soil filter (columns)
 integer :: num_soilc ! number of columns in soil filter
 integer, pointer :: snowc(:) ! snow filter (columns)
 integer :: num_snowc ! number of columns in snow filter
 integer, pointer :: nosnowc(:) ! non-snow filter (columns)
 integer :: num_nosnowc ! number of columns in non-snow filter
end type clumpfilter
type(clumpfilter), allocatable, public :: filter(:)
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
2004.04.27 DGVM naturally-vegetated filter added by Forrest Hoffman
```

#### A.54.1 initFilters

INTERFACE:

```
subroutine initFilters()
```

DESCRIPTION:

Initialize CLM filters.

USES:

```
use clmtype
use decompMod , only : get_nclumps, get_proc_clumps, get_clump_bounds
use clm_varcon, only : istsoil
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Mariana Vertenstein  
2004.04.27 DGVM naturally-vegetated filter added by Forrest Hoffman

## A.55 `getdatetime` (Source File: `getdatetime.F90`)

### INTERFACE:

```
subroutine getdatetime (cdate, ctime)
```

### DESCRIPTION:

A generic Date and Time routine

### ARGUMENTS:

```
implicit none
character(len=8), intent(out) :: cdate !current date
character(len=8), intent(out) :: ctime !current time
```

### REVISION HISTORY:

Created by Mariana Vertenstein



## A.56 Module histFileMod

### (Source File: histFileMod.F90)

Module containing methods to for CLM history file handling.

USES:

```

use shr_kind_mod, only : r8 => shr_kind_r8
use shr_sys_mod , only : shr_sys_getenv, shr_sys_flush
use abortutils , only : endrun
use clm_varcon , only : spval
implicit none
save
private
include "netcdf.inc"

```

PUBLIC TYPES:

```

Constants
integer , public, parameter :: max_tapes = 6 ! max number of histo
integer , public, parameter :: max_flds = 1000 ! max number of histo
integer , public, parameter :: max_namlen = 32 ! maximum number of c
Counters
integer , public :: ntapes = 0 ! index of max history file request
Namelist
integer :: ni ! implicit index below
logical, public :: &
 hist_empty_htapes = .false. ! namelist: flag indicates no default
integer, public :: &
 hist_ndens(max_tapes) = 2 ! namelist: output density of netcdf
integer, public :: &
 hist_mfilt(max_tapes) = 30 ! namelist: number of time samples p
logical, public :: &
 hist_dov2xy(max_tapes) = (/ .true., (.true., ni=2, max_tapes) /) ! namelist
integer, public :: &
 hist_nhtfrq(max_tapes) = (/ 0, (-24, ni=2, max_tapes) /) ! namelis
character(len=1), public :: &
 hist_avgflag_pertape(max_tapes) = (/ (' ', ni=1, max_tapes) /) ! namelis
character(len=max_namlen), public :: &
 hist_type1d_pertape(max_tapes) = (/ (' ', ni=1, max_tapes) /) ! namelis

character(len=max_namlen+2), public :: &
 hist_fincl1(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 hist_fincl2(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 hist_fincl3(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 hist_fincl4(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 hist_fincl5(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 hist_fincl6(max_flds) = (/ (' ', ni=1, max_flds) /) ! namelist: list of fi
character(len=max_namlen+2), public :: &
 fincl(max_flds, max_tapes) ! namelist-equivalence list of fields to

```

```

character(len=max_namlen), public :: &
 hist_fexcl1(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 hist_fexcl2(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 hist_fexcl3(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 hist_fexcl4(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 hist_fexcl5(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 hist_fexcl6(max_flds) = (/(' ',ni=1,max_flds)/) ! namelist: list of fi
character(len=max_namlen), public :: &
 fexcl(max_flds,max_tapes) ! namelist-equivalence list of fields to
Equivalence used to satisfy namelist on a wide variety of platforms
NOTE: It is *ASSUMED* that max_tapes is 6
equivalence (hist_fincl1,fincl(1,1))
equivalence (hist_fincl2,fincl(1,2))
equivalence (hist_fincl3,fincl(1,3))
equivalence (hist_fincl4,fincl(1,4))
equivalence (hist_fincl5,fincl(1,5))
equivalence (hist_fincl6,fincl(1,6))

equivalence (hist_fexcl1,fexcl(1,1))
equivalence (hist_fexcl2,fexcl(1,2))
equivalence (hist_fexcl3,fexcl(1,3))
equivalence (hist_fexcl4,fexcl(1,4))
equivalence (hist_fexcl5,fexcl(1,5))
equivalence (hist_fexcl6,fexcl(1,6))
Restart
logical, public :: if_writrest ! true=> write restart file now

```

#### PUBLIC MEMBER FUNCTIONS:

```

public :: add_fld1d ! Add a 1d single-level field to the mast
public :: add_fld2d ! Add a 2d multi-level field to the maste
public :: add_subscript ! Add a 2d subscript dimension
public :: masterlist_make_active ! Add a field to the given history file d
public :: masterlist_printflds ! Print summary of master field list
public :: htapes_build ! Initialize history file handler for ini
public :: update_hbuf ! Updates history buffer for all fields a
public :: htapes_wrapup ! Write and/or dispose history tape(s)
public :: restart_history ! Read/write history file restart data

```

#### REVISION HISTORY:

Created by Mariana Vertenstein

### A.56.1 masterlist\_printflds

#### INTERFACE:

```
subroutine masterlist_printflds()
```

## DESCRIPTION:

Print summary of master field list.

## USES:

```
use spmdMod, only : masterproc
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Mariana Vertenstein 03/2003

---

## A.56.2 masterlist\_addfld

## INTERFACE:

```
subroutine masterlist_addfld (fname, type1d, type2d, num2d, &
 typexy, nlonxy, nlatxy, &
 units, avgflag, long_name, hpindex, &
 p2c_scale_type, c2l_scale_type, l2g_scale_type)
```

## DESCRIPTION:

Add a field to the master field list. Put input arguments of field name, units, number of levels, averaging flag, and long name into a type entry in the global master field list (masterlist).

## USES:

```
use clmtype , only : nameg, namel, namec, namep, lndrof, ocnrof
use decompMod, only : get_proc_bounds, get_proc_global
#if (defined RTM)
use RunoffMod, only : get_proc_rof_bounds, get_proc_rof_global
#endif
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fname ! field name
character(len=*), intent(in) :: type1d ! 1d output type
character(len=*), intent(in) :: type2d ! 2d output type
character(len=*), intent(in) :: typexy ! xy grid output type
integer , intent(in) :: nlonxy ! xy grid longitude dimension
integer , intent(in) :: nlatxy ! xy grid latitude dimension
integer , intent(in) :: num2d ! size of second dimension (
character(len=*), intent(in) :: units ! units of field
character(len=1), intent(in) :: avgflag ! time averaging flag
character(len=*), intent(in) :: long_name ! long name of field
integer , intent(in) :: hpindex ! clmtype index for history
character(len=*), intent(in) :: p2c_scale_type ! scale type for subgrid a
character(len=*), intent(in) :: c2l_scale_type ! scale type for subgrid a
character(len=*), intent(in) :: l2g_scale_type ! scale type for subgrid a
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---



### A.56.3 htapes\_build

#### INTERFACE:

```
subroutine htapes_build ()
```

#### DESCRIPTION:

Initialize history file for initial or continuation run. For example, on an initial run, this routine initializes “ntapes” history files. On a restart run, this routine only initializes history files declared beyond what existed on the previous run. Files which already existed on the previous run have already been initialized (i.e. named and opened) in routine restart\_history. Loop over tapes and fields per tape setting appropriate variables and calling appropriate routines

#### USES:

```
use spmdMod, only : masterproc
use time_manager, only: get_curr_date, get_curr_time
```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.56.4 masterlist\_make\_active

#### INTERFACE:

```
subroutine masterlist_make_active (name, tape_index, avgflag)
```

#### DESCRIPTION:

Add a field to the default “on” list for a given history file. Also change the default time averaging flag if requested.

#### ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: name ! field name
integer, intent(in) :: tape_index ! history tape index
character(len=1), intent(in), optional :: avgflag ! time averaging flag
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.5 masterlist\_change\_timeavg**

INTERFACE:

```
subroutine masterlist_change_timeavg (t)
```

DESCRIPTION:

Override default history tape contents for a specific tape. Copy the flag into the master field list.

USES:

ARGUMENTS:

```
implicit none
integer, intent(in) :: t ! history tape index
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.6 htapes\_fieldlist**

INTERFACE:

```
subroutine htapes_fieldlist()
```

DESCRIPTION:

Define the contents of each history file based on namelist input for initial or branch run, and restart data if a restart run. Use arrays fincl and fexcl to modify default history tape contents. Then sort the result alphanumerically.

USES:

```
use spmdMod, only : masterproc
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.7 htape\_addfld**

INTERFACE:

```
subroutine htape_addfld (t, f, avgflag)
```

## DESCRIPTION:

Add a field to the active list for a history tape. Copy the data from the master field list to the active list for the tape.

## USES:

```
use decompMod , only : get_proc_bounds, get_proc_global
use clmtype , only : nameg, namel, namec, namep
```

## ARGUMENTS:

```
implicit none
integer, intent(in) :: t ! history tape index
integer, intent(in) :: f ! field index from master field
character(len=1), intent(in) :: avgflag ! time averaging flag
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.8 update\_hbuf**

## INTERFACE:

```
subroutine update_hbuf()
```

## DESCRIPTION:

Accumulate (or take min, max, etc. as appropriate) input field into its history buffer for appropriate tapes.

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.9 update\_hbuf\_field**

## INTERFACE:

```
subroutine update_hbuf_field (t,f)
```

## DESCRIPTION:

Accumulate (or take min, max, etc. as appropriate) input field into its history buffer for appropriate tapes.

## USES:

```
use spmdMod , only : masterproc
```

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index
integer, intent(in) :: f ! field index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.10 hfields\_normalize**

## INTERFACE:

```

subroutine hfields_normalize (t)

```

## DESCRIPTION:

Normalize fields on a history file by the number of accumulations. Loop over fields on the tape. Need averaging flag and number of accumulations to perform normalization.

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.11 hfields\_zero**

## INTERFACE:

```

subroutine hfields_zero (t)

```

## DESCRIPTION:

Zero out accumulation and history buffers for a given history tape. Loop through fields on the tape.

## USES:

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.12 htape\_create**

## INTERFACE:

```
subroutine htape_create (t)
```

## DESCRIPTION:

Define contents of history file t. Issue the required netcdf wrapper calls to define the history file contents.

## USES:

```

use clmtype
use ncio
use decompMod , only : get_proc_global, map_dc2sn, get_sn_landid, get_s
use clm_varpar , only : lsmlon, lsmlat, nlevsoi
use clm_varctl , only : caseid, ctitle, frivinp_rtm, fsurdat, finidat, f
use clm_varsur , only : fullgrid, offline_rdgrid, longxy, latixy, area,
 landfrac, landmask, numlon, lsmedge
use clm_varcon , only : zsoi, zlak
use fileutils , only : get_filename
use time_manager , only : get_ref_date
#if (defined RTM)
use RtmMod , only : latixy_r, longxy_r
use RunoffMod , only : get_proc_rof_global
#endif

```

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.13 htape\_timeconst**

## INTERFACE:

```
subroutine htape_timeconst(t, mode)
```

## DESCRIPTION:

Write time constant values to primary history tape. Issue the required netcdf wrapper calls to define the history file contents.

## USES:

```

use clmtype
use subgridAveMod, only : c2g
use decompMod , only : get_proc_bounds, get_proc_global
use clm_varpar , only : lsmlon, lsmlat, nlevsoi
use ncio

```

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index
character(len=*), intent(in) :: mode ! 'define' or 'write'

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.14 hfields\_write**

## INTERFACE:

```

subroutine hfields_write(t, mode)

```

## DESCRIPTION:

Write history tape. If SPMD, first gather the data to the master processor. Issue the netcdf call to write the variable.

## USES:

```

use ncdfio
use subgridAveMod
use clmtype , only : nameg, namel, namec, namep, lndrof, ocnrof
use decompMod, only : get_proc_bounds, get_proc_global
#if (defined RTM)
use RunoffMod, only : get_proc_rof_bounds, get_proc_rof_global
#endif

```

## ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index
character(len=*), intent(in) :: mode ! 'define' or 'write'

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.56.15 hfields\_1dinfo**

## INTERFACE:

```

subroutine hfields_1dinfo(t, mode)

```

## DESCRIPTION:

Write/define 1d info for history tape.

## USES:

```

use ncldio
use clmtype , only : nameg, namel, namec, namep
use decompMod, only : get_proc_bounds, get_proc_global, &
 get_sn_land1d, get_sn_cols1d, get_sn_pfts1d, map_dc

```

#### ARGUMENTS:

```

implicit none
integer, intent(in) :: t ! tape index
character(len=*), intent(in) :: mode ! 'define' or 'write'

```

#### REVISION HISTORY:

Created by Mariana Vertenstein

### A.56.16 htapes\_wrapup

#### INTERFACE:

```
subroutine htapes_wrapup ()
```

#### DESCRIPTION:

Write and/or dispose history tape(s) Determine if next time step is beginning of history interval and if so: increment the current time sample counter, open a new history file and if needed (i.e., when  $ntim = 1$ ), write history data to current history file, reset field accumulation counters to zero. If primary history file is full or at the last time step of the simulation, write restart dataset and close and dispose all history fiels. If history file is full or at the last time step of the simulation: close history file and dispose to mass store (only if file is open) and reset time sample counter to zero if file is full. Daily-averaged data for the first day in September are written on date = 00/09/02 with mscur = 0. Daily-averaged data for the first day in month mm are written on date = yyyy/mm/02 with mscur = 0. Daily-averaged data for the 30th day (last day in September) are written on date = 0000/10/01 mscur = 0. Daily-averaged data for the last day in month mm are written on date = yyyy/mm+1/01 with mscur = 0.

#### USES:

```

use clm_varctl , only : archive_dir, mss_wpass, mss_irt
use fileutils , only : set_filename, putfil
use spmdMod , only : masterproc
use time_manager , only : get_nstep, get_curr_date, get_curr_time, get_pr
use shr_const_mod, only : SHR_CONST_CDAY
use ncldio
use shr_sys_mod , only : shr_sys_flush

```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

### A.56.17 restart\_history

#### INTERFACE:

```
subroutine restart_history (nio, flag)
```

#### DESCRIPTION:

Read/write history file restart data. If the current history file(s) are not full, file(s) are opened so that subsequent time samples are added until the file is full. A new history file is used on a branch run.

#### USES:

```

 use iobinary
 use ncdio
 use clmtype , only : nameg, namel, namec, namep, ocnrof, lndrof
 use decompMod , only : get_proc_bounds, get_proc_global
 #if (defined RTM)
 use RunoffMod , only : get_proc_rof_bounds, get_proc_rof_global
 #endif
 use clm_varctl, only : archive_dir, nsrest, mss_irt
 use fileutils , only : set_filename, getfil
 #if (defined SPMD)
 use spmdMod , only : masterproc, mpicom, MPI_REAL8, MPI_INTEGER, MPI_CH
 #else
 use spmdMod , only : masterproc
 #endif

```

#### ARGUMENTS:

```

 implicit none
 integer, intent(in) :: nio !restart unit
 character(len=*), intent(in) :: flag !'read' or 'write'

```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.56.18 getname

#### INTERFACE:

```
character(len=max_namlen) function getname (iname)
```

#### DESCRIPTION:

Retrieve name portion of inname. If an averaging flag separator character is present (:) in inname, lop it off.

#### ARGUMENTS:

```

 implicit none
 character(len=*), intent(in) :: inname

```

#### REVISION HISTORY:

Created by Jim Rosinski

---



**A.56.19 getflag**

## INTERFACE:

```
character(len=1) function getflag (iname)
```

## DESCRIPTION:

Retrieve flag portion of inname. If an averaging flag separator character is present (:) in inname, return the character after it as the flag

## ARGUMENTS:

```
implicit none
character(len=*) inname ! character string
```

## REVISION HISTORY:

Created by Jim Rosinski

---

**A.56.20 list\_index**

## INTERFACE:

```
subroutine list_index (list, name, index)
```

## DESCRIPTION:

## USES:

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: list(max_flds) ! input list of names, p
character(len=max_namlen), intent(in) :: name ! name to be searched fo
integer, intent(out) :: index ! index of "name" in "li
```

## REVISION HISTORY:

Created by Jim Rosinski

---

**A.56.21 set\_hist\_filename**

## INTERFACE:

```
character(len=256) function set_hist_filename (hist_freq, hist_file)
```

## DESCRIPTION:

Determine history dataset filenames.

## USES:

```

 use clm_varctl, only : caseid
 use time_manager, only : get_curr_date, get_prev_date

```

## ARGUMENTS:

```

 implicit none
 integer, intent(in) :: hist_freq !history file frequency
 integer, intent(in) :: hist_file !history file index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.56.22 add fld1d

## INTERFACE:

```

 subroutine add_fld1d (fname, units, avgflag, long_name, &
 ptr_gcell, ptr_lunit, ptr_col, ptr_pft, ptr_roflnd, &
 ptr_rofocn, p2c_scale_type, c2l_scale_type, &
 l2g_scale_type, set_lake, default, typexy)

```

## DESCRIPTION:

Initialize a single level history field. The pointer, ptrhist, is a pointer to the clmtype array that the history buffer will use. The value of typeld passed to masterlist.add fld determines which of the 1d type of the output and the beginning and ending indices the history buffer field). Default history contents for given field on all tapes are set by calling [masterlist.make\_active] for the appropriate tape. After the masterlist is built, routine [htapes.build] is called for an initial or branch run to initialize the actual history tapes.

## USES:

```

 use clmtype
 use decompMod , only : get_proc_bounds
 use clm_varpar, only : lsmlon, lsmlat, rtmlon, rtmlat

```

## ARGUMENTS:

```

 implicit none
 character(len=*), intent(in) :: fname ! field name
 character(len=*), intent(in) :: units ! units of field
 character(len=1), intent(in) :: avgflag ! time averaging
 character(len=*), intent(in) :: long_name ! long name of f
 real(r8) , optional, pointer :: ptr_gcell(:) ! pointer to gri
 real(r8) , optional, pointer :: ptr_lunit(:) ! pointer to lan
 real(r8) , optional, pointer :: ptr_col(:) ! pointer to col
 real(r8) , optional, pointer :: ptr_pft(:) ! pointer to pft
 real(r8) , optional, pointer :: ptr_roflnd(:) ! pointer to lan
 real(r8) , optional, pointer :: ptr_rofocn(:) ! pointer to oce
 real(r8) , optional, intent(in) :: set_lake ! value to set l
 character(len=*), optional, intent(in) :: p2c_scale_type ! scale type for
 character(len=*), optional, intent(in) :: c2l_scale_type ! scale type for
 character(len=*), optional, intent(in) :: l2g_scale_type ! scale type for
 character(len=*), optional, intent(in) :: default ! if set to 'ina
 character(len=*), optional, intent(in) :: typexy ! by default set

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

A.56.23 `add_fld2d`

## INTERFACE:

```
subroutine add_fld2d (fname, type2d, units, avgflag, long_name, &
 ptr_gcell, ptr_lunit, ptr_col, ptr_pft, &
 p2c_scale_type, c2l_scale_type, l2g_scale_type, &
 set_lake, default, typexy)
```

## DESCRIPTION:

Initialize a single level history field. The pointer, `ptrhist`, is a pointer to the `clmtype` array that the history buffer will use. The value of `type1d` passed to `masterlist_add_fld` determines which of the 1d type of the output and the beginning and ending indices the history buffer field). Default history contents for given field on all tapes are set by calling `[masterlist_make_active]` for the appropriate tape. After the masterlist is built, routine `[htapes.build]` is called for an initial or branch run to initialize the actual history tapes.

## USES:

```
use clmtype
use decompMod , only : get_proc_bounds
use clm_varpar, only : lsmlon, lsmlat, nlevsoi, nlevlak, numrad
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fname ! field name
character(len=*), intent(in) :: type2d ! 2d output type
character(len=*), intent(in) :: units ! units of field
character(len=1), intent(in) :: avgflag ! time averaging
character(len=*), intent(in) :: long_name ! long name of f
real(r8) , optional, pointer :: ptr_gcell(:, :) ! pointer to gri
real(r8) , optional, pointer :: ptr_lunit(:, :) ! pointer to lan
real(r8) , optional, pointer :: ptr_col(:, :) ! pointer to col
real(r8) , optional, pointer :: ptr_pft(:, :) ! pointer to pft
real(r8) , optional, intent(in) :: set_lake ! value to set l
character(len=*), optional, intent(in) :: p2c_scale_type ! scale type for
character(len=*), optional, intent(in) :: c2l_scale_type ! scale type for
character(len=*), optional, intent(in) :: l2g_scale_type ! scale type for
character(len=*), optional, intent(in) :: default ! if set to 'ina
character(len=*), optional, intent(in) :: typexy ! by default set
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.56.24 pointer\_index

INTERFACE:

```
integer function pointer_index ()
```

DESCRIPTION:

Set the current pointer index and increment the value of the index.

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```

---

### A.56.25 add\_subscript

INTERFACE:

```
subroutine add_subscript(subname, subdim)
```

DESCRIPTION:

Add a history variable to the output history tape.

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: subname ! name of subscript
integer , intent(in) :: subdim ! dimension of subscript
```

REVISION HISTORY:

```
Created by Mariana Vertenstein
```



## A.57 Module histFldsMod

### (Source File: histFldsMod.F90)

Module containing initialization of clm history fields and files This is the module that the user must modify in order to add new history fields or modify defaults associated with existing history fields.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
public initHistFlds ! Build master field list of all possible history
 ! file fields
```

REVISION HISTORY:

Created by Mariana Vertenstein 03/2003

---

#### A.57.1 initHistFlds

INTERFACE:

```
subroutine initHistFlds()
```

DESCRIPTION:

Build master field list of all possible fields in a history file. Each field has associated with it a “long\_name” netcdf attribute that describes what the field is, and a “units” attribute. A subroutine is called to add each field to the masterlist.

USES:

```
use clmtype
use clm_varcon , only : spval
use clm_varctl , only : nsrest
#if (defined RTM)
use RunoffMod , only : runoff
#endif
use histFileMod, only : add_subscript, add_fld1d, add_fld2d, &
 masterlist_printflds, htapes_build
```

ARGUMENTS:

```
implicit none
```

REVISION HISTORY:

Mariana Vertenstein: Created 03/2003

Mariana Vertenstein: Updated interface to create history fields 10/2003



## A.58 iniTimeConst (Source File: iniTimeConst.F90)

### INTERFACE:

```
subroutine iniTimeConst
```

### DESCRIPTION:

Initialize time invariant clm variables

- 1) removed references to shallow lake - since it is not used
- 2) \*\*\*Make c%z, c%zi and c%dz allocatable depending on if you have lake or soil
- 3) rootfr only initialized for soil points

### USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use nanMod
use clmtype
use decompMod , only : get_proc_bounds, get_proc_global
use clm_varpar, only : nlevsoi, nlevlak, lsmlon, lsmlat, numpft
use clm_varsur, only : soic2d, sand3d, clay3d
use clm_varcon, only : istance, istdlak, istwet, isturb, &
 zlak, dzlak, zsoi, dzsoi, zisoi, spval
use clm_varctl, only : nsrest
use pftvarcon , only : ncorn, nwheat, noveg, ntree, roota_par, rootb_par,
 z0mr, displar, dleaf, rho1, rhos, taul, taus, xl, &
 qe25, vcmx25, mp, c3psn, &
 pftpar , tree , summergreen, raingreen , sla
 lm_sapl, sm_sapl, hm_sapl , rm_sapl , latosa
 allom1 , allom2 , allom3 , reinickerp , wooddens
use time_manager, only : get_step_size
use abortutils, only : endrun
```

### ARGUMENTS:

```
implicit none
```

### CALLED FROM:

```
subroutine initialize in module initializeMod.
```

### REVISION HISTORY:

Created by Gordon Bonan.

Updated to clm2.1 data structures by Mariana Vertenstein

### LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: ivt(:) ! vegetation type index
integer , pointer :: ixy(:) ! xy lon index (column-level)
integer , pointer :: jxy(:) ! xy lat index (column_level)
integer , pointer :: pcolumn(:) ! column index of corresponding pft
integer , pointer :: clandunit(:) ! landunit index of corresponding c
integer , pointer :: ltype(:) ! landunit type index
```



```

local pointers to implicit out arguments
real(r8), pointer :: z(:, :) ! layer depth (m)
real(r8), pointer :: zi(:, :) ! interface level below a "z" level
real(r8), pointer :: dz(:, :) ! layer thickness depth (m)
real(r8), pointer :: rootfr(:, :) ! fraction of roots in each soil la
real(r8), pointer :: dewmx(:) ! maximum allowed dew [mm]
real(r8), pointer :: bsw(:, :) ! Clapp and Hornberger "b" (nlevsoi
real(r8), pointer :: watsat(:, :) ! volumetric soil water at saturati
real(r8), pointer :: hksat(:, :) ! hydraulic conductivity at saturat
real(r8), pointer :: sucsat(:, :) ! minimum soil suction (mm) (nlevso
real(r8), pointer :: csol(:, :) ! heat capacity, soil solids (J/m**
real(r8), pointer :: tkmg(:, :) ! thermal conductivity, soil minera
real(r8), pointer :: tkdry(:, :) ! thermal conductivity, dry soil (W
real(r8), pointer :: tksatu(:, :) ! thermal conductivity, saturated s
real(r8), pointer :: wtfact(:) ! Fraction of model area with high
real(r8), pointer :: smpmin(:) ! restriction for min of soil poten
integer , pointer :: isoicol(:) ! soil color class
real(r8), pointer :: gwc_thr(:) ! threshold soil moisture based on
real(r8), pointer :: mss_frc_cly_vld(:) ! [frc] Mass fraction clay limited

```

## A.59 iniTimeVar (Source File: iniTimeVar.F90)

### INTERFACE:

```
subroutine iniTimeVar(readini, eccen, obliqr, lambm0 , mvelpp)
```

### DESCRIPTION:

Initializes the following time varying variables:

water : h2osno, h2ocan, h2osoi\_liq, h2osoi\_ice, h2osoi\_vol

snow : snowdp, snowage, snl, dz, z, zi

temperature: t\_soisno, t\_veg, t\_grnd

The variable, h2osoi\_vol, is needed by the soil albedo routine - this is not on restart since it is computed before the soil albedo computation is called.

The remaining variables are initialized by calls to ecosystem dynamics and albedo subroutines.

### USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
use clmtype
use spmdMod , only : masterproc
use decompMod , only : get_proc_clumps, get_clump_bounds
use filterMod , only : filter
use clm_varpar , only : nlevsoi, nlevsno, nlevlak
use clm_varcon , only : denice, denh2o, zlnd
use time_manager , only : get_curr_calday
use inicFileMod , only : inicfile
use FracWetMod , only : FracWet
use SurfaceAlbedoMod , only : SurfaceAlbedo
#ifdef DGVM
 use DGVMMod , only : resetTimeConstDGVM
 use DGVMEcosystemDynMod , only : DGVMEcosystemDyn
#else
 use STATICEcosysDynMod , only : EcosystemDyn, interpMonthlyVeg
#endif
```

### ARGUMENTS:

```
implicit none
logical , intent(in) :: readini !true if read in initial data set
real(r8), intent(in) :: eccen !Earth's orbital eccentricity
real(r8), intent(in) :: obliqr !Earth's obliquity in radians
real(r8), intent(in) :: lambm0 !Mean longitude of perihelion at the vernal
real(r8), intent(in) :: mvelpp !Earth's moving vernal equinox long. of pe
```

### CALLED FROM:

```
subroutine initialize in module initializeMod
```

### REVISION HISTORY:

Created by Mariana Vertenstein

### LOCAL VARIABLES:

```

local pointers to implicit in arguments
integer , pointer :: plandunit(:) ! landunit index associated with eac
logical , pointer :: lakpoi(:) ! true => landunit is a lake point
real(r8), pointer :: dz(:, :) ! layer thickness depth (m)
real(r8), pointer :: h2osoi_ice(:, :) ! ice lens (kg/m2)
real(r8), pointer :: h2osoi_liq(:, :) ! liquid water (kg/m2)
integer , pointer :: frac_veg_nosno_alb(:) ! fraction of vegetation not cov
local pointers to implicit out arguments
real(r8), pointer :: h2osoi_vol(:, :) ! volumetric soil water (0<=h2osoi_v
real(r8), pointer :: snowdp(:) ! snow height (m)
real(r8), pointer :: frac_sno(:) ! fraction of ground covered by snow
integer , pointer :: frac_veg_nosno(:) ! fraction of vegetation not covered
real(r8), pointer :: fwet(:) ! fraction of canopy that is wet (0
local pointers to implicit out arguments (lake points only)
real(r8), pointer :: fdry(:) ! fraction of foliage that is green and dr
real(r8), pointer :: tlai(:) ! one-sided leaf area index, no burying by
real(r8), pointer :: tsai(:) ! one-sided stem area index, no burying by
real(r8), pointer :: htop(:) ! canopy top (m)
real(r8), pointer :: hbot(:) ! canopy bottom (m)
real(r8), pointer :: elai(:) ! one-sided leaf area index with burying b
real(r8), pointer :: esai(:) ! one-sided stem area index with burying b

```

---

### A.59.1 mkarbinit

INTERFACE:

```
subroutine mkarbinit()
```

DESCRIPTION:

Initializes the following time varying variables:  
 water : h2osno, h2ocan, h2osoi\_liq, h2osoi\_ice, h2osoi\_vol  
 snow : snowdp, snowage, snl, dz, z, zi  
 temperature: t\_soisno, t\_veg, t\_grnd  
 The variable, h2osoi\_vol, is needed by clm\_soilalb -this is not needed on  
 restart since it is computed before the soil albedo computation is called.  
 The remaining variables are initialized by calls to ecosystem dynamics  
 and albedo subroutines.

USES:

```

use shr_kind_mod , only : r8 => shr_kind_r8
use clmtype
use decompMod , only : get_proc_bounds
use clm_varpar , only : nlevsoi, nlevsno, nlevlak
use clm_varcon , only : bdsno, istic, istwet, istsoil, denice, denh2o, s
use shr_const_mod, only : SHR_CONST_TKFRZ
#ifdef DGVM
use DGVMMod , only : resetWeightsDGVM, gatherWeightsDGVM
#endif

```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine iniTimeVar
```

REVISION HISTORY:

Created by Mariana Vertenstein

LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: pcolumn(:) ! column index associated with each
integer , pointer :: clandunit(:) ! landunit index associated with eac
integer , pointer :: ltype(:) ! landunit type
logical , pointer :: lakpoi(:) ! true => landunit is a lake point
real(r8), pointer :: dz(:, :) ! layer thickness depth (m)
real(r8), pointer :: watsat(:, :) ! volumetric soil water at saturatio
real(r8), pointer :: h2osoi_ice(:, :) ! ice lens (kg/m2)
real(r8), pointer :: h2osoi_liq(:, :) ! liquid water (kg/m2)
local pointers to implicit out arguments
integer , pointer :: snl(:) ! number of snow layers
real(r8), pointer :: t_soisno(:, :) ! soil temperature (Kelvin) (-nlevs
real(r8), pointer :: t_lake(:, :) ! lake temperature (Kelvin) (1:nlev
real(r8), pointer :: t_grnd(:) ! ground temperature (Kelvin)
real(r8), pointer :: t_veg(:) ! vegetation temperature (Kelvin)
real(r8), pointer :: h2osoi_vol(:, :) ! volumetric soil water (0<=h2osoi_v
real(r8), pointer :: h2ocan_col(:) ! canopy water (mm H2O) (column-leve
real(r8), pointer :: h2ocan_pft(:) ! canopy water (mm H2O) (pft-level)
real(r8), pointer :: h2osno(:) ! snow water (mm H2O)
real(r8), pointer :: snowdp(:) ! snow height (m)
real(r8), pointer :: snowage(:) ! non dimensional snow age [-] (new)
real(r8), pointer :: eflx_lwrad_out(:) ! emitted infrared (longwave) radiat
```



## A.60 Module inicFileMod

### (Source File: inicFileMod.F90)

Read and writes CLM initial data netCDF files

USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
use clm_varpar , only : nlevsno, nlevsoi, nlevlak, rtmlon, rtmlat
use spmdMod , only : masterproc, iam
use shr_sys_mod , only : shr_sys_flush
use ncdio
use abortutils , only : endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: do_inicwrite ! true=> write initial data file
public :: inicfile ! creat initial dataset
public :: inicfields ! read/write initial dataset fields
#ifdef COUP_CAM
public :: inicperp ! perpetual read
#endif
```

REVISION HISTORY:

Created by Mariana Vertenstein 1/04

---

#### A.60.1 inicfile

INTERFACE:

```
subroutine inicfile(flag)
```

DESCRIPTION:

Write instantaneous initial data to netCDF initial data file

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use clm_varctl , only : finidat, caseid, ctitle, version, fsurdat, archi
use time_manager, only : get_nstep, get_curr_date
use fileutils , only : set_filename, putfil, getfil
use ncdio , only : check_ret, check_dim
use decompMod , only : get_proc_global
#ifdef RTM
use RunoffMod , only : get_proc_rof_global
#endif
use shr_sys_mod , only : shr_sys_getenv
```

## ARGUMENTS:

```
implicit none
character(len=*) :: flag ! flag to specify read or write
```

## CALLED FROM:

```
subroutine driver
```

## REVISION HISTORY:

---

## A.60.2 inicfields

## INTERFACE:

```
subroutine inicfields(flag, ncid)
```

## DESCRIPTION:

Read/Write initial data from/to netCDF instantaneous initial data file

## USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use time_manager, only : get_curr_date
use decompMod , only : get_proc_bounds, get_proc_global, map_dc2sn, &
 get_sn_land1d, get_sn_cols1d, get_sn_pfts1d
use clm_varcon , only : sb, denice, denh2o
#ifdef DGVM
use DGVMMod , only : resetWeightsDGVM, gatherWeightsDGVM
#endif
#ifdef RTM
use RunoffMod , only : get_proc_rof_global
use RtmMod , only : volr, Rtmfluxini
#endif
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: flag ! flag to determine if define, writ
integer, intent(in) :: ncid ! netCDF dataset id
```

## REVISION HISTORY:

---

### A.60.3 inicperp

#### INTERFACE:

```
subroutine inicperp
```

#### DESCRIPTION:

Read perpetual initial data fields

#### USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use clm_varctl , only : finidat
use clm_varcon , only : zlnd, denice, denh2o
use decompMod , only : get_proc_bounds, get_proc_global
use fileutils , only : getfil
```

#### ARGUMENTS:

```
implicit none
```

#### REVISION HISTORY:

### A.60.4 do\_inicwrite

#### INTERFACE:

```
logical function do_inicwrite()
```

#### DESCRIPTION:

Determine if initial dataset is to be written at this time step

#### USES:

```
use time_manager, only : get_curr_date, get_prev_date, get_step_size
use clm_varctl , only : hist_crtinic
```

#### ARGUMENTS:

```
implicit none
```

#### CALLED FROM:

```
subroutine driver
```

#### REVISION HISTORY:

Created by Mariana Vertenstein



**A.60.5 set\_init\_filename**

## INTERFACE:

```
character(len=256) function set_init_filename ()
```

## DESCRIPTION:

```
Determine initial dataset filenames
```

## USES:

```
use clm_varctl , only : caseid
use time_manager, only : get_curr_date, get_step_size
```

## ARGUMENTS:

```
implicit none
```

## CALLED FROM:

```
subroutine inicwrt in this module
```

## REVISION HISTORY:

```
Created by Mariana Vertenstein
```

## A.61 Module initGridcellsMod

### (Source File: initGridCellsMod.F90)

Initializes sub-grid mapping for each land grid cell

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varpar, only : lsmlon, lsmlat, maxpatch, maxpatch_pft
use clm_varsur, only : numlon, area, latixy, longxy, landfrac

use shr_sys_mod, only : shr_sys_flush
use spmdMod , only : iam
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
private
save
```

PUBLIC MEMBER FUNCTIONS:

```
public initGridcells ! Initialize sub-grid gridcell mapping
!PRIVATE MEMBER FUNCTIONS:
private landunit_veg_compete
private landunit_veg_noncompete
private landunit_special
private landunit_crop_noncompete
private initGridcellsGlob ! Initialize global part clmtype
 ! (topological info)
```

REVISION HISTORY:

Created by Mariana Vertenstein

### A.61.1 initGridcells

INTERFACE:

```
subroutine initGridcells (vegxy, wtxy)
```

DESCRIPTION:

Initialize sub-grid mapping and allocates space for derived type hierarchy. For each land gridcell determine landunit, column and pft properties. Note that ngcells, nlunits, ncols and npfts are per-processor totals here and are currently not used for anything other than placeholders. Determine if there are any vegetated landunits and if so---the weight of the vegetated landunit relative to the gridcell. The first landunit contains all the vegetated patches (if any) For now, the vegetated patches will all be gathered on a single landunit, with each vegetated type having its own column on that landunit. The special patches (urban, lake, wetland, glacier) each get their own landunit having a single column and one non-vegetated pfts

## USES:

```

use decompMod , only : get_proc_bounds, get_proc_global, &
 get_gcell_info, get_gcell_xyind, &
 get_sn_index
use shr_const_mod, only : SHR_CONST_PI

```

## ARGUMENTS:

```

implicit none
integer , intent(in) :: vegxy(lsmlon,lsmlat,maxpatch) ! PFT type
real(r8), intent(in) :: wtxy(lsmlon,lsmlat,maxpatch) ! subgrid patch
 ! weights

```

## REVISION HISTORY:

Created by Peter Thornton and Mariana Vertenstein

---

**A.61.2 landunit\_veg\_compete**

## INTERFACE:

```

subroutine landunit_veg_compete (nveg, wtveg, wtxy, vegxy, i, j, &
 gi, li, ci, pi)

```

## DESCRIPTION:

Initialize vegetated landunit with competition

## USES:

```

use clm_varcon, only : istsoil

```

## ARGUMENTS:

```

implicit none
integer , intent(in) :: nveg ! number of vegetated patches in gridcell
real(r8), intent(in) :: wtveg ! weight relative to gridcell of veg
 ! landunit
real(r8), intent(in) :: wtxy(lsmlon,lsmlat,maxpatch) ! subgrid patch
 ! weights
integer , intent(in) :: vegxy(lsmlon,lsmlat,maxpatch) ! PFT type
integer , intent(in) :: i ! 2d longitude index
integer , intent(in) :: j ! 2d latitude index
integer , intent(in) :: gi ! gridcell index
integer , intent(inout) :: li ! landunit index
integer , intent(inout) :: ci ! column index
integer , intent(inout) :: pi ! pft index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.61.3 landunit\_veg\_noncompete**

## INTERFACE:

```
subroutine landunit_veg_noncompete (nveg, wtveg, wtxy, vegxy, i, j, &
 gi, li, ci, pi)
```

## DESCRIPTION:

Initialize vegetated landunit without competition

## USES:

```
use clm_varcon, only : istsoil
```

## ARGUMENTS:

```
implicit none
integer , intent(in) :: nveg ! number of vegetated patches in gridc
real(r8), intent(in) :: wtveg ! weight relative to gridcell of veg l
real(r8), intent(in) :: wtxy(lsmlon,lsmlat,maxpatch) ! subgrid patch wei
integer , intent(in) :: vegxy(lsmlon,lsmlat,maxpatch) ! PFT type
integer , intent(in) :: i ! 2d longitude index
integer , intent(in) :: j ! 2d latitude index
integer , intent(in) :: gi ! gridcell index
integer , intent(inout) :: li ! landunit index
integer , intent(inout) :: ci ! column index
integer , intent(inout) :: pi ! pft index
```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.61.4 landunit\_special**

## INTERFACE:

```
subroutine landunit_special (wtxy, i, j, m, gi, li, ci, pi)
```

## DESCRIPTION:

Initialize special landunits (urban, lake, wetland, glacier)

## USES:

```
use pftvarcon, only : noveg
use clm_varcon, only : istice, istwet, istdlak, isturb
use clm_varpar, only : npatch_lake, npatch_wet, npatch_urban, &
 npatch_glacier
```

## ARGUMENTS:

```

implicit none
real(r8), intent(in) :: wtxy(lsmlon,lsmlat,maxpatch) !subgrid patch weig
integer, intent(in) :: i !2-dim longitude index
integer, intent(in) :: j !2-dim latitude index
integer, intent(in) :: m !2-dim PFT patch index
integer, intent(in) :: gi !gridcell index
integer, intent(inout) :: li !landunit index
integer, intent(inout) :: ci !column index
integer, intent(inout) :: pi !pft index

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.61.5 landunit\_crop\_noncompete

## INTERFACE:

```

subroutine landunit_crop_noncompete (ncrop, wtcrop, wtxy, vegxy, i, j, &
 gi, li, ci, pi)

```

## DESCRIPTION:

Initialize crop landunit without competition

## USES:

```

use clm_varcon, only : istsoil
use clm_varpar, only : npatch_crop

```

## ARGUMENTS:

```

implicit none
integer , intent(in) :: ncrop ! number of vegetated patches in grid
real(r8), intent(in) :: wtcrop ! weight relative to gridcell of veg
real(r8), intent(in) :: wtxy(lsmlon,lsmlat,maxpatch) ! subgrid patch wei
integer , intent(in) :: vegxy(lsmlon,lsmlat,maxpatch) ! PFT type
integer , intent(in) :: i ! 2d longitude index
integer , intent(in) :: j ! 2d latitude index
integer , intent(in) :: gi ! gridcell index
integer , intent(inout) :: li ! landunit index
integer , intent(inout) :: ci ! column index
integer , intent(inout) :: pi ! pft index

```

## REVISION HISTORY:

Created by Sam Levis

---

### A.61.6 initGridcellsGlob

## INTERFACE:

```
subroutine initGridcellsGlob()
#if (defined SPMD)
```

## DESCRIPTION:

Set up 1d array of weights and indices for xy mapping.  
Note: if DGVM is defined, weights are updated in DGVM mode

## USES:

```
use decompMod , only : get_proc_bounds, get_proc_global
use spmdMod , only : masterproc, MPI_INTEGER, mpicom
use spmdGathScatMod, only : gather_data_to_master
```

## ARGUMENTS:

```
implicit none
```

## REVISION HISTORY:

Created by Mariana Vertenstein



## A.62 Module initializeMod (Source File: initializeMod.F90)

Performs land model initialization

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: initialize
```

REVISION HISTORY:

Created by Gordon Bonan, Sam Levis and Mariana Vertenstein

### A.62.1 initialize

INTERFACE:

```
#if (defined OFFLINE) || (defined COUP_CSM)
 subroutine initialize(eccen , obliqr , lambm0 , mvelpp)
#elif (defined COUP_CAM)
 subroutine initialize(eccen , obliqr , lambm0 , mvelpp , &
 cam_caseid , cam_ctitle , cam_nsrest, cam_nstep, &
 cam_irad , cam_crtinic , cam_nhtfrq, cam_mfilt, &
 cam_longxy , cam_latixy , cam_numlon, &
 cam_landmask, cam_landfrac, cam_irt)
#endif
```

DESCRIPTION:

Land model initialization.

- o Initializes run control variables via the [clmexp] namelist.
- o Reads surface data on model grid.
- o Defines the multiple plant types and fraction areas for each surface type
- o Builds the appropriate subgrid <-> grid mapping indices and weights.
- o Set up parallel processing.
- o Initializes time constant variables.
- o Reads restart data for a restart or branch run.
- o Reads initial data and initializes the time variant variables for an init
- o Initializes history file output.
- o Initializes river routing model.
- o Initializes accumulation variables.

USES:

```
use shr_kind_mod , only : r8 => shr_kind_r8
use spmdMod , only : masterproc
use clmtypeInitMod , only : initClmtype
use initGridCellsMod, only : initGridCells
```



```

 use clm_varpar , only : lsmlon, lsmlat, maxpatch
 use clm_varsur , only : varsur_alloc, varsur_dealloc
 use clm_varctl , only : fsurdat, finidat, nsrest, irad, &
 mksrf_offline_fgrid, mksrf_offline_fnavyoro

 use controlMod , only : control_init, control_print
 use filterMod , only : initFilters
 use decompMod , only : initDecomp, get_proc_clumps, get_clump_bound
 use histFldsMod , only : initHistFlds
 use restFileMod , only : restart
 use accFldsMod , only : initAccFlds, initAccClmtype
 use mksrfdatMod , only : mksrfdat
 use surfFileMod , only : surfrd
 use pftvarcon , only : pftconrd

#if (defined DGVM)
 use DGVMEcosystemDynMod, only : DGVMEcosystemDynini
#else
 use STATICEcosysDynMod , only : EcosystemDynini
#endif
#if (defined DGVM)
 use DGVMMod , only : resetTimeConstDGVM
#endif
#if (defined RTM)
 use RtmMod , only : Rtmgridini, Rtmlandini
#endif
#if (defined OFFLINE)
 use atmdrvMod , only : atm_getgrid
#endif
#if (defined COUP_CSM)
 use clm_csmMod , only : csm_recvgrid, csm_initialize, csm_sendalb
#endif
#if (defined COUP_CAM)
 use lp_coupling , only : lp_coupling_init
#endif
#if (defined COUP_CAM)
 use time_manager , only : get_curr_date, get_nstep
#else
 use time_manager , only : get_curr_date, get_nstep, advance_timestep,
#endif
#endif

```

#### ARGUMENTS:

```

implicit none
real(r8), intent(inout) :: eccen !Earth's orbital eccentricity
real(r8), intent(inout) :: obliqr !Earth's obliquity in radians
real(r8), intent(inout) :: lambm0 !Mean longitude of perihelion at the
real(r8), intent(inout) :: mvelpp !Earth's moving vernal equinox longit
#if (defined COUP_CAM)
character(len=*), intent(in) :: cam_caseid !cam caseid
character(len=*), intent(in) :: cam_ctitle !cam title
character(len=*), intent(in) :: cam_crtinic !cam initial dataset genera
integer , intent(in) :: cam_irad !cam radiation frequency
integer , intent(in) :: cam_nsrest !cam 0=initial run, > 0=con
integer , intent(in) :: cam_nstep !cam current time index
integer , intent(in) :: cam_nhtfrq !cam history write freq for
integer , intent(in) :: cam_mfilt !cam number of files per ta

```

```

integer , intent(in) :: cam_irt !cam mss retention time
integer , intent(in) :: cam_numlon(:) !cam number of longitudes
real(r8), intent(in) :: cam_longxy(:, :) !cam lon values
real(r8), intent(in) :: cam_latixy(:, :) !cam lat values
real(r8), intent(in) :: cam_landfrac(:, :) !cam fractional land
integer , intent(in) :: cam_landmask(:, :) !cam land mask
#endif

```

CALLED FROM:

```

routine program_off if cpp token OFFLINE is defined
routine program_csm if cpp token COUP_CSM is defined
routine atm_lnd_ini in module atm_lndMod if cpp token COUP_CAM is defined

```

REVISION HISTORY:

Created by Gordon Bonan, Sam Levis and Mariana Vertenstein

---

## A.62.2 header

INTERFACE:

```

subroutine header()

```

DESCRIPTION:

Echo and save model version number

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varctl , only : version
use spmdMod , only : masterproc

```

ARGUMENTS:

```

implicit none

```

CALLED FROM:

```

subroutine initialize in this module

```

REVISION HISTORY:

Created by Gordon Bonan



## A.63 Module iobinary

### (Source File: iobinary.F90)

Set of wrappers to write binary I/O

USES:

```

 use shr_kind_mod, only: r8 => shr_kind_r8
 #if (defined SPMD)
 use spmdMod , only : masterproc, mpicom, MPI_REAL8, MPI_INTEGER, &
 MPI_LOGICAL
 use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_master
 #else
 use spmdMod , only : masterproc
 #endif
 use decompMod , only : map_sn2dc, map_dc2sn
 use abortutils , only : endrun
 use clmtype , only : nameg, namel, namec, namep, lndrof, ocnrof

```

PUBLIC TYPES:

```

 implicit none
 save

```

PUBLIC MEMBER FUNCTIONS:

```

public :: readin, wrtout ! read and write bindary I/O
interface readin
 module procedure readin_1darray_int
 module procedure readin_2darray_int
 module procedure readin_1darray_real
 module procedure readin_2darray_real
end interface
interface wrtout
 module procedure wrtout_1darray_int
 module procedure wrtout_2darray_int
 module procedure wrtout_1darray_real
 module procedure wrtout_2darray_real
end interface

```

REVISION HISTORY:

Created by Mariana Vertenstein

### A.63.1 readin\_1d\_array\_int

INTERFACE:

```

 subroutine readin_1darray_int (iu, iarr, clmlevel)

```

DESCRIPTION:

Wrapper routine to read integer 1d array from restart binary file

## ARGUMENTS:

```

implicit none
integer, intent(in) :: iu !input unit
integer, pointer :: iarr(:) !input data
character(len=*), intent(in) :: clmlevel !type of input data

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.63.2 readin\_1darray\_real**

## INTERFACE:

```

subroutine readin_1darray_real (iu, rarr, clmlevel)

```

## DESCRIPTION:

Wrapper routine to read real 1d array from restart binary file

## ARGUMENTS:

```

implicit none
integer, intent(in) :: iu !input unit
real(r8), pointer :: rarr(:) !input data
character(len=*), intent(in) :: clmlevel !input data type

```

## REVISION HISTORY:

**A.63.3 readin\_2d\_arrayint**

## INTERFACE:

```

subroutine readin_2darray_int (iu, iarr, clmlevel)

```

## DESCRIPTION:

Wrapper routine to read integer 2d array from restart binary file

## ARGUMENTS:

```

implicit none
integer, intent(in) :: iu !input unit
integer, pointer :: iarr(:, :) !input data
character(len=*), intent(in) :: clmlevel !type of input data

```

## REVISION HISTORY:

**A.63.4 readin\_2darray\_real**

INTERFACE:

```
subroutine readin_2darray_real (iu, rarr, clmlevel)
```

DESCRIPTION:

Wrapper routine to read real 2d array from restart binary file

ARGUMENTS:

```
implicit none
integer, intent(in) :: iu !input unit
real(r8), pointer :: rarr(:, :) !input data
character(len=*), intent(in) :: clmlevel !type of input data
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.63.5 wrtout\_1d\_array\_int**

INTERFACE:

```
subroutine wrtout_1darray_int (iu, iarr, clmlevel)
```

DESCRIPTION:

Wrapper routine to write integer array to restart binary file

ARGUMENTS:

```
implicit none
integer, intent(in) :: iu !output unit
integer, pointer :: iarr(:) !output data
character(len=*), intent(in) :: clmlevel !output 1d type
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.63.6 wrtout\_1d\_array\_real**

INTERFACE:

```
subroutine wrtout_1darray_real (iu, rarr, clmlevel)
```

DESCRIPTION:

Wrapper routine to write real array to restart binary file

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: iu !output unit
real(r8), pointer :: rarr(:) !output data
character(len=*), intent(in) :: clmlevel !input data type
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.63.7 wrtout\_2d\_array\_int

#### INTERFACE:

```
subroutine wrtout_2darray_int (iu, iarr, clmlevel)
```

#### DESCRIPTION:

Wrapper routine to write integer array to restart binary file

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: iu !output unit
integer, pointer :: iarr(:, :) !output data
character(len=*), intent(in) :: clmlevel !output data 1d type
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

### A.63.8 wrtout\_2darray\_real

#### INTERFACE:

```
subroutine wrtout_2darray_real (iu, rarr, clmlevel)
```

#### DESCRIPTION:

Wrapper routine to write real array to restart binary file

#### ARGUMENTS:

```
implicit none
integer, intent(in) :: iu !input unit
real(r8), pointer :: rarr(:, :) !output data
character(len=*), intent(in) :: clmlevel !type of input data
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.63.9 getnum**

INTERFACE:

```
integer function getnum (type1d)
```

DESCRIPTION:

Determines size (across all processors) from 1d type

USES:

```
use decompMod, only : get_proc_global
#ifdef RTM
use RunoffMod, only : get_proc_rof_global
#endif
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: type1d ! type of 1d array
```

REVISION HISTORY:

Created by Mariana Vertenstein





## A.64 Module lnd2atmMod (Source File: lnd2atmMod.F90)

Compute l2a component of gridcell derived type

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: lnd2atm
```

REVISION HISTORY:

```
03-04-27 : Created by Mariana Vertenstein
```

---

### A.64.1 lnd2atm

INTERFACE:

```
subroutine lnd2atm(init)
```

DESCRIPTION:

Compute l2a component of gridcell derived type

USES:

```
use decompMod, only : get_proc_clumps, get_clump_bounds
```

ARGUMENTS:

```
implicit none
logical, optional, intent(in) :: init ! if true=>only set a subset of ar
```

REVISION HISTORY:

```
Mariana Vertenstein: created 03/10-25
```

---

### A.64.2 makel2a

INTERFACE:

```
subroutine makel2a(lbp, ubp, lbc, ubc, lbg, ubg, init)
```

DESCRIPTION:

Compute l2a component of gridcell derived type

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon , only : sb
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbp, ubp ! per-proc beginning and ending pft ind
integer, intent(in) :: lbc, ubc ! per-proc beginning and ending column
integer, intent(in) :: lbg, ubg ! per-proc gridcell ending gridcell ind
logical, optional, intent(in) :: init ! if true=>only set a subset of ar
```

REVISION HISTORY:

```
03-04-27 : Created by Mariana Vertenstein
03-08-25 : Updated to vector data structure (Mariana Vertenstein)
```

## A.65 Module lp\_coupling

### (Source File: lp\_coupling.F90)

Module provides coupling between the atmosphere physics (decomposed into chunks) and the land (decomposed into clumps).

USES:

```

 use shr_kind_mod, only : r8 => shr_kind_r8

 #if (defined SPMD)
 use mpishorthand, only : mpir8, mpicom
 use spmd_dyn , only : npes
 use pmgrid , only : iam
 #else
 use spmdMod , only : npes, iam
 #endif
 use decompMod , only : get_nclumps, get_clump_owner_id, &
 get_clump_ncells_id, get_clump_coord_id, &
 get_clump_gcell_info
 use phys_grid , only : get_chunk_coord_owner_p
 use abortutils , only : endrun

```

PUBLIC TYPES:

```

 implicit none

```

PUBLIC MEMBER FUNCTIONS:

```

 public lp_coupling_init ! initialize clump<-->chunk mapping
 public lp_coupling_finalize ! destroy clump<-->chunk mapping
 public alltoall_clump_to_chunk_init ! communicate fluxes from lnd to atm
 public alltoall_clump_to_chunk ! communicate fluxes from lnd to atm
 public alltoall_chunk_to_clump ! communicate fluxes from atm to lnd
 SAVE

```

REVISION HISTORY:

```

 2002.09.11 Forrest Hoffman Creation.
 2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.

```

### A.65.1 lp\_coupling\_init

INTERFACE:

```

 subroutine lp_coupling_init()

```

DESCRIPTION:

This subroutine initializes the mapping between the atmosphere physics chunks and the land clumps. It may (and must) be called repeatedly to re-initialize the mapping if the decomposition of either the atmosphere physics or the land changes. It allocates communication buffers constructs vectors of counts and displacements used for subsequent communication between MPI processes.

ARGUMENTS:

```
implicit none
```

LOCAL VARIABLES:

```
integer :: p, c, g ! loop indices
integer :: nclumps ! number of clumps defined
integer :: ncells ! number of clump cells
integer :: clump_owner ! clump owner
integer, dimension(:), allocatable :: lons ! clump longitudes
integer, dimension(:), allocatable :: lats ! clump latitudes
integer, dimension(:), allocatable :: lchnks ! chunk ids
integer, dimension(:), allocatable :: cols ! chunk columns
integer, dimension(:), allocatable :: chunk_owners ! chunk owners
integer :: max_gpc = 0 ! max cells per clump
integer :: ier ! error codes
```

REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.
```

---

### A.65.2 `lp_coupling_finalize`

INTERFACE:

```
subroutine lp_coupling_finalize()
```

ARGUMENTS:

```
implicit none
```

DESCRIPTION:

This subroutine destroys the mapping between the atmosphere physics chunks and the land clumps if the `lpc_init_flag` flag is set. It is called from `lp_coupling_init()` to ensure memory is recycled when a new mapping is to be created.

REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.
```

---

### A.65.3 `alltoall_clump_to_chunk_init`

INTERFACE:

```
subroutine alltoall_clump_to_chunk_init (srfflx2d)
```

DESCRIPTION:

This subroutine performs the initial communication from the land model to the atmosphere physics (from clumps to chunks) based on the mapping constructed in `lp_coupling_init()`.

USES:

```

use ppgrid , only : begchunk, endchunk
use comsrf , only : srfflx_parm, snowland
use clm_varcon, only : sb
use clmtype
use lnd2atmMod, only : lnd2atm

```

#### ARGUMENTS:

```

implicit none
type(srfflx_parm), intent(inout), dimension(begchunk:endchunk) :: srfflx2d

```

#### REVISION HISTORY:

```

2002.09.11 Forrest Hoffman Creation.
2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.
2003.05.01 Mariana Vertenstein Updated to l2as data structures

```

### A.65.4 alltoall\_clump\_to\_chunk

#### INTERFACE:

```

subroutine alltoall_clump_to_chunk (srfflx2d)

```

#### DESCRIPTION:

This subroutine performs the communication from the land model to the atmosphere physics (from clumps to chunks) based on the mapping constructed in `lp_coupling_init()`.

#### USES:

```

use ppgrid , only : begchunk, endchunk
use comsrf , only : srfflx_parm, snowland
use constituents, only : pcnst, pnats
use clmtype
use lnd2atmMod , only : lnd2atm

```

#### ARGUMENTS:

```

implicit none
type(srfflx_parm), intent(inout), dimension(begchunk:endchunk) :: srfflx2d

```

#### REVISION HISTORY:

```

2002.09.11 Forrest Hoffman Creation.
2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.
2003.05.01 Mariana Vertenstein Updated to l2as data structures

```

### A.65.5 alltoall\_chunk\_to\_clump

#### INTERFACE:

```

subroutine alltoall_chunk_to_clump (srf_state)

```

## DESCRIPTION:

This subroutine performs communication from the atmosphere physics to the land model (from chunks to clumps) based on the mapping constructed in `lp_coupling_init()`.

## USES:

```
use ppgrid , only: begchunk, endchunk
use comsrf , only: surface_state
use clmtype
use clm_varcon, only: rair, po2, pco2
```

## ARGUMENTS:

```
implicit none
type(surface_state), intent(in), dimension(begchunk:endchunk) :: srf_state
```

## REVISION HISTORY:

```
2002.09.11 Forrest Hoffman Creation.
2002.11.18 Mariana Vertenstein and Forrest Hoffman Updated to clm2.1.
```

**A.65.6 mkglacier**

## INTERFACE:

```
subroutine mkglacier (fgla, ndiag, gla_o)
```

## DESCRIPTION:

```
make percent glacier
```

## USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varsur !surface variables
use clm_varctl !run control variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils, only : endrun
```

## ARGUMENTS:

|                                               |                           |
|-----------------------------------------------|---------------------------|
| implicit none                                 |                           |
| character(len=*), intent(in) :: fgla          | !input glacier dataset fi |
| integer , intent(in) :: ndiag                 | !unit number for diagnost |
| real(r8), intent(out):: gla_o(lsmllon,lsmlat) | !percent glacier on outpu |

## CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

## REVISION HISTORY:

```
Author: Sam Levis
```





## A.66 Module mkgridMod (Source File: mkgridMod.F90)

Routines to create land model grid

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar
use clm_varsur
use clm_varctl
use areaMod
use fileutils , only : getfil
use spmdMod , only : masterproc
use ncdio
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
#if (defined OFFLINE)
 public :: mkgrid_offline ! Obtain land model grid when mode is offline
#else
 public :: mkgrid_cam ! Generate land model grid when mode is CAM or CS
#endif
```

!PRIVATE MEMBER FUNCTIONS:

```
#if (defined OFFLINE)
 private :: read_grid_offline ! Read land model grid when mode is offline
 private :: create_grid_offline ! Generate land model grid when mode is off
#endif
```

REVISION HISTORY:

Author: Mariana Vertenstein

### A.66.1 mkgrid\_offline

INTERFACE:

```
subroutine mkgrid_offline()
```

DESCRIPTION:

Obtain land model grid when mode is offline  
 If namelist variable mksrf\_offline\_fgrid is the empty string, then  
 the corresponding file will be used to determine the land model grid  
 If namelist variable mksrf\_offline is not the empty string, then  
 the land model grid will be generated at run time

## ARGUMENTS:

```
implicit none
```

## CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

## REVISION HISTORY:

```
Author: Mariana Vertenstein
```

---

**A.66.2 read\_grid\_offline**

## INTERFACE:

```
subroutine read_grid_offline()
```

## DESCRIPTION:

Read land model grid when mode is offline.  
 If namelist variable mksrf\_offline\_fgrid is the empty string, then the corresponding file will be used to determine the land model grid. Assume that the input data file has the land grid in the following form:

|                         |                                        |
|-------------------------|----------------------------------------|
| lon                     | => dimension                           |
| lat                     | => dimension                           |
| lon(lsmlon)             | => full grid longitudes                |
| nlon(lsmlat)            | => reduced grid number of lats per lon |
| r lon(lsmlon,lsmlat)    | => reduced grid longitudes             |
| lat(lsmlat)             | => grid latitudes                      |
| oro(lsmlon,lsmlat)      | => 2d land mask                        |
| landfrac(lsmlon,lsmlat) | => 2d land fraction                    |

## ARGUMENTS:

```
implicit none
```

## CALLED FROM:

```
subroutine mkgrid_offline in this module
```

## REVISION HISTORY:

```
Author: Mariana Vertenstein
```

---

**A.66.3 create\_grid\_offline**

## INTERFACE:

```
subroutine create_grid_offline()
```

## DESCRIPTION:

Generate land model grid when mode is offline.

Surface grid edges -- Grids do not have to be global. To allow this, grids must define the north, east, south, and west edges:

If namelist variable mksrf\_offline is not the empty string, then the land m grid will be generated at run time using the settings of the namelist variables

- o mksrf\_offline\_fnavyoro : 20 min navy orography dataset
- o mksrf\_offline\_edgen (edge(1)) : northern edge of grid (degrees): > -9
- o mksrf\_offline\_edgee (edge(2)) : eastern edge of grid (degrees) : see f
- o mksrf\_offline\_edges (edge(3)) : southern edge of grid (degrees): >= -9
- o mksrf\_offline\_edgew (edge(4)) : western edge of grid (degrees) : see f

For partial grids, northern and southern edges are any latitude between 90 (North Pole) and -90 (South Pole). Western and eastern edges are any longitude between -180 and 180, with longitudes west of Greenwich negative. That is, western edge >= -180 and < 180; eastern edge > western edge and <= 180.

For global grids, northern and southern edges are 90 (North Pole) and -90 (South Pole). The western edge of the longitude grid starts at the dateline if the grid is generated (the longitudes for each grid cell correspond with the edges (i.e., range from -180 to 180)).

## ARGUMENTS:

implicit none

## CALLED FROM:

subroutine mkgrid\_offline in this module

## REVISION HISTORY:

Author: Mariana Vertenstein

## A.66.4 mkgrid\_cam

## INTERFACE:

```
subroutine mkgrid_cam(cam_longxy, cam_latixy, cam_numlon, cam_landfrac, &
 cam_landmask)
```

## DESCRIPTION:

Generate land model grid when mode is CAM or CSM

For CAM mode get grid AND fractional land and land mask from cam model

For CSM mode get grid AND fractional land and land mask from coupler

## ARGUMENTS:

```
implicit none
real(r8), intent(in) :: cam_longxy(:, :) !cam longitudes
integer, intent(in) :: cam_numlon(:) !number of cam longitudes pe
real(r8), intent(in) :: cam_latixy(:, :) !cam latitudes
real(r8), intent(in) :: cam_landfrac(:, :) !cam land fraction
integer, intent(in) :: cam_landmask(:, :) !cam land mask
```

CALLED FROM:

```
subroutine mkfsurdat in module mksrfdatMod
```

REVISION HISTORY:

Author: Mariana Vertenstein

## A.67 Module mklai

### (Source File: mklai.F90)

Make LAI/SAI/height data

REVISION HISTORY:

Author: Sam Levis

---

#### A.67.1 mklai

INTERFACE:

```
subroutine mklais (flai, ndiag, ncido, pft_max)
```

DESCRIPTION:

Make LAI/SAI/height data  
 Portions of this code could be moved out of the month loop  
 for improved efficiency

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varctl !run control variables
use clm_varsur !surface variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils , only : endrun
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: flai ! input lai-sai-hgt dataset
integer , intent(in) :: ndiag ! unit number for diagnostic ou
integer , intent(in) :: ncido ! output netcdf file id
integer , intent(in), optional :: pft_max(lsmlon,lsmlat,maxpatch_pft) ! PFT
```

CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

REVISION HISTORY:

Author: Sam Levis



**A.68 mklanwat (Source File: mklanwat.F90)**

## INTERFACE:

```
subroutine mklanwat (flanwat, ndiag, lake_o, swmp_o)
```

## DESCRIPTION:

make %lake and %wetland from Cogley's one degree data

## USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varsur !surface variables
use clm_varctl !run control variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils , only : endrun
```

## ARGUMENTS:

|                                               |                           |
|-----------------------------------------------|---------------------------|
| implicit none                                 |                           |
| character(len=*), intent(in) :: flanwat       | !input lanwat dataset fil |
| integer , intent(in) :: ndiag                 | !unit number for diagnost |
| real(r8), intent(out):: lake_o(lsmlon,lsmlat) | !percent lake on output g |
| real(r8), intent(out):: swmp_o(lsmlon,lsmlat) | !percent wetland on outpu |

## CALLED FROM:

## REVISION HISTORY:

Author: Gordon Bonan





## A.69 Module mkpft

### (Source File: mkpft.F90)

Make PFT data

#### REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.69.1 mkpfts

#### INTERFACE:

```
subroutine mkpfts(fpft, ndiag, noveg, pctlnd_o, pctpft_o, pctpft_max, &
 pft_max)
```

#### DESCRIPTION:

Make PFT data

This dataset consists of the %cover of the [numpft]+1 PFTs used by the model. The input %cover pertains to the "vegetated" portion of the grid cell and sums to 100. The real portion of each grid cell covered by each PFT is the PFT cover times the fraction of the grid cell that is land. This is the quantity preserved when area-averaging from the input (1/2 degree) grid to the models grid.

#### USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use fileutils , only : getfil
use shr_sys_mod , only : shr_sys_flush
use clm_varpar ! parameters
use clm_varsur ! surface variables
use clm_varctl ! run control variables
use areaMod ! area averaging routines
use ncio
use abortutils, only : endrun
```

#### ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fpft ! input pft dat
integer , intent(in) :: ndiag ! unit number f
integer , intent(in) :: noveg ! PFT number fo
real(r8), intent(out):: pctlnd_o(lsmlon,lsmlat) ! output grid:
real(r8), intent(out):: pctpft_o(lsmlon,lsmlat,0:numpft) ! output grid P
real(r8), optional, intent(out) :: pctpft_max(lsmlon,lsmlat,maxpatch_pft) !
integer , optional, intent(out) :: pft_max(lsmlon,lsmlat,maxpatch_pft) !
```

#### CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

#### REVISION HISTORY:

Author: Mariana Vertenstein



## A.70 mkrank (Source File: mkrank.F90)

### INTERFACE:

```
subroutine mkrank (n, a, miss, iv, num)
```

### DESCRIPTION:

Return indices of largest [num] values in array [a]

### USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use abortutils, only : endrun
```

### ARGUMENTS:

```
implicit none
integer , intent(in) :: n !array length
real(r8), intent(in) :: a(0:n) !array to be ranked
integer , intent(in) :: miss !missing data value
integer , intent(in) :: num !number of largest values requested
integer , intent(out):: iv(num) !index to [num] largest values in array [a]
```

### CALLED FROM:

```
subroutine mkpft
subroutine mksoicol
subroutine mksoitex
```

### REVISION HISTORY:

Author: Gordon Bonan



**A.71 mksoicol (Source File: mksoicol.F90)**

## INTERFACE:

```
subroutine mksoicol (fsoicol, ndiag, pctgla_o, soil_color_o)
```

## DESCRIPTION:

Make soil color classes for model grid from BATS T42 data

## USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varsur !surface variables
use clm_varctl !run control variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils, only : endrun
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fsoicol !input soicol dataset
integer , intent(in) :: ndiag !unit number for diagn
real(r8), intent(in) :: pctgla_o(lsmlon,lsmlat) !output grid: percent
integer , intent(out):: soil_color_o(lsmlon,lsmlat) !output grid: soil col
```

## CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

## REVISION HISTORY:

Author: Gordon Bonan



## A.72 mksoitex (Source File: mksoitex.F90)

### INTERFACE:

```
subroutine mksoitex (fsoitex, ndiag, pctgla_o, sand_o, clay_o)
```

### DESCRIPTION:

make %sand and %clay from IGBP soil data, which includes  
igbp soil 'mapunits' and their corresponding textures

### USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varsur !surface variables
use clm_varctl !run control variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils, only : endrun
```

### ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: fsoitex !soil texture dataset file n
integer , intent(in) :: ndiag !unit # for diagnostic outpu
real(r8), intent(in) :: pctgla_o(lsmlon,lsmlat) !% glacier (output gri
real(r8), intent(out):: sand_o(lsmlon,lsmlat,nlevsoi) !% sand (output grid)
real(r8), intent(out):: clay_o(lsmlon,lsmlat,nlevsoi) !% clay (output grid)
```

### CALLED FROM:

```
subroutine mksrfdat in module mksrfdatMod
```

### REVISION HISTORY:

Authors: Gordon Bonan and Sam Levis





## A.73 Module mksrfdatMod

### (Source File: mksrfdatMod.F90)

Creates land model surface dataset from original "raw" data files.  
Surface dataset contains model grid, pfts, inland water, glacier,  
soil texture, soil color, LAI and SAI and urban fraction.

USES:

```
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
```

```
save
```

```
!PRIVATE MEMBER FUNCTIONS:
```

```
public :: mksrfdat ! create model surface dataset
```

REVISION HISTORY:

Author: Mariana Vertenstein

### A.73.1 mksrfdat

INTERFACE:

```
subroutine mksrfdat(cam_longxy, cam_latixy, cam_numlon, cam_landfrac, &
 cam_landmask)
```

DESCRIPTION:

Creates land model surface dataset from original "raw" data files.  
Surface dataset contains model grid, pfts, inland water, glacier,  
soil texture, soil color, LAI and SAI and urban fraction.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use shr_sys_mod , only : shr_sys_getenv
use nanMod
use clm_varpar
use clm_varctl
use clm_varsur
use pftvarcon , only : noveg
use time_manager, only : is_last_step
use fileutils , only : putfil, opnfil, getavu, relavu, get_filename
use areaMod
use spmdMod
use mkgridMod
use ncdio
use mklai
use mkpft
```

## ARGUMENTS:

```
implicit none
real(r8), optional, intent(in) :: cam_longxy(:, :) !cam lon values
real(r8), optional, intent(in) :: cam_latixy(:, :) !cam lat values
integer , optional, intent(in) :: cam_numlon(:) !cam number of longit
real(r8), optional, intent(in) :: cam_landfrac(:, :) !cam fractional land
integer , optional, intent(in) :: cam_landmask(:, :) !cam land mask
```

## CALLED FROM:

```
routine initialize
```

## REVISION HISTORY:

Authors: Gordon Bonan, Sam Levis and Mariana Vertenstein

**A.74 mkurban (Source File: mkurban.F90)**

## INTERFACE:

```
subroutine mkurban (furb, ndiag, urb_o)
```

## DESCRIPTION:

```
make %urban
```

## USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varsur !surface variables
use clm_varctl !run control variables
use fileutils, only : getfil
use areaMod !area averaging routines
use shr_sys_mod, only : shr_sys_flush
use ncdio
use abortutils , only : endrun
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: furb !input urban dataset file n
integer , intent(in) :: ndiag !unit number for diagnostic
real(r8), intent(out) :: urb_o(lsmlon,lsmlat) !percent urban on output gr
```

## CALLED FROM:

```
subroutine mksrfdat in module srfdatMod
```

## REVISION HISTORY:

```
Author: Sam Levis
```



## A.75 Module mpiinc (Source File: mpiinc.F90)

Data and parameters used for MPI. Note: The #include of "mpif.h" which is typically in f77 fixed format means that this module MUST be in fixed format. However, the advantage of this module is that modules which use this module (and theremore the mpif.h variables) must not be in f77 fixed format.

### REVISION HISTORY:

Created by Mariana Vertenstein



## A.76 Module nanMod

### (Source File: nanMod.F90)

Set parameters for the floating point flags "inf" Infinity and "nan" not-a-number. As well as "bigint" the point at which integers start to overflow. These values are used to initialize arrays with as a way to detect if arrays are being used before being set.

Note that bigint is the largest possible 32-bit integer.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
```

PUBLIC TYPES:

```
implicit none
save
#ifdef __PGI
 quiet nan for portland group compilers
 real(r8), parameter :: inf = 0'0777600000000000000000'
 real(r8), parameter :: nan = 0'0777700000000000000000'
 integer, parameter :: bigint = 0'1777777777'
#else
 signaling nan otherwise
 real(r8), parameter :: inf = 0'0777600000000000000000'
 real(r8), parameter :: nan = 0'0777610000000000000000'
 integer, parameter :: bigint = 0'1777777777'
#endif
```

REVISION HISTORY:

Created by Mariana Vertenstein based on cam module created by  
CCM core group





## A.77 Module nc dioMod (Source File: nc dio.F90)

Generic interfaces to write fields to netcdf files

USES:

```

 use shr_kind_mod , only : r8 => shr_kind_r8
#ifdef SPMD
 use spmdMod , only : masterproc, mpicom, MPI_REAL8, MPI_INTEGER, &
 MPI_LOGICAL
#else
 use spmdMod , only : masterproc
#endif
 use clmtype
 use clm_varcon , only : spval
 use shr_sys_mod , only : shr_sys_flush
 use abortutils , only : endrun
#ifdef (defined SCAM)
 use scamMod, only : initlonidx,initlatidx
#endif

```

PUBLIC TYPES:

```

 implicit none
 include 'netcdf.inc'
 save
 public :: check_ret ! checks return status of netcdf calls
 public :: check_var ! determine if variable is on netcdf file
 public :: check_dim ! validity check on dimension

```

REVISION HISTORY:

### A.77.1 check\_dim

INTERFACE:

```

 subroutine check_dim(ncid, dimname, value)

```

DESCRIPTION:

Validity check on dimension

ARGUMENTS:

```

 implicit none
 integer, intent(in) :: ncid
 character(len=*), intent(in) :: dimname
 integer, intent(in) :: value

```

REVISION HISTORY:

**A.77.2 check\_var**

INTERFACE:

```
subroutine check_var(ncid, varname, varid, readvar)
```

DESCRIPTION:

Check if variable is on netcdf file

ARGUMENTS:

```
implicit none
integer, intent(in) :: ncid
character(len=*), intent(in) :: varname
integer, intent(out) :: varid
logical, intent(out) :: readvar
```

REVISION HISTORY:

**A.77.3 check\_ret**

INTERFACE:

```
subroutine check_ret(ret, calling)
```

DESCRIPTION:

Check return status from netcdf call

ARGUMENTS:

```
implicit none
integer, intent(in) :: ret
character(len=*) :: calling
```

REVISION HISTORY:

**A.77.4 ncd\_defvar**

INTERFACE:

```
subroutine ncd_defvar(ncid, varname, xtype, &
 dim1name, dim2name, dim3name, dim4name, dim5name, &
 long_name, units, cell_method, missing_value, fill_value, &
 imissing_value, ifill_value)
```

## DESCRIPTION:

Define a netcdf variable

## ARGUMENTS:

```
implicit none
integer , intent(in) :: ncid ! input unit
character(len=*) , intent(in) :: varname ! variable name
integer , intent(in) :: xtype ! external type
character(len=*) , intent(in), optional :: dim1name ! dimension name
character(len=*) , intent(in), optional :: dim2name ! dimension name
character(len=*) , intent(in), optional :: dim3name ! dimension name
character(len=*) , intent(in), optional :: dim4name ! dimension name
character(len=*) , intent(in), optional :: dim5name ! dimension name
character(len=*) , intent(in), optional :: long_name ! attribute
character(len=*) , intent(in), optional :: units ! attribute
character(len=*) , intent(in), optional :: cell_method ! attribute
real(r8) , intent(in), optional :: missing_value ! attribute for
real(r8) , intent(in), optional :: fill_value ! attribute for
integer , intent(in), optional :: imissing_value ! attribute for
integer , intent(in), optional :: ifill_value ! attribute for
```

## REVISION HISTORY:

## A.77.5 ncd\_iolocal\_int\_1d

## INTERFACE:

```
subroutine ncd_iolocal_int_1d(varname, data, datadim1, &
 flag, ncid, nlonxy, nlatxy, nt, readvar)
```

## DESCRIPTION:

I/O for 1d int field

## USES:

```
use decompMod, only : map_dc2sn, map_sn2dc
#ifdef SPMD
use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_master
#endif
```

## ARGUMENTS:

```
implicit none
character(len=*) , intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*) , intent(in) :: varname ! variable name
integer , pointer :: data(:) ! local decomposition
character(len=*) , intent(in) :: datadim1 ! dimension name
```

```

integer , optional, intent(in) :: nlonxy ! 2d longitude size
integer , optional, intent(in) :: nlatxy ! 2d latitude size
integer , optional, intent(in) :: nt ! time sample index
logical , optional, intent(out):: readvar ! true => variable is

```

REVISION HISTORY:

---

### A.77.6 `ncd_iolocal_real_1d`

INTERFACE:

```

subroutine ncd_iolocal_real_1d(varname, data, datadim1, flag, ncid, &
 nlonxy, nlatxy, nt, readvar)

```

DESCRIPTION:

I/O for 1d int field

USES:

```

 use decompMod, only : map_dc2sn, map_sn2dc
#ifdef RTM
 use RunoffMod , only : runoff
#endif
#ifdef SPMD
 use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_master
#endif

```

ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , pointer :: data(:) ! local decomposition
character(len=*), intent(in) :: datadim1 ! dimension name
integer , optional, intent(in) :: nlonxy ! 2d longitude size
integer , optional, intent(in) :: nlatxy ! 2d latitude size
integer , optional, intent(in) :: nt ! time sample index
logical , optional, intent(out):: readvar ! true => variable is

```

REVISION HISTORY:

---

### A.77.7 `ncd_iolocal_int_2d`

INTERFACE:

```
subroutine ncd_iolocal_int_2d(varname, data, datadim1, datadim2, &
 lowerb2, upperb2, flag, ncid, nlonxy, nlatxy, nt, readvar)
```

DESCRIPTION:

Netcdf i/o of 2d initial integer field out to netCDF file

USES:

```
use decompMod, only : map_dc2sn, map_sn2dc
#ifdef SPMD
use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_master
#endif
```

ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
integer , pointer :: data(:, :) ! local decomposition
character(len=*), intent(in) :: datadim1 ! dimension 1 name
character(len=*), intent(in) :: datadim2 ! dimension 2 name
integer , optional, intent(in) :: nlonxy ! 2d longitude size
integer , optional, intent(in) :: nlatxy ! 2d latitude size
integer , optional, intent(in) :: nt ! time sample index
integer , optional, intent(in) :: lowerb2, upperb2 ! lower and upp
logical , optional, intent(out):: readvar ! true => variable is
```

REVISION HISTORY:

### A.77.8 ncd\_iolocal\_real\_2d

INTERFACE:

```
subroutine ncd_iolocal_real_2d(varname, data, datadim1, datadim2, &
 lowerb2, upperb2, flag, ncid, nlonxy, nlatxy, nt, readvar)
```

DESCRIPTION:

Netcdf i/o of 2d initial integer field out to netCDF file

USES:

```
use decompMod, only : map_dc2sn, map_sn2dc
#ifdef SPMD
use spmdGathScatMod, only : scatter_data_from_master, gather_data_to_master
#endif
```

ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , pointer :: data(:, :) ! local decomposition
character(len=*), intent(in) :: datadim1 ! dimension 1 name
character(len=*), intent(in) :: datadim2 ! dimension 2 name
integer , optional, intent(in) :: nlonxy ! 2d longitude size
integer , optional, intent(in) :: nlatxy ! 2d latitude size
integer , optional, intent(in) :: nt ! time sample index
integer , optional, intent(in) :: lowerb2, upperb2 ! lower and upper
logical , optional, intent(out):: readvar ! true => variable is

```

REVISION HISTORY:

### A.77.9 ncd\_ioglobal\_int\_var

INTERFACE:

```
subroutine ncd_ioglobal_int_var(varname, data, flag, ncid, readvar, nt)
```

DESCRIPTION:

I/O of integer variable

ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
integer , intent(in) :: data ! local decomposition
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

REVISION HISTORY:

### A.77.10 ncd\_ioglobal\_real\_var

INTERFACE:

```
subroutine ncd_ioglobal_real_var(varname, data, flag, ncid, readvar, nt)
```

DESCRIPTION:

I/O of real variable

#### ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , intent(in) :: data ! local decomposition
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

#### REVISION HISTORY:

### A.77.11 ncd\_ioglobal\_int\_1d

#### INTERFACE:

```
subroutine ncd_ioglobal_int_1d(varname, data, flag, ncid, readvar, nt)
```

#### DESCRIPTION:

Master I/O for 1d integer data

#### ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
integer , intent(inout) :: data(:) ! local decomposition
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

#### REVISION HISTORY:

### A.77.12 ncd\_ioglobal\_real\_1d

#### INTERFACE:

```
subroutine ncd_ioglobal_real_1d(varname, data, flag, ncid, readvar, nt)
```

#### DESCRIPTION:

Master I/O for 1d real data



## ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , intent(inout) :: data(:) ! local decomposition
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

## REVISION HISTORY:

---

**A.77.13** `ncd_ioglobal_int_2d`

## INTERFACE:

```
subroutine ncd_ioglobal_int_2d(varname, data, flag, ncid, readvar, nt)
```

## DESCRIPTION:

netcdf I/O of global 2d integer array

## ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
integer , intent(inout) :: data(:, :) ! local decomposition
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

## REVISION HISTORY:

---

**A.77.14** `ncd_ioglobal_real_2d`

## INTERFACE:

```
subroutine ncd_ioglobal_real_2d(varname, data, long_name, units, flag, &
 ncid, readvar, nt)
```

## DESCRIPTION:

netcdf I/O of global 2d real array

## ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , intent(inout) :: data(:,,:) ! local decomposition
character(len=*), optional, intent(in) :: long_name ! variable long name
character(len=*), optional, intent(in) :: units ! variable units
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

REVISION HISTORY:

---

### A.77.15 ncd\_ioglobal\_int\_3d

INTERFACE:

```

subroutine ncd_ioglobal_int_3d(varname, data, long_name, units, flag, &
 ncid, readvar, nt)

```

DESCRIPTION:

netcdf I/O of global 3d integer array

ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
integer , intent(inout) :: data(:,,:) ! local decomposition
character(len=*), optional, intent(in) :: long_name ! variable long name
character(len=*), optional, intent(in) :: units ! variable units
logical , optional, intent(out):: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

REVISION HISTORY:

---

### A.77.16 ncd\_ioglobal\_real\_3d

INTERFACE:

```

subroutine ncd_ioglobal_real_3d(varname, data, long_name, units, flag, &
 ncid, readvar, nt)

```

DESCRIPTION:

netcdf I/O of global 3d real array

#### ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: flag ! 'read' or 'write'
integer , intent(in) :: ncid ! input unit
character(len=*), intent(in) :: varname ! variable name
real(r8) , intent(inout) :: data(:, :, :) ! local decomposition
character(len=*), optional, intent(in) :: long_name ! variable long name
character(len=*), optional, intent(in) :: units ! variable units
logical , optional, intent(out) :: readvar ! true => variable is
integer , optional, intent(in) :: nt ! time sample index

```

#### REVISION HISTORY:

### A.77.17 `get_size_dim1`

#### INTERFACE:

```
integer function get_size_dim1 (datadim1)
```

#### DESCRIPTION:

Determine 1d size from datadim1

#### USES:

```

use decompMod, only : get_proc_global
#ifdef RTM
use RunoffMod, only : get_proc_rof_global
#endif

```

#### ARGUMENTS:

```

implicit none
character(len=*), intent(in) :: datadim1 !type of clm 1d array

```

#### REVISION HISTORY:

### A.77.18 `subroutine scam_field_offsets`

#### INTERFACE:

```
subroutine scam_field_offsets(ncid, datadim1, data_offset, ndata)
```

#### DESCRIPTION:

Read/Write initial data from/to netCDF instantaneous initial data file

## USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use decompMod , only : get_proc_bounds
use nanMod
```

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: datadim1 ! dimension 1 name
integer, intent(in) :: ncid ! netCDF dataset id
integer, intent(out) :: data_offset ! offset into land array
 ! 1st column
integer, intent(out) :: ndata ! number of column (or
 ! pft points to read)
```

## CALLED FROM:

## REVISION HISTORY:

Created by John Truesdale



## A.78 Module pft2colMod

### (Source File: pft2colMod.F90)

Contains calls to methods to perform averages over from pfts to columns for model variables.

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use subgridAveMod
use clmtype
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: pft2col ! obtain column properties from average over column pfts
```

REVISION HISTORY:

```
03/09/08: Created by Mariana Vertenstein
```

---

### A.78.1 pft2col

INTERFACE:

```
subroutine pft2col (lbc, ubc, num_nolakec, filter_nolakec)
```

DESCRIPTION:

Averages over all pfts for variables defined over both soil and lake to provide the column-level averages of state and flux variables defined at the pft level.

ARGUMENTS:

```
implicit none
integer, intent(in) :: lbc, ubc ! column bounds
integer, intent(in) :: num_nolakec ! number of column non
integer, intent(in) :: filter_nolakec(ubc-lbc+1) ! column filter for no
```

REVISION HISTORY:

```
03/09/08: Created by Mariana Vertenstein
```



## A.79 Module pftvarcon

### (Source File: pftvarcon.F90)

Module containing vegetation constants and method to  
eads and initialize vegetation (PFT) constants.

USES:

```

use shr_kind_mod, only : r8 => shr_kind_r8
use abortutils , only : endrun
use clm_varpar
#if (defined SCAM)
 use scamMod, only : lsmptfile
#endif

```

PUBLIC TYPES:

```

implicit none
save
Vegetation type constants
character(len=40) pftname(0:numpft) !PFT description

integer ncorn ! value for corn
integer nwheat ! value for wheat
integer noveg ! value for not vegetated
integer ntree ! value for last type of tree

real(r8) dleaf(0:numpft) ! characteristic leaf dimension (m)
real(r8) c3psn(0:numpft) ! photosynthetic pathway: 0. = c4, 1. = c3
real(r8) vcmx25(0:numpft) ! max rate of carboxylation at 25C (umol CO2
real(r8) mp(0:numpft) ! slope of conductance-to-photosynthesis rel
real(r8) qe25(0:numpft) ! quantum efficiency at 25C (umol CO2 / umol
real(r8) xl(0:numpft) ! leaf/stem orientation index
real(r8) rhol(0:numpft,numrad) ! leaf reflectance: 1=vis, 2=nir
real(r8) rhos(0:numpft,numrad) ! stem reflectance: 1=vis, 2=nir
real(r8) taul(0:numpft,numrad) ! leaf transmittance: 1=vis, 2=nir
real(r8) taus(0:numpft,numrad) ! stem transmittance: 1=vis, 2=nir
real(r8) z0mr(0:numpft) ! ratio of momentum roughness length to cano
real(r8) displar(0:numpft) ! ratio of displacement height to canopy top
real(r8) roota_par(0:numpft) ! CLM rooting distribution parameter [1/m]
real(r8) rootb_par(0:numpft) ! CLM rooting distribution parameter [1/m]
real(r8) crop(0:numpft) ! crop pft: 0. = not crop, 1. = crop pft

real(r8) sla(0:numpft) ! specific leaf area [m2 leaf g-1 carbo
real(r8) pftpar(0:numpft,1:npftpar) ! the rest for use with DGVM
real(r8) lm_sapl(0:numpft)
real(r8) sm_sapl(0:numpft)
real(r8) hm_sapl(0:numpft)
real(r8) rm_sapl(0:numpft)
logical tree(0:numpft)
logical summergreen(0:numpft)
logical raingreen(0:numpft)

real(r8), parameter :: reinickerp = 1.6 !parameter in allometric equation
real(r8), parameter :: wooddens = 2.0e5 !wood density (gC/m3)

```



```

real(r8), parameter :: latosa = 8.0e3 !ratio of leaf area to sapwood cros
 !area (Shinozaki et al 1964a,b)
real(r8), parameter :: allom1 = 100.0 !parameters in allometric
real(r8), parameter :: allom2 = 40.0
real(r8), parameter :: allom3 = 0.5

```

## PUBLIC MEMBER FUNCTIONS:

```

public :: pftconrd ! Read and initialize vegetation (PFT) constants

```

## REVISION HISTORY:

Created by Sam Levis (put into module form by Mariana Vertenstein)

---

**A.79.1 pftconrd**

## INTERFACE:

```

subroutine pftconrd

```

## DESCRIPTION:

Read and initialize vegetation (PFT) constants

## USES:

```

 use fileutils , only : opnfil, getfil, relavu, getavu
 use clm_varctl, only : fpftcon
#ifdef SPMD
 use spmdMod , only : masterproc, mpicom, MPI_REAL8
#else
 use spmdMod , only : masterproc
#endif

```

## ARGUMENTS:

```

implicit none

```

## CALLED FROM:

```

routine initialize in module initializeMod

```

## REVISION HISTORY:

Created by Gordon Bonan

## A.80 program\_csm (Source File: program\_csm.F90)

INTERFACE:

PROGRAM program\_csm

DESCRIPTION:

Driver for CLM as the land component of CCSM.

This program is the driver for CLM to work as the land component of CCSM. The flux coupler will provide all the appropriate atmospheric forcing for the land model to run.

- o the land surface model returns to the CCSM flux coupler surface fluxes, temperatures, and albedos for only the land points on the [lsmlon x lsmlat] grid
- o the land surface model uses its own surface type data set. because it processes only land points, this data set must correctly define what points are land and what are not land
- o the land surface model uses its own grid dimensions (lsmlon and lsmlat). currently these must equal lsmlon and lsmlat so that there is a direct correspondence between the atmosphere and land grids
- o the zenith angle calculation is calculated for the NEXT time step rather than the current time step. make sure the calendar day is for the NEXT time step. make sure the solar declination calculation is the same as in the atmospheric model but for the NEXT time step. make sure the calendar day is for greenwich time (see next comment).
- o subroutine calendr: this generates a julian day (with fraction) based on the time step, which is used to calculate the solar zenith angle. this time must be at the greenwich meridian to get the correct zenith angle. also, output from this subroutine is used to calculate the month (1, ..., 12), day (1, ..., 31), and year (00, ...) of the simulation.
- o the land surface model calculates its own net solar radiation and net longwave radiation at the surface. the net longwave radiation at the surface will differ somewhat from that calculated from the CCSM flux coupler because the cpl model will use the upward longwave flux (or radiative temperature) from the previous time step whereas the land surface model uses the flux for the current time step. the net solar radiation should equal that calculated from the flux coupler. if not, there is a problem.

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar !parameters
use clm_varctl !run control variables
use shr_orb_mod !orbital parameters and routines
use shr_msg_mod !csm message passing routines and variables
#if (defined SPMD)
 use spmdMod , only : masterproc, iam, spmd_init, mpicom
#else
 use spmdMod , only : masterproc, iam
#endif

```

```
use initializeMod, only : initialize
use clm_csmMod , only : csmstop_now, csm_setup, csm_shutdown
use time_manager , only : advance_timestep, get_nstep
```

## ARGUMENTS:

```
implicit none
#include "gpt.inc"
```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.81 program\_off (Source File: program\_off.F90)

### INTERFACE:

PROGRAM program\_off

### DESCRIPTION:

"off-line" code to mimic coupling to an atmospheric model.  
 This program is an "off-line" driver for clm2.  
 This code can be used to run the clm2 uncoupled from any atmospheric model.  
 The appropriate atmospheric forcing is provided in module [atmdrvMod.F90]

- o If running as an offline driver, the land surface model may use a different grid than the input atmospheric data. The atmospheric data is then interpolated to the land model grid inside the atmospheric driver module [atmdrvMod.F90].
- o If running as part of cam, the land surface model must use the same grid as the cam.
- o If running through the flux coupler, the land surface model grid is interpolated to the atmospheric grid inside the flux coupler
- o To map from the atmospheric grid to the land grid, the atmospheric model must provide latitudes and longitudes (degrees) for each grid point and the North, East, South, and West edges of atmospheric grid. Comparable data for the land grid are provided by the land model. When mapping from land to atm grid, an atm grid cell that is part land and part ocean (as defined by the land surface grid) will have fluxes only based on the land portion.
- o The zenith angle calculation is for the NEXT time step rather than the current time step. Make sure the calendar day is for the NEXT time step. Make sure the calendar day is for Greenwich time (see next comment).
- o The land surface model calculates its own net solar radiation and net longwave radiation at the surface. The net longwave radiation at the surface will differ somewhat from that calculated in the atmospheric model because the atm model will use the upward longwave flux (or radiative temperature) from the previous time step whereas the land surface model uses the flux for the current time step. The net solar radiation should equal that calculated in the atmospheric model. If not, there is a problem in how the models are coupled.

### USES:

```

use shr_kind_mod , only : r8 => shr_kind_r8
use clm_varpar
use clm_varctl , only : irad, nsrest
use initializeMod, only : initialize
use atmdrvMod , only : atmdrv
#if (defined SPMD)
 use spmdMod , only : masterproc, iam, mpicom, spmd_init
#else
 use spmdMod , only : masterproc, iam
#endif
use shr_orb_mod !orbital parameters and routines
use time_manager, only : is_last_step, advance_timestep, get_nstep, get_ste
```

## ARGUMENTS:

```
implicit none
#include "gpt.inc"
```

## REVISION HISTORY:

```
Author: Gordon Bonan and Mariana Vertenstein
11/30/01 Peter Thornton : Added use globals, removed use clm_varctl
```

## A.82 Module restFileMod

### (Source File: restFileMod.F90)

Reads from or writes to/ the CLM restart file.

USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use spmdMod , only : masterproc
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: restart
!PRIVATE MEMBER FUNCTIONS:
private :: restart_setup !Setup restart files, perform consis
private :: restart_time !Read/write restart time manager dat
private :: restart_biogeophys !Read/write restart biogeophysics da
private :: restart_wrapup !Close restart file and write restar
private :: write_rest_pfile !Writes restart pointer file
private :: set_restart_filename !Sets restart filename
```

REVISION HISTORY:

Author: Mariana Vertenstein

### A.82.1 restart

INTERFACE:

```
subroutine restart (flag)
```

DESCRIPTION:

Read CLM restart file.

USES:

```
use clm_varctl , only : nsrest, rpntdir, rpntfil, nrevs, &
 archive_dir, mss_irt, mss_wpass, &
 csm_doflxave, caseid
use accumulMod , only : restart_accum
#if (defined RTM)
use RtmMod , only : restart_rtm
#endif
use histFileMod, only : restart_history
#if (defined DGVM)
use DGVMRestMod, only : restart_dgvm
#endif
#if (defined COUP_CSM)
use clm_csmMod , only : restart_coupler
#endif
```

## ARGUMENTS:

```
implicit none
include 'netcdf.inc'
character(len=*), intent(in) :: flag !'read' or 'write'
```

## CALLED FROM:

```
subroutine driver
```

## REVISION HISTORY:

```
Author: Mariana Vertenstein
```

---

**A.82.2 restart\_setup**

## INTERFACE:

```
subroutine restart_setup (nio, flag)
```

## DESCRIPTION:

```
Setup restart file and perform necessary consistency checks
```

## USES:

```
use fileutils , only : opnfil, getfil, getavu, relavu
use time_manager, only : get_step_size, get_nstep
use clm_varctl , only : nsrest, nrevsn, rpntdir, rpntfil, caseid, &
 brnch_retain_casename
```

## ARGUMENTS:

```
implicit none
include 'netcdf.inc'
integer, intent(out) :: nio !restart unit
character(len=*), intent(in) :: flag !'read' or 'write'
```

## CALLED FROM:

```
subroutine restart in this module
```

## REVISION HISTORY:

```
Created by Mariana Vertenstein
```

---

**A.82.3 restart\_time**

## INTERFACE:

```
subroutine restart_time (nio, flag)
```

## DESCRIPTION:

Read/write time manager information to/from restart

USES:

```
#if (defined COUP_CSM)
 use controlMod, only : csm_dtime !dtime from input namelist
#endif
 use time_manager, only : get_step_size, timemgr_write_restart, &
 timemgr_read_restart, timemgr_restart, get_nstep
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: nio !restart unit
character(len=*), intent(in) :: flag !'read' or 'write'
```

CALLED FROM:

subroutine restart in this module

REVISION HISTORY:

Author: Mariana Vertenstein

---

#### A.82.4 restart\_biogeophys

INTERFACE:

```
subroutine restart_biogeophys (nio, flag)
```

DESCRIPTION:

Read/Write biogeophysics information to/from restart file.

USES:

```
use clmtype
use iobinary
use decompMod , only : get_proc_bounds, get_proc_global
use clm_varpar, only : nlevsoi, numrad, lsmlon, lsmlat
use clm_varcon, only : denice, denh2o
```

ARGUMENTS:

```
implicit none
integer, intent(in) :: nio !restart unit
character(len=*), intent(in) :: flag !'read' or 'write'
```

CALLED FROM:

subroutine restart in this module

REVISION HISTORY:

Author: Mariana Vertenstein

---



### A.82.5 restart\_wrapup

#### INTERFACE:

```
subroutine restart_wrapup (nio, flag)
```

#### DESCRIPTION:

Close and archive restart file and write restart pointer file if in write mode, otherwise just close restart file if in read mode

#### USES:

```
use clm_varctl, only : mss_irt, mss_wpass, archive_dir
#if (defined COUP_CSM)
use clm_csmMod, only : csmstop_next
#endif
use fileutils, only : putfil, relavu, set_filename
use time_manager, only : is_last_step
```

#### ARGUMENTS:

```
implicit none
integer, intent(inout) :: nio !restart unit
character(len=*), intent(in) :: flag !'read' or 'write'
```

#### CALLED FROM:

```
subroutine restart in this module
```

#### REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.82.6 write\_rest\_pfile

#### INTERFACE:

```
subroutine write_rest_pfile ()
```

#### DESCRIPTION:

Open restart pointer file. Write names of current restart and history files. If using mass store, these are the mass store names except if mss\_irt=0 (no mass store files written). Close.

#### USES:

```
use clm_varctl, only : rpntdir, mss_irt, archive_dir, rpntfil
use fileutils, only : set_filename, relavu
use fileutils, only : getavu, opnfil
```

#### ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine restart in this module
```

REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.82.7 set\_restart\_filename

INTERFACE:

```
character(len=256) function set_restart_filename ()
```

DESCRIPTION:

USES:

```
use clm_varctl , only : caseid
use time_manager, only : get_curr_date
```

ARGUMENTS:

```
implicit none
```

CALLED FROM:

```
subroutine restart in this module
```

REVISION HISTORY:

Author: Mariana Vertenstein



## A.83 snowdp2lev (Source File: snowdp2lev.F90)

### INTERFACE:

```
subroutine snowdp2lev(lbc, ubc)
```

### DESCRIPTION:

Create snow layers and interfaces given snow depth.  
Note that `cps%zi(0)` is set in routine `iniTimeConst`.

### USES:

```
use shr_kind_mod, only : r8 => shr_kind_r8
use clmtype
use clm_varpar , only : nlevsno
```

### ARGUMENTS:

```
implicit none
integer, intent(in) :: lbc, ubc ! column bounds
```

### REVISION HISTORY:

Created by Mariana Vertenstein

### LOCAL VARIABLES:

```
local pointers to implicit in arguments
integer , pointer :: clandunit(:) ! landunit index associated with each co
real(r8), pointer :: snowdp(:) ! snow height (m)
logical , pointer :: lakpoi(:) ! true => landunit is a lake point
local pointers to implicit out arguments
integer , pointer :: snl(:) ! number of snow layers
real(r8), pointer :: z(:, :) ! layer depth (m) over snow only
real(r8), pointer :: dz(:, :) ! layer thickness depth (m) over snow on
real(r8), pointer :: zi(:, :) ! interface depth (m) over snow only
```



## A.84 Module spmdGathScatMod (Source File: spmdGathScatMod.F90)

Perform SPMD gather and scatter operations.

USES:

```
use spmdMod
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
```

PUBLIC MEMBER FUNCTIONS:

```
public scatter_data_from_master, gather_data_to_master, allgather_data

interface scatter_data_from_master
 module procedure scatter_1darray_int
 module procedure scatter_1darray_real
 module procedure scatter_2darray_int
 module procedure scatter_2darray_real
end interface

interface gather_data_to_master
 module procedure gather_1darray_int
 module procedure gather_1darray_real
 module procedure gather_2darray_int
 module procedure gather_2darray_real
end interface

interface allgather_data
 module procedure allgather_1darray_int
 module procedure allgather_1darray_real
 module procedure allgather_2darray_int
 module procedure allgather_2darray_real
end interface
```

REVISION HISTORY:

Author: Mariana Vertenstein

### A.84.1 spmd\_compute\_mpigs

INTERFACE:

```
subroutine spmd_compute_mpigs (clmlevel, nfact, numtot, numperproc, &
 displs, indexi)
```

DESCRIPTION:

Compute arguments for gatherv, scatternv for vectors

## USES:

```

 use clmtype , only : nameg, namel, namec, namep, ocnrof, lndrof
 use decompMod, only : get_proc_bounds, get_proc_total
 #if (defined RTM)
 use RunoffMod, only : get_proc_rof_bounds, get_proc_rof_total
 #endif

```

## ARGUMENTS:

```

 implicit none
 character(len=*), intent(in) :: clmlevel ! type of input data
 integer, intent(in) :: nfact ! multiplicative factor for
 integer, intent(out) :: numtot ! total number of elements (
 integer, intent(out) :: numperproc(0:npes-1) ! per-PE number of items to
 integer, intent(out) :: displs(0:npes-1) ! per-PE displacements
 integer, intent(out) :: indexi ! beginning array index (gri

```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.2 scatter\_1darray\_int**

## INTERFACE:

```

 subroutine scatter_1darray_int (ilocal, iglobal, clmlevel)

```

## DESCRIPTION:

Wrapper routine to scatter integer 1d array

## USES:

## ARGUMENTS:

```

 implicit none
 integer, pointer, dimension(:) :: ilocal(:) !local read data
 integer, pointer, dimension(:) :: iglobal(:) !global read data
 character(len=*), intent(in) :: clmlevel !type of input data

```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.3 scatter\_1darray\_real**

## INTERFACE:

```

 subroutine scatter_1darray_real (rlocal, rglobal, clmlevel)

```

## DESCRIPTION:

Wrapper routine to scatter 1d real array from master processor

## ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:) :: rlocal !local read data
real(r8), pointer, dimension(:) :: rglobal !global read data
character(len=*), intent(in) :: clmlevel !input data type
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

## A.84.4 scatter\_2darray\_int

## INTERFACE:

```
subroutine scatter_2darray_int (ilocal, iglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to scatter 2d integer array from master processor

## ARGUMENTS:

```
implicit none
integer, pointer, dimension(:, :) :: ilocal !local read data
integer, pointer, dimension(:, :) :: iglobal !global read data
character(len=*), intent(in) :: clmlevel !type of input data
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

## A.84.5 scatter\_2darray\_real

## INTERFACE:

```
subroutine scatter_2darray_real (rlocal, rglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to scatter 2d integer array from master processor

## ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:, :) :: rlocal !local read data
real(r8), pointer, dimension(:, :) :: rglobal !global read data
character(len=*), intent(in) :: clmlevel !type of input data
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---



**A.84.6 gather\_1darray\_int**

## INTERFACE:

```
subroutine gather_1darray_int (ilocal, iglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to gather 1d integer array on master processor

## ARGUMENTS:

```
implicit none
integer, pointer, dimension(:) :: ilocal !output data
integer, pointer, dimension(:) :: iglobal !output data
character(len=*), intent(in) :: clmlevel !input data type
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.7 gather\_1darray\_real**

## INTERFACE:

```
subroutine gather_1darray_real (rlocal, rglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to gather 1d real array on master processor

## ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:) :: rlocal !output data
real(r8), pointer, dimension(:) :: rglobal !output data
character(len=*), intent(in) :: clmlevel !input data type
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.8 gather\_2darray\_int**

## INTERFACE:

```
subroutine gather_2darray_int (ilocal, iglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to gather 2d integer array on master processor

#### ARGUMENTS:

```
implicit none
integer, pointer, dimension(:, :) :: ilocal !read data
integer, pointer, dimension(:, :) :: iglobal !global data
character(len=*), intent(in) :: clmlevel !type of input data
```

#### REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.84.9 gather\_2darray\_real

#### INTERFACE:

```
subroutine gather_2darray_real (rlocal, rglobal, clmlevel)
```

#### DESCRIPTION:

Wrapper routine to gather 2d real array

#### ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:, :) :: rlocal !local data
real(r8), pointer, dimension(:, :) :: rglobal !global data
character(len=*), intent(in) :: clmlevel !type of input data
```

#### REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.84.10 allgather\_1darray\_int

#### INTERFACE:

```
subroutine allgather_1darray_int (ilocal, iglobal, clmlevel)
```

#### DESCRIPTION:

Wrapper routine to perform an allgather of 1d integer array

#### ARGUMENTS:

```
implicit none
integer, pointer, dimension(:) :: ilocal !output data
integer, pointer, dimension(:) :: iglobal !output data
character(len=*), intent(in) :: clmlevel !input data type
```

#### REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.11 allgather\_1darray\_real**

## INTERFACE:

```
subroutine allgather_1darray_real (rlocal, rglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to perform an allgather of 1d real array

## ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:) :: rlocal !output data
real(r8), pointer, dimension(:) :: rglobal !output data
character(len=*), intent(in) :: clmlevel !input data type
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.12 allgather\_2darray\_int**

## INTERFACE:

```
subroutine allgather_2darray_int (ilocal, iglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to perform an allgather of 2d integer array

## ARGUMENTS:

```
implicit none
integer, pointer, dimension(:, :) :: ilocal !read data
integer, pointer, dimension(:, :) :: iglobal !global data
character(len=*), intent(in) :: clmlevel !type of input data
```

## REVISION HISTORY:

Author: Mariana Vertenstein

---

**A.84.13 allgather\_2darray\_real**

## INTERFACE:

```
subroutine allgather_2darray_real (rlocal, rglobal, clmlevel)
```

## DESCRIPTION:

Wrapper routine to perform an allgather of 2d real array

ARGUMENTS:

```
implicit none
real(r8), pointer, dimension(:,:) :: rlocal !local data
real(r8), pointer, dimension(:,:) :: rglobal !global data
character(len=*), intent(in) :: clmlevel !type of input data
```

REVISION HISTORY:

Author: Mariana Vertenstein



## A.85 Module spmdMod (Source File: spmdMod.F90)

SPMD initialization

REVISION HISTORY:

Author: Mariana Vertenstein

---

### A.85.1 spmd\_init

INTERFACE:

subroutine spmd\_init

DESCRIPTION:

MPI initialization (number of cpus, processes, tids, etc)

ARGUMENTS:

implicit none

REVISION HISTORY:

Author: Mariana Vertenstein



## A.86 Module subgridAveMod (Source File: subgridAveMod.F90)

Utilities to perform subgrid averaging

USES:

```
use shr_kind_mod, only: r8 => shr_kind_r8
use clmtype
use clm_varcon, only : spval
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: p2c ! Perform an average from pfts to columns
public :: p2l ! Perform an average from pfts to landunits
public :: p2g ! Perform an average from pfts to gridcells
public :: c2l ! Perform an average from columns to landunits
public :: c2g ! Perform an average from columns to gridcells
public :: l2g ! Perform an average from landunits to gridcells
```

```
interface p2c
 module procedure p2c_1d
 module procedure p2c_2d
 module procedure p2c_1d_filter
 module procedure p2c_2d_filter
end interface
interface p2l
 module procedure p2l_1d
 module procedure p2l_2d
end interface
interface p2g
 module procedure p2g_1d
 module procedure p2g_2d
end interface
interface c2l
 module procedure c2l_1d
 module procedure c2l_2d
end interface
interface c2g
 module procedure c2g_1d
 module procedure c2g_2d
end interface
interface l2g
 module procedure l2g_1d
 module procedure l2g_2d
end interface
```

REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---



**A.86.1 p2c\_1d**

## INTERFACE:

```
subroutine p2c_1d (lbp, ubp, lbc, ubc, parr, carr, p2c_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from pfts to columns.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft
integer , intent(in) :: lbc, ubc ! beginning and ending col
real(r8), intent(in) :: parr(lbp:ubp) ! pft array
real(r8), intent(out) :: carr(lbc:ubc) ! column array
character(len=*), intent(in) :: p2c_scale_type ! scale type
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.2 p2c\_2d**

## INTERFACE:

```
subroutine p2c_2d (lbp, ubp, lbc, ubc, num2d, parr, carr, p2c_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from landunits to gridcells.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: parr(lbp:ubp,num2d) ! pft array
real(r8), intent(out) :: carr(lbc:ubc,num2d) ! column array
character(len=*), intent(in) :: p2c_scale_type ! scale type
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.3 p2c\_1d\_filter**

## INTERFACE:

```
subroutine p2c_1d_filter (numfc, filterc, pftarr, colarr)
```

## DESCRIPTION:

perform pft to column averaging for single level pft arrays

## USES:

```
use clm_varpar, only : max_pft_per_col
```

## ARGUMENTS:

```
implicit none
integer , intent(in) :: numfc
integer , intent(in) :: filterc(numfc)
real(r8), pointer :: pftarr(:)
real(r8), pointer :: colarr(:)
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.4 p2c\_2d\_filter**

## INTERFACE:

```
subroutine p2c_2d_filter (lev, numfc, filterc, pftarr, colarr)
```

## DESCRIPTION:

perform pft to column averaging for multi level pft arrays

## USES:

```
use clm_varpar, only : max_pft_per_col
```

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lev
integer , intent(in) :: numfc
integer , intent(in) :: filterc(numfc)
real(r8), pointer :: pftarr(:, :)
real(r8), pointer :: colarr(:, :)
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.5 p2l\_1d**

## INTERFACE:

```
subroutine p2l_1d (lbp, ubp, lbc, ubc, lbl, ubl, parr, larr, &
 p2c_scale_type, c2l_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from pfts to landunits  
 Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: lbl, ubl ! beginning and ending lan
real(r8), intent(in) :: parr(lbp:ubp) ! input column array
real(r8), intent(out) :: larr(lbl:ubl) ! output landunit array
character(len=*), intent(in) :: p2c_scale_type ! scale factor type for av
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.6 p2l\_2d**

## INTERFACE:

```
subroutine p2l_2d(lbp, ubp, lbc, ubc, lbl, ubl, num2d, parr, larr, &
 p2c_scale_type, c2l_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from pfts to landunits  
 Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: lbl, ubl ! beginning and ending lan
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: parr(lbp:ubp,num2d) ! input pft array
real(r8), intent(out) :: larr(lbl:ubl,num2d) ! output gridcell array
character(len=*), intent(in) :: p2c_scale_type ! scale factor type for av
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

### A.86.7 p2g\_1d

#### INTERFACE:

```
subroutine p2g_1d(lbp, ubp, lbc, ubc, lbl, ubl, lbg, ubg, parr, garr, &
 p2c_scale_type, c2l_scale_type, l2g_scale_type)
```

#### DESCRIPTION:

Perfrom subgrid-average from pfts to gridcells.  
Averaging is only done for points that are not equal to "spval".

#### ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft i
integer , intent(in) :: lbc, ubc ! beginning and ending colum
integer , intent(in) :: lbl, ubl ! beginning and ending landu
integer , intent(in) :: lbg, ubg ! beginning and ending gridc
real(r8), intent(in) :: parr(lbp:ubp) ! input pft array
real(r8), intent(out) :: garr(lbg:ubg) ! output gridcell array
character(len=*), intent(in) :: p2c_scale_type ! scale factor type for av
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

#### REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

### A.86.8 p2g\_2d

#### INTERFACE:

```
subroutine p2g_2d(lbp, ubp, lbc, ubc, lbl, ubl, lbg, ubg, num2d, &
 parr, garr, p2c_scale_type, c2l_scale_type, l2g_scale_type)
```

#### DESCRIPTION:

Perfrom subgrid-average from pfts to gridcells.  
Averaging is only done for points that are not equal to "spval".

#### ARGUMENTS:

```
implicit none
integer , intent(in) :: lbp, ubp ! beginning and ending pft
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: lbl, ubl ! beginning and ending lan
integer , intent(in) :: lbg, ubg ! beginning and ending gri
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: parr(lbp:ubp,num2d) ! input pft array
real(r8), intent(out) :: garr(lbg:ubg,num2d) ! output gridcell array
character(len=*), intent(in) :: p2c_scale_type ! scale factor type for av
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.9 c2l\_1d**

## INTERFACE:

```
subroutine c2l_1d (lbc, ubc, lbl, ubl, carr, larr, c2l_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from columns to landunits  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc ! beginning and ending column indi
integer , intent(in) :: lbl, ubl ! beginning and ending landunit in
real(r8), intent(in) :: carr(lbc:ubc) ! input column array
real(r8), intent(out) :: larr(lbl:ubl) ! output landunit array
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.10 c2l\_2d**

## INTERFACE:

```
subroutine c2l_2d (lbc, ubc, lbl, ubl, num2d, carr, larr, c2l_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from columns to landunits  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc ! beginning and ending colum
integer , intent(in) :: lbl, ubl ! beginning and ending landu
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: carr(lbc:ubc,num2d) ! input column array
real(r8), intent(out) :: larr(lbl:ubl,num2d) ! output landunit array
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.11 c2g\_1d**

## INTERFACE:

```
subroutine c2g_1d(lbc, ubc, lbl, ubl, lbg, ubg, carr, garr, &
 c2l_scale_type, l2g_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from columns to gridcells.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: lbl, ubl ! beginning and ending lan
integer , intent(in) :: lbg, ubg ! beginning and ending lan
real(r8), intent(in) :: carr(lbc:ubc) ! input column array
real(r8), intent(out) :: garr(lbg:ubg) ! output gridcell array
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.12 c2g\_2d**

## INTERFACE:

```
subroutine c2g_2d(lbc, ubc, lbl, ubl, lbg, ubg, num2d, carr, garr, &
 c2l_scale_type, l2g_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from columns to gridcells.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbc, ubc ! beginning and ending col
integer , intent(in) :: lbl, ubl ! beginning and ending lan
integer , intent(in) :: lbg, ubg ! beginning and ending gri
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: carr(lbc:ubc,num2d) ! input column array
real(r8), intent(out) :: garr(lbg:ubg,num2d) ! output gridcell array
character(len=*), intent(in) :: c2l_scale_type ! scale factor type for av
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.13 l2g\_1d**

## INTERFACE:

```
subroutine l2g_1d(lbl, ubl, lbg, ubg, larr, garr, l2g_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from landunits to gridcells.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbl, ubl ! beginning and ending sub landun
integer , intent(in) :: lbg, ubg ! beginning and ending gridcell i
real(r8), intent(in) :: larr(lbl:ubl) ! input landunit array
real(r8), intent(out) :: garr(lbg:ubg) ! output gridcell array
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

---

**A.86.14 l2g\_2d**

## INTERFACE:

```
subroutine l2g_2d(lbl, ubl, lbg, ubg, num2d, larr, garr, l2g_scale_type)
```

## DESCRIPTION:

Perfrom subgrid-average from landunits to gridcells.  
Averaging is only done for points that are not equal to "spval".

## ARGUMENTS:

```
implicit none
integer , intent(in) :: lbl, ubl ! beginning and ending colu
integer , intent(in) :: lbg, ubg ! beginning and ending grid
integer , intent(in) :: num2d ! size of second dimension
real(r8), intent(in) :: larr(lbl:ubl,num2d) ! input landunit array
real(r8), intent(out) :: garr(lbg:ubg,num2d) ! output gridcell array
character(len=*), intent(in) :: l2g_scale_type ! scale factor type for av
```

## REVISION HISTORY:

Created by Mariana Vertenstein 12/03

## A.87 Module surfFileMod

### (Source File: surfFileMod.F90)

Contains methods for reading in surface data file and determining two-dimensional subgrid weights as well as writing out new surface dataset. When reading in the surface dataset, determines array which sets the PFT for each of the [maxpatch] patches and array which sets the relative abundance of the PFT. Also fills in the PFTs for vegetated portion of each grid cell. Fractional areas for these points pertain to "vegetated" area not to total grid area. Need to adjust them for fraction of grid that is vegetated. Also fills in urban, lake, wetland, and glacier patches.

#### USES:

```
use abortutils, only : endrun
#if (defined SCAM)
 use scamMod, only : initlonidx,initlatidx
#endif
```

#### PUBLIC TYPES:

```
implicit none
save
```

#### PUBLIC MEMBER FUNCTIONS:

```
public :: surfrd ! Read surface dataset and determine subgrid weights
```

#### REVISION HISTORY:

Created by Mariana Vertenstein

### A.87.1 surfdrd

#### INTERFACE:

```
subroutine surfdrd(veg, wt, &
 cam_longxy, cam_latixy, cam_numlon, cam_landfrac, cam_landmask)
```

#### DESCRIPTION:

Read the surface dataset and create subgrid weights. The model's surface dataset recognizes 5 basic land cover types within a grid cell: lake, wetland, urban, glacier, and vegetated. The vegetated portion of the grid cell is comprised of up to [maxpatch\_pft] PFTs. These subgrid patches are read in explicitly for each grid cell. This is in contrast to LSMv1, where the PFTs were built implicitly from biome types. Read surface boundary data with the exception of monthly lai,sai,and heights at top and bottom of canopy on [lsmlon] x [lsmlat] grid.

- o real edges of grid
- o integer number of longitudes per latitude



```

o real latitude of grid cell (degrees)
o real longitude of grid cell (degrees)
o integer surface type: 0 = ocean or 1 = land
o integer soil color (1 to 9) for use with soil albedos
o real soil texture, %sand, for thermal and hydraulic properties
o real soil texture, %clay, for thermal and hydraulic properties
o real % of cell covered by lake for use as subgrid patch
o real % of cell covered by wetland for use as subgrid patch
o real % of cell that is urban for use as subgrid patch
o real % of cell that is glacier for use as subgrid patch
o integer PFTs
o real % abundance PFTs (as a percent of vegetated area)
OFFLINE MODE ONLY:
Surface grid edges -- Grids do not have to be global.
If grid is read in from dataset, grid is assumed to be global
(does not have to be regular, however)
If grid is generated by model, grid does not have to be global but must the
define the north, east, south, and west edges:
o lsmedge(1) = northern edge of grid (degrees): > -90 and <= 90
o lsmedge(2) = eastern edge of grid (degrees) : see following notes
o lsmedge(3) = southern edge of grid (degrees): >= -90 and < 90
o lsmedge(4) = western edge of grid (degrees) : see following notes
For partial grids, northern and southern edges are any latitude
between 90 (North Pole) and -90 (South Pole). Western and eastern
edges are any longitude between -180 and 180, with longitudes
west of Greenwich negative. That is, western edge >= -180 and < 180;
eastern edge > western edge and <= 180.
For global grids, northern and southern edges are 90 (North Pole)
and -90 (South Pole). The western and eastern edges depend on
whether the grid starts at Dateline or Greenwich. Regardless,
these edges must span 360 degrees. Examples:
 West edge East edge

(1) Dateline : -180 to 180 (negative W of Greenw
(2) Greenwich (centered): 0 - dx/2 to 360 - dx/2
Grid 1 is the grid for offline mode
Grid 2 is the grid for cam and csm mode since the NCAR CAM
starts at Greenwich, centered on Greenwich

```

## USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use clm_varpar
use clm_varctl
use clm_varsur
use ncdio, only : check_dim, check_ret
use pftvarcon, only : noveg, crop
use fileutils, only : getfil
use spmdMod
use areaMod
#if (defined SCAM)
use getnetcdfdata
#endif

```

## ARGUMENTS:

```
implicit none
include 'netcdf.inc'
integer , intent(out) :: veg(lsmlon,lsmlat,maxpatch) ! PFT
real(r8), intent(out) :: wt(lsmlon,lsmlat,maxpatch) ! subgrid weights
real(r8), optional, intent(in) :: cam_longxy(:, :) ! cam lon values
real(r8), optional, intent(in) :: cam_latixy(:, :) ! cam lat values
integer , optional, intent(in) :: cam_numlon(:) ! cam number of long
real(r8), optional, intent(in) :: cam_landfrac(:, :) ! cam fractional lan
integer , optional, intent(in) :: cam_landmask(:, :) ! cam land mask
```

CALLED FROM:

```
subroutine initialize in module initializeMod
```

REVISION HISTORY:

Created by Mariana Vertenstein, Sam Levis and Gordon Bonan



## A.88 `system_cmd` (Source File: `system_cmd.c`)

### INTERFACE:

```
int system_cmd(const char *text)
```

### DESCRIPTION:

Wrapper to "C" system command. Allows all platforms to have a consistent interface that includes an error return code.

### REVISION HISTORY:



## A.89 Module system\_messages

### (Source File: system\_messages.F90)

Contains general purpose routines for checking system messages.

USES:

```
use abortutils, only : endrun
```

PUBLIC TYPES:

```
implicit none
save
```

PUBLIC MEMBER FUNCTIONS:

```
public :: allocation_err ! allocation error message
public :: netcdf_err ! netCD return error message
public :: iobin_err ! binary i/o error message
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

#### A.89.1 allocation\_err

INTERFACE:

```
subroutine allocation_err(ier, routine_name, array_name, nsize)
```

DESCRIPTION:

Issue error message after non-zero return from an allocate statement.

ARGUMENTS:

```
implicit none
integer, intent(in) :: ier ! status from allocate state
character(len=*), intent(in) :: routine_name ! routine name that invoked
character(len=*), intent(in), optional :: array_name ! array name
integer, intent(in), optional :: nsize ! size of attempted allocati
```

REVISION HISTORY:

Created by Mariana Vertenstein

---

#### A.89.2 netcdf\_err

INTERFACE:

```
subroutine netcdf_err (ier, routine_name)
```

## DESCRIPTION:

If error detected in netCDF call, issue error message and abort.

## ARGUMENTS:

```

 implicit none
 #include <netcdf.inc>
 integer, intent(in) :: ier ! return code from netCDF ca
 character(len=*), intent(in) :: routine_name ! calling routine name

```

## REVISION HISTORY:

Created by Mariana Vertenstein

---

**A.89.3 iobin\_err**

## INTERFACE:

```

 subroutine iobin_err (ier, routine_name)

```

## DESCRIPTION:

If error detected during i/o binary read/write

## ARGUMENTS:

```

 implicit none
 integer, intent(in) :: ier ! return code from netCDF call
 character(len=*), intent(in) :: routine_name ! calling routine name

```

## REVISION HISTORY:

Created by Mariana Vertenstein

## A.90 Module time\_manager

### (Source File: time\_manager.F90)

Provides generic interface to time/calendar management routines.

USES:

```

use shr_kind_mod, only: r8 => shr_kind_r8
use ESMF_TimeMgmtMod, only: &
 esmf_errhandlersettype, esmf_err_return, esmf_errprint, esmf_success, &
 esmf_time, esmf_timeinit, esmf_timeget, esmf_timegetdays, &
 esmf_timeincrement, esmf_timedecrement, &
 esmf_date, esmf_dateinit, esmf_gregorian, esmf_no_leap, esmf_dateget, &
 esmf_dateincrementsec, esmf_dateincrementday, esmf_datedecrement, &
 esmf_datediff, esmf_dategetfltdayofyear, &
 esmf_timemgr, esmf_timemgrinit, esmf_timemgradvance, &
 esmf_timemgrgetnstep, esmf_timemgrgetstepsize, &
 esmf_timemgrgetstartdate, esmf_timemgrgetbasedate, &
 esmf_timemgrlaststep, esmf_timemgrgetcurrdate, &
 esmf_timemgrgetprevdate, esmf_dateislater, &
 esmf_timemgrrestartwrite, esmf_timemgrrestartread
#if (defined SPMD)
 use spmdMod, only: masterproc, mpicom, MPI_INTEGER
#else
 use spmdMod, only: masterproc
#endif
 use abortutils, only: endrun

```

PUBLIC TYPES:

```

implicit none
private
save

```

PUBLIC MEMBER FUNCTIONS:

```

public :: &
 timemgr_init, &! time manager initialization
 advance_timestep, &! increment timestep number
 get_step_size, &! return step size in seconds
 get_nstep, &! return timestep number
 get_curr_date, &! return date components at end of current t
 get_prev_date, &! return date components at beginning of cur
 get_start_date, &! return date components of the start date
 get_ref_date, &! return date components of the reference da
 get_curr_time, &! return components of elapsed time since re
 get_curr_calday, &! return calendar day at end of current time
 is_first_step, &! return true on first step of initial run
 is_first_restart_step, &! return true on first step of restart or br
 is_end_curr_day, &! return true on last timestep in current da
 is_end_curr_month, &! return true on last timestep in current mo
 is_last_step, &! return true on last timestep
 timemgr_write_restart, &! write info to file needed to restart the t
 timemgr_read_restart, &! read info from file needed to restart the
 timemgr_restart ! restart the time manager

```

PUBLIC DATA MEMBERS:



```

character(len=32), public :: &
 calendar = 'NO_LEAP' ! Calendar in date calculations ('NO_LEAP'

integer, parameter :: uninit_int = -999999999 !This is private to this mo

integer, public :: &
 dtime = uninit_int, &! timestep in seconds
 nestep = uninit_int, &! final timestep (or day if negative) numb
 nelapse = uninit_int, &! number of timesteps (or days if negative
 start_ymd = uninit_int, &! starting date for run in yearmdd format
 start_tod = 0, &! starting time of day for run in seconds
 stop_ymd = uninit_int, &! stopping date for run in yearmdd format
 stop_tod = 0, &! stopping time of day for run in seconds
 ref_ymd = uninit_int, &! reference date for time coordinate in ye
 ref_tod = 0 ! reference time of day for time coordinat

```

REVISION HISTORY:

### A.90.1 `timemgr_init`

INTERFACE:

```
subroutine timemgr_init()
```

DESCRIPTION:

Initialize the ESMF time manager.  
 NOTE - This assumes that the namelist variables  
 have been set before this routine is called.

ARGUMENTS:

```
implicit none
```

LOCAL VARIABLES:

```

character(len=*) , parameter :: sub = 'timemgr_init'
character(len=len(calendar)) :: cal
integer :: rc ! return code
integer :: cal_type ! calendar type
type(esmf_time) :: step_size ! timestep size
type(esmf_date) :: start_date ! start date for run
type(esmf_date) :: stop_date ! stop date for run
type(esmf_date) :: ref_date ! reference date for time coordinate

```

Some backwards compatibility stuff:

```

type(esmf_time) :: diff
integer :: ntspday, ndays, nsecs
logical :: islater

```

REVISION HISTORY:

### A.90.2 timemgr\_restart

#### INTERFACE:

```
subroutine timemgr_restart()
```

#### DESCRIPTION:

Restart the ESMF time manager.

NOTE - Assumptions:

- 1) The namelist variables have been set before this routine is called. The stop date is the only thing that can be changed by the user on restart.
- 2) Restart data have been read on the master process before this routine is called.  
(timemgr\_read\_restart called from control/restart.F90::read\_restart)

#### ARGUMENTS:

```
implicit none
```

#### LOCAL VARIABLES:

```
character(len=*) , parameter :: sub = 'timemgr_restart'
integer :: rc ! return code
type(esmf_date) :: start_date ! start date for run
type(esmf_date) :: stop_date ! stop date for run
type(esmf_date) :: curr_date ! date of data in restart file
logical :: islater
integer :: ymd, tod
integer :: ier !error code
```

Some backwards compatibility stuff:

```
type(esmf_time) :: diff
integer :: ntspday, ndays, nsecs
```

#### REVISION HISTORY:

### A.90.3 timemgr\_print

#### INTERFACE:

```
subroutine timemgr_print()
```

#### DESCRIPTION:

Print out ESMF time manager information.

#### ARGUMENTS:

```
implicit none
```

## LOCAL VARIABLES:

```

character(len=*), parameter :: sub = 'timemgr_print'
integer :: rc
integer :: day, sec, ymd, tod
character(len=32) :: cal ! Calendar to use in date calculations.
integer :: & ! Data required to restart time manager:
 type = uninit_int, &! calendar type
 nstep = uninit_int, &! current step number
 step_days = uninit_int, &! days component of timestep size
 step_sec = uninit_int, &! seconds component of timestep size
 start_ymd = uninit_int, &! start date
 start_tod = uninit_int, &! start time of day
 stop_ymd = uninit_int, &! stop date
 stop_tod = uninit_int, &! stop time of day
 ref_ymd = uninit_int, &! reference date
 ref_tod = uninit_int, &! reference time of day
 curr_ymd = uninit_int, &! current date
 curr_tod = uninit_int ! current time of day

```

## REVISION HISTORY:

---

**A.90.4** `advance_timestep`

## INTERFACE:

```
subroutine advance_timestep()
```

## DESCRIPTION:

Increment the timestep number.

## ARGUMENTS:

```
implicit none
```

## LOCAL VARIABLES:

```

character(len=*), parameter :: sub = 'advance_timestep'
integer :: rc

```

## REVISION HISTORY:

---

**A.90.5** `get_step_size`

## INTERFACE:

```
function get_step_size()
```

## DESCRIPTION:

Return the step size in seconds.

## ARGUMENTS:

implicit none

*RETURN VALUE:*

integer :: get\_step\_size

## LOCAL VARIABLES:

character(len=\*), parameter :: sub = 'get\_step\_size'  
integer :: days, seconds  
integer :: rc

## REVISION HISTORY:

---

## A.90.6 get\_nstep

## INTERFACE:

function get\_nstep()

## DESCRIPTION:

Return the timestep number.

## ARGUMENTS:

implicit none

*RETURN VALUE:*

integer :: get\_nstep

## LOCAL VARIABLES:

character(len=\*), parameter :: sub = 'get\_nstep'  
integer :: rc

## REVISION HISTORY:

---

**A.90.7** `get_curr_date`

## INTERFACE:

```
subroutine get_curr_date(yr, mon, day, tod, offset)
```

## DESCRIPTION:

Return date components valid at end of current timestep with an optional offset (positive or negative) in seconds.

## ARGUMENTS:

```
implicit none
integer, intent(out) ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)

integer, optional, intent(in) :: offset ! Offset from current time in
 ! seconds. Positive for future
 ! times, negative for previous
 ! times.
```

## LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'get_curr_date'
integer :: rc
type(esmf_date) :: date
type(esmf_time) :: off
integer :: ymd
```

## REVISION HISTORY:

**A.90.8** `get_prev_date`

## INTERFACE:

```
subroutine get_prev_date(yr, mon, day, tod)
```

## DESCRIPTION:

Return date components valid at beginning of current timestep.

## ARGUMENTS:

```
implicit none
integer, intent(out) ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)
```

## LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'get_prev_date'
integer :: rc
type(esmf_date) :: date
integer :: ymd
```

## REVISION HISTORY:

---

**A.90.9 get\_start\_date**

## INTERFACE:

```
subroutine get_start_date(yr, mon, day, tod)
```

## DESCRIPTION:

Return date components valid at beginning of initial run.

## ARGUMENTS:

```
implicit none
integer, intent(out) ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)
```

## LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'get_start_date'
integer :: rc
type(esmf_date) :: date
integer :: ymd
```

## REVISION HISTORY:

---

**A.90.10 get\_ref\_date**

## INTERFACE:

```
subroutine get_ref_date(yr, mon, day, tod)
```

## DESCRIPTION:

Return date components of the reference date.

## ARGUMENTS:

```

implicit none
integer, intent(out) ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)

```

## LOCAL VARIABLES:

```

character(len=*), parameter :: sub = 'get_ref_date'
integer :: rc
type(esmf_date) :: date
integer :: ymd

```

## REVISION HISTORY:

---

**A.90.11** `get_curr_time`

## INTERFACE:

```

subroutine get_curr_time(days, seconds)

```

## DESCRIPTION:

Return time components valid at end of current timestep. Current time is the time interval between the current date and the reference date.

## ARGUMENTS:

```

implicit none
integer, intent(out) ::&
 days, &! number of whole days in time interval
 seconds ! remaining seconds in time interval

```

## LOCAL VARIABLES:

```

character(len=*), parameter :: sub = 'get_curr_time'
integer :: rc
type(esmf_date) :: cdate, rdate
type(esmf_time) :: diff
logical :: islater

```

## REVISION HISTORY:

---

**A.90.12 get\_curr\_calday**

## INTERFACE:

```
function get_curr_calday(offset)
```

## DESCRIPTION:

Return calendar day at end of current timestep with optional offset.  
Calendar day 1.0 = 0Z on Jan 1.

## ARGUMENTS:

```
implicit none
integer, optional, intent(in) :: offset ! Offset from current time in
 ! seconds. Positive for future
 ! times, negative for previous
 ! times.
```

*RETURN VALUE:*

```
real(r8) :: get_curr_calday
```

## LOCAL VARIABLES:

```
character(len=*) , parameter :: sub = 'get_curr_calday'
integer :: rc
type(esmf_date) :: date
type(esmf_time) :: off
integer :: ymd, tod
```

## REVISION HISTORY:

**A.90.13 is\_end\_curr\_day**

## INTERFACE:

```
function is_end_curr_day()
```

## DESCRIPTION:

Return true if current timestep is last timestep in current day.

## ARGUMENTS:

```
implicit none
```

*RETURN VALUE:*

```
logical :: is_end_curr_day
```

## LOCAL VARIABLES:



```

integer ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)

```

REVISION HISTORY:

---

#### A.90.14 `is_end_curr_month`

INTERFACE:

```
function is_end_curr_month()
```

DESCRIPTION:

Return true if current timestep is last timestep in current month.

ARGUMENTS:

```
implicit none
```

*RETURN VALUE:*

```
logical :: is_end_curr_month
```

LOCAL VARIABLES:

```

integer ::&
 yr, &! year
 mon, &! month
 day, &! day of month
 tod ! time of day (seconds past 0Z)

```

REVISION HISTORY:

---

#### A.90.15 `is_first_step`

INTERFACE:

```
function is_first_step()
```

DESCRIPTION:

Return true on first step of initial run only.

ARGUMENTS:

```
implicit none
```

*RETURN VALUE:*

```
logical :: is_first_step
```

LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'is_first_step'
```

```
integer :: rc
```

```
integer :: nstep
```

REVISION HISTORY:

### A.90.16 is\_first\_restart\_step

INTERFACE:

```
function is_first_restart_step()
```

DESCRIPTION:

Return true on first step of restart run only.

ARGUMENTS:

```
implicit none
```

*RETURN VALUE:*

```
logical :: is_first_restart_step
```

REVISION HISTORY:

### A.90.17 is\_last\_step

INTERFACE:

```
function is_last_step()
```

DESCRIPTION:

Return true on last timestep.

ARGUMENTS:

```
implicit none
```

*RETURN VALUE:*

```
logical :: is_last_step
```

LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'is_last_step'
integer :: rc
```

REVISION HISTORY:

---

### A.90.18 timemgr\_write\_restart

INTERFACE:

```
subroutine timemgr_write_restart(ftn_unit)
```

DESCRIPTION:

Write information needed on restart to a binary Fortran file. It is assumed that this routine is called only from the master proc if in SPMD mode.

ARGUMENTS:

```
implicit none
integer, intent(in) :: ftn_unit ! Fortran unit number
```

LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'timemgr_write_restart'
integer :: rc ! return code
```

REVISION HISTORY:

---

### A.90.19 timemgr\_read\_restart

INTERFACE:

```
subroutine timemgr_read_restart(ftn_unit)
```

DESCRIPTION:

Read information needed on restart from a binary Fortran file. It is assumed that this routine is called only from the master proc if in SPMD mode.

ARGUMENTS:

```
implicit none
integer, intent(in) :: ftn_unit ! Fortran unit number
```

## LOCAL VARIABLES:

```
character(len=*), parameter :: sub = 'timemgr_read_restart'
integer :: rc ! return code
```

## REVISION HISTORY:

---

**A.90.20 chkrc**

## INTERFACE:

```
subroutine chkrc(rc, mes)
```

## DESCRIPTION:

Check time manager return code. If an error occurs, print the appropriate error message and abort the run.

## ARGUMENTS:

```
implicit none
integer, intent(in) :: rc ! return code from time management
 ! library
character(len=*), intent(in) :: mes ! error message
```

## REVISION HISTORY:

---

**A.90.21 to\_upper**

## INTERFACE:

```
function to_upper(str)
```

## DESCRIPTION:

Convert character string to upper case. Use achar and iachar intrinsics to ensure use of ascii collating sequence.

## ARGUMENTS:

```
implicit none
character(len=*), intent(in) :: str ! String to convert to upper case
```

*RETURN VALUE:*

```
character(len=len(str)) :: to_upper
```

## LOCAL VARIABLES:

```
integer :: i ! Index
integer :: aseq ! ascii collating sequence
character(len=1) :: ctmp ! Character temporary
```

## REVISION HISTORY:

Author: B. Eaton, July 2001

# Index

averaging, [14](#)

CAM, *see* Community Atmosphere Model

cam mode, [6](#), [7](#)

CCSM, *see* Community Climate System Model

clmlevel, [19](#)

clumps, [11](#)

column, [9](#)

Community Atmosphere Model, [1](#), [6](#)

Community Climate System Model, [1](#), [6](#)

CPL6, *see* flux coupler

CPP variables

    COUP\_CAM, [7](#)

    COUP\_CSM, [3](#), [5](#), [7](#), [8](#), [19](#)

    DGVM, [3–8](#), [19](#)

    OFFLINE, [5](#), [7](#), [8](#)

    RTM, [5](#), [7](#), [8](#), [19](#)

    SPMD, [6](#), [11](#)

    VOC, [4](#)

Cray Streaming Directives, [11](#), [12](#)

Cray X1, [1](#), [11](#), [12](#)

CSDs, *see* Cray Streaming Directives

csm mode, [6](#), [7](#)

DEBUG, [25](#)

DGVM, *see* Dynamic Global Vegetation Model

directories

    biogeochem/, [3](#)

    biogeophys/, [3](#)

    main/, [3](#)

    mksrfddata/, [3](#)

    riverroute/, [3](#)

    src/, [3](#)

    tools/convert\_ascii/, [15](#)

Dynamic Global Vegetation Model, [3](#)

Earth Simulator, [1](#)

eflx\_lh\_vege, [16](#)

files

    abortutils.F90, [153](#)

    accFldsMod.F90, [155](#)

    accumulMod.F90, [159](#)

    areaMod.F90, [165](#)

    atm\_lndMod.F90, [181](#)

    atmdrvMod.F90, [185](#)

    BalanceCheckMod.F90, [28](#)

    BareGroundFluxesMod.F90, [31](#)

    Biogeophysics1Mod.F90, [35](#)

    Biogeophysics2Mod.F90, [39](#)

    BiogeophysicsLakeMod.F90, [43](#)

    CanopyFluxesMod.F90, [47](#)

    clm\_csmMod.F90, [191](#)

    clm\_varcon.F90, [201](#)

    clm\_varctl.F90, [203](#)

    clm\_varpar.F90, [205](#)

    clm\_varsur.F90, [207](#)

    clmtype.F90, [209](#)

    clmtypeInitMod.F90, [233](#)

    controlMod.F90, [247](#)

    decompMod.F90, [253](#)

    DGVMAllocationMod.F90, [53](#)

    DGVMEcosystemDynMod.F90, [57](#)

    DGVMEestablishmentMod.F90, [65](#)

    DGVMFireMod.F90, [67](#)

    DGVMKillMod.F90, [69](#)

    DGVMLightMod.F90, [71](#)

    DGVMMod.F90, [73](#)

    DGVMMortalityMod.F90, [81](#)

    DGVMReproductionMod.F90, [83](#)

    DGVMRestMod.F90, [85](#)

    DGVMTurnoverMod.F90, [87](#)

    do\_close\_dispose.F90, [269](#)

    do\_restwrite.F90, [271](#)

    driver.F90, [3](#), [273](#)

    DriverInitMod.F90, [89](#)

    fileutils.F90, [277](#)

    filterMod.F90, [283](#)

    FracWetMod.F90, [91](#)

    FrictionVelocityMod.F90, [93](#)

    getdatetime.F90, [285](#)

    histFileMod.F90, [15](#), [287](#)

    histFldsMod.F90, [15](#), [16](#), [303](#)

- Hydrology1Mod.F90, 97
- Hydrology2Mod.F90, 101
- HydrologyLakeMod.F90, 103
- inicFileMod.F90, 20, 311
- initGridCellsMod.F90, 315
- initializeMod.F90, 321
- iniTimeConst.F90, 305
- iniTimeVar.F90, 307
- iobinary.F90, 325
- lnd2atmMod.F90, 331
- lp\_coupling.F90, 333
- mkgridMod.F90, 339
- mklai.F90, 343
- mklanwat.F90, 345
- mkpft.F90, 347
- mkrank.F90, 349
- mksoicol.F90, 351
- mksoitex.F90, 353
- mksrf\_glacier.nc, 15
- mksrf\_lai.nc, 15
- mksrf\_lanwat.nc, 15
- mksrf\_pft.nc, 15
- mksrf\_soicol\_clm2.nc, 15
- mksrf\_soitex.10level.nc, 15
- mksrf\_urban.nc, 15
- mksrfdatMod.F90, 355
- mkurban.F90, 357
- mpiinc.F90, 359
- nanMod.F90, 361
- ncdio.F90, 363
- pft2colMod.F90, 375
- pftvarcon.F90, 377
- program\_csm.F90, 379
- program\_off.F90, 381
- QSatMod.F90, 105
- restFileMod.F90, 383
- RtmMod.F90, 107
- RunoffMod.F90, 113
- snowdp2lev.F90, 389
- SnowHydrologyMod.F90, 123
- SoilHydrologyMod.F90, 129
- SoilTemperatureMod.F90, 135
- spmdGathScatMod.F90, 391
- spmdMod.F90, 399
- STATICEcossysDynMod.F90, 119
- subgridAveMod.F90, 401
- SurfaceAlbedoMod.F90, 141
- SurfaceRadiationMod.F90, 147
- surfFileMod.F90, 409
- system\_cmd.c, 413
- system\_messages.F90, 415
- time\_manager.F90, 417
- TridiagonalMod.F90, 149
- VOCEmissionMod.F90, 151
- flux coupler, 1, 6, 7
- fname, 17
- Fortran 90, 1, 9, 16
- gridcell, 9
- landunit, 9
- load balancing, 11
- long\_name, 17
- Message Passing Interface, 1, 6, 7, 11, 12, 18
- modules
  - abortutils, 153
  - accFldsMod, 155
  - accumulMod, 19, 159
  - areaMod, 165
  - atm\_lndMod, 7, 181
  - atmdrvMod, 185
  - BalanceCheckMod, 25, 28
  - BareGroundFluxesMod, 31
  - Biogeophysics1Mod, 35
  - Biogeophysics2Mod, 39
  - BiogeophysicsLakeMod, 43
  - CanopyFluxesMod, 47
  - clm\_csmMod, 19, 191
  - clm\_varcon, 201
  - clm\_varctl, 203
  - clm\_varpar, 205
  - clm\_varsur, 207
  - clmtype, 10, 11, 16–20, 209
  - clmtypeInitMod, 11, 233
  - controlMod, 247
  - decompMod, 253
  - DGVMAallocationMod, 53
  - DGVMEcosystemDynMod, 57
  - DGVMEestablishmentMod, 65
  - DGVMMMod, 73
  - DGVMRestMod, 19
  - DriverInitMod, 89
  - fileutils, 277
  - filterMod, 283
  - FireMod, 67
  - FracWetMod, 91
  - FrictionVelocityMod, 93
  - histFileMod, 14, 287
  - histFldsMod, 16, 303

- Hydrology1Mod, 97
- Hydrology2Mod, 101
- HydrologyLakeMod, 103
- inicFileMod, 311
- initGridcellsMod, 315
- initializeMod, 7, 321
- iobinary, 19, 20, 325
- KillMod, 69
- LightMod, 71
- lnd2atmMod, 331
- lp\_coupling, 333
- mkgridMod, 339
- mklai, 343
- mkpft, 347
- mksrfdatMod, 355
- MortalityMod, 81
- mpiinc, 359
- nanMod, 361
- ncdioMod, 363
- pft2colMod, 14, 375
- pftvarcon, 377
- QSatMod, 105
- ReproductionMod, 83
- restDGVMod, 85
- restFileMod, 18, 19, 383
- RtmMod, 19, 25, 107
- RunoffMod, 18, 113
- SnowHydrologyMod, 123
- SoilHydrologyMod, 129
- SoilTemperatureMod, 135
- spmdGathScatMod, 391
- spmdMod, 399
- STATICEcosysDynMod, 119
- subgridAveMod, 14, 401
- SurfaceAlbedoMod, 141
- SurfaceRadiationMod, 147
- surfFileMod, 409
- system\_messages, 415
- time\_manager, 417
- TridiagonalMod, 149
- TurnoverMod, 87
- VOCeMissionMod, 151
- MPI, *see* Message Passing Interface
- MSP, *see* Multi-Streaming Processor
- Multi-Streaming Processor, 11, 12
- namelist variables
  - BRNCH\_RETAIN\_CASENAME, 18
  - clump\_pproc, 12
  - FINIDAT, 20
  - HIST\_CRTINIC, 20
  - HIST\_DOV2XY, 16
  - nclumps, 12
  - netCDF, 15, 20
  - offline mode, 6
  - OMP\_NUM\_THREADS, 12
  - OpenMP, 1, 11, 12, 18
  - output fields
    - FCEV, 16
    - H2OSOILLIQ, 22
    - SNOWDP, 17
    - T\_GRND, 21
  - PFT, *see* plant functional type
  - pft\_pstate\_type, 11
  - plant functional type, 9
  - pointers, 9
  - ptr\_col, 17
  - ptr\_gcell, 17
  - ptr\_lunit, 17
  - ptr\_pft, 16–18
  - ptr\_roflnd, 18
  - ptr\_rofocn, 18
  - rbuf1dc, 19
  - rbuf1dg, 19
  - rbuf1dl, 19
  - rbuf1dp, 19
  - routines
    - add\_fld1d, 16–18, 299
    - add\_fld2d, 16, 17, 300
    - add\_subscript, 301
    - advance\_timestep, 7, 420
    - allgather\_1darray\_int, 395
    - allgather\_1darray\_real, 396
    - allgather\_2darray\_int, 396
    - allgather\_2darray\_real, 396
    - Allocation, 6, 53
    - allocation\_err, 415
    - alltoall\_chunk\_to\_clump, 335
    - alltoall\_clump\_to\_chunk, 335
    - alltoall\_clump\_to\_chunk\_init, 334
    - areaave, 167
    - areaini, 165
    - areaini\_point, 173
    - areamap, 168
    - areamap\_point, 175
    - areaovr, 169
    - areaovr\_point, 176



- atm\_getgrid, 8, 186
- atm\_openfile, 187
- atm\_readdata, 187
- atmdrv, 185
- atmlnd\_drv, 7, 182
- atmlnd\_ini, 7, 181
- BalanceCheck, 5, 28
- BareGroundFluxes, 4, 31
- Biogeophysics1, 4, 35
- Biogeophysics2, 4, 39
- BiogeophysicsLake, 4, 43
- BuildNatVegFilter, 5, 80
- BuildSnowFilter, 5, 127
- c2g, 6, 14
- c2g\_1d, 407
- c2g\_2d, 407
- c2l, 14
- c2l\_1d, 406
- c2l\_2d, 406
- CanopyFluxes, 4, 47
- cellarea\_global, 171
- cellarea\_regional, 170
- celledge\_global, 172
- celledge\_regional, 171
- check\_dim, 363
- check\_ret, 364
- check\_var, 364
- chkrc, 429
- CombineSnowLayers, 5, 125
- Combo, 127
- compat\_check, 196
- control\_init, 7, 11, 249
- control\_print, 250
- control\_spm, 250
- create\_grid\_offline, 340
- csm\_compat, 197
- csm\_dosndrcv, 3, 194
- csm\_fxave, 5, 196
- csm\_initialize, 8, 192
- csm\_recv, 3, 195
- csm\_recvgrid, 7, 193
- csm\_send, 5, 195
- csm\_sendalb, 8, 194
- csm\_setup, 192
- csm\_shutdown, 192
- DGVM\_EcosystemDyn, 4, 6, 58
- DGVM\_EcosystemDynini, 7, 57
- DGVM\_Respiration, 4, 59
- DivideSnowLayers, 5, 126
- do\_close\_dispose, 269
- do\_inicwrite, 313
- do\_restwrite, 271
- Drainage, 5, 133
- driver, 3, 6, 12, 273
- DriverInit, 4, 89
- EcosystemDyn, 4, 120
- EcosystemDynini, 7, 119
- endrun, 153
- Establishment, 6, 65
- extract\_accum\_field\_ml, 161
- extract\_accum\_field\_sl, 161
- Fire, 6, 67
- FireSeason, 4, 61
- FracWet, 4, 91
- FrictionVelocity, 4, 93
- gather\_1darray\_int, 394
- gather\_1darray\_real, 394
- gather\_2darray\_int, 394
- gather\_2darray\_real, 395
- gatherWeightsDGVM, 6, 78
- get\_clump\_bounds, 4–6, 12, 259
- get\_clump\_cell\_id\_coord, 255
- get\_clump\_coord\_id, 257
- get\_clump\_gcell\_info, 259
- get\_clump\_ncells\_id, 257
- get\_clump\_ncells\_proc, 256
- get\_clump\_owner\_id, 256
- get\_curr\_calday, 425
- get\_curr\_date, 5, 422
- get\_curr\_time, 424
- get\_filename, 277
- get\_gcell\_info, 258
- get\_gcell\_xyind, 261
- get\_nclumps, 255
- get\_nstep, 5, 421
- get\_prev\_date, 422
- get\_proc\_bounds, 260
- get\_proc\_clumps, 4, 261
- get\_proc\_global, 261
- get\_proc\_rof\_bounds, 116
- get\_proc\_rof\_global, 117
- get\_proc\_rof\_total, 116
- get\_proc\_total, 260
- get\_ref\_date, 5, 423
- get\_size\_dim1, 372
- get\_sn\_cols1d, 266
- get\_sn\_land1d, 266
- get\_sn\_pfts1d, 267
- get\_start\_date, 423
- get\_step\_size, 420

getavu, 279  
 getdatetime, 285  
 getfil, 278  
 getflag, 298  
 getname, 297  
 getnum, 329  
 global\_sum fld1d, 198  
 global\_sum fld2d, 198  
 header, 323  
 hfields\_1dinfo, 295  
 hfields\_normalize, 293  
 hfields\_write, 295  
 hfields\_zero, 293  
 histDGVM, 6, 76  
 htape\_addfld, 291  
 htape\_create, 294  
 htape\_timeconst, 294  
 htapes\_build, 290  
 htapes\_fielddlist, 291  
 htapes\_wrapup, 6, 296  
 Hydrology1, 4, 12, 97  
 Hydrology2, 5, 101  
 HydrologyLake, 5, 103  
 Infiltration, 5, 130  
 inicfields, 20, 312  
 inicfile, 6, 20, 311  
 inicperp, 313  
 init\_accum\_field, 159  
 init\_atm2lnd\_flux\_type, 246  
 init\_atm2lnd\_state\_type, 245  
 init\_column\_cstate\_type, 242  
 init\_column\_efflux\_type, 243  
 init\_column\_estate\_type, 242  
 init\_column\_pstate\_type, 241  
 init\_column\_type, 235  
 init\_column\_wflux\_type, 243  
 init\_column\_wstate\_type, 242  
 init\_energy\_balance\_type, 236  
 init\_gridcell\_dgvstate\_type, 244  
 init\_gridcell\_pstate\_type, 244  
 init\_gridcell\_type, 235  
 init\_gridcell\_wflux\_type, 246  
 init\_landunit\_pstate\_type, 244  
 init\_landunit\_type, 235  
 init\_lnd2atm\_flux\_type, 246  
 init\_lnd2atm\_state\_type, 245  
 init\_pft\_cflux\_type, 240  
 init\_pft\_dflux\_type, 241  
 init\_pft\_DGVMecophys\_constants, 237  
 init\_pft\_ecophys\_constants, 237  
 init\_pft\_efflux\_type, 239  
 init\_pft\_energy\_type, 238  
 init\_pft\_mflux\_type, 239  
 init\_pft\_pdgvstate\_type, 238  
 init\_pft\_pstate\_type, 11, 237  
 init\_pft\_type, 234  
 init\_pft\_vflux\_type, 240  
 init\_pft\_wflux\_type, 240  
 init\_pft\_wstate\_type, 238  
 init\_water\_balance\_type, 236  
 initAccClmtype, 8, 157  
 initAccFlds(), 155  
 initClmtype, 7, 234  
 initDecomp, 7, 11, 254  
 initFilters, 7, 12, 283  
 initGridcells, 7, 315  
 initGridcellsGlob, 318  
 initHistFlds, 8, 16, 303  
 inithistFlds, 16  
 initialize, 7, 321  
 iniTimeConst, 7, 305  
 iniTimeVar, 8, 307  
 interpa2s, 189  
 interpa2si, 188  
 interpMonthlyVeg, 3, 120  
 iobin\_err, 416  
 is\_end\_curr\_day, 425  
 is\_end\_curr\_month, 426  
 is\_first\_restart\_step, 427  
 is\_first\_step, 426  
 is\_last\_step, 427  
 Kill, 5, 69  
 l2g, 14  
 l2g\_1d, 408  
 l2g\_2d, 408  
 landunit\_crop\_noncompete, 318  
 landunit\_special, 317  
 landunit\_veg\_compete, 316  
 landunit\_veg\_noncompete, 317  
 Light, 6, 71  
 list\_index, 298  
 LitterSOM, 4, 62  
 lnd2atm, 5, 331  
 lp\_coupling\_finalize, 334  
 lp\_coupling\_init, 7, 333  
 lpj, 5, 73  
 lpjreset1, 6, 74  
 lpjreset2, 6, 75  
 makel2a, 331  
 map\_dc2sn\_ml1\_int, 263

- map\_dc2sn\_ml1\_real, 263
- map\_dc2sn\_sl\_int, 262
- map\_dc2sn\_sl\_real, 262
- map\_sn2dc\_ml1\_int, 265
- map\_sn2dc\_ml1\_real, 265
- map\_sn2dc\_sl\_int, 264
- map\_sn2dc\_sl\_real, 264
- masterlist\_addfld, 289
- masterlist\_change\_timeavg, 291
- masterlist\_make\_active, 290
- masterlist\_printflds, 288
- mkarbinit, 308
- mkglacier, 337
- mkgrid\_cam, 341
- mkgrid\_offline, 339
- mklai, 343
- mklanwat, 345
- mkmxovr, 177
- mkpfts, 347
- mkrank, 349
- mksoicol, 351
- mksoitex, 353
- mksrfdat, 355
- mkurban, 357
- MoninObukIni, 4, 95
- Mortality, 6, 81
- ncd\_defvar, 20–22, 364
- ncd\_ioglobal\_int\_1d, 369
- ncd\_ioglobal\_int\_2d, 370
- ncd\_ioglobal\_int\_3d, 371
- ncd\_ioglobal\_int\_var, 368
- ncd\_ioglobal\_real\_1d, 369
- ncd\_ioglobal\_real\_2d, 370
- ncd\_ioglobal\_real\_3d, 371
- ncd\_ioglobal\_real\_var, 368
- ncd\_iolocal, 20–22
- ncd\_iolocal\_int\_1d, 365
- ncd\_iolocal\_int\_2d, 366
- ncd\_iolocal\_real\_1d, 366
- ncd\_iolocal\_real\_2d, 367
- netcdf\_err, 415
- opnfil, 279
- p2c, 4, 5, 14
- p2c\_1d, 402
- p2c\_1d\_filter, 403
- p2c\_2d, 402
- p2c\_2d\_filter, 403
- p2g, 14
- p2g\_1d, 405
- p2g\_2d, 405
- p2l, 14
- p2l\_1d, 404
- p2l\_2d, 404
- pft2col, 375
- pftconrd, 7, 378
- PhaseChange, 138
- Phenology, 4, 60
- pointer\_index, 301
- print\_accum\_fields, 160
- program\_csm, 7, 379
- program\_off, 6, 381
- putfil, 278
- QSat, 4, 105
- read\_grid\_offline, 340
- readin, 19, 20
- readin\_1d\_array\_int, 325
- readin\_1darray\_real, 326
- readin\_2d\_arrayint, 326
- readin\_2darray\_real, 327
- readMonthlyVegetation, 4, 121
- relavu, 280
- Reproduction, 5, 83
- resetTimeConstDGVM, 6, 8, 77
- resetWeightsDGVM, 6, 77
- restart, 6, 18, 383
- restart\_accum, 19, 163
- restart\_biogeophys, 19, 385
- restart\_biogeophysics, 19
- restart\_coupler, 19, 197
- restart\_dgvm, 19, 85
- restart\_history, 297
- restart\_rtm, 19, 112
- restart\_setup, 18, 384
- restart\_time, 19, 384
- restart\_wrapup, 386
- Rtm, 111
- Rtmfluxini(), 109
- Rtmgridini, 7, 107
- Rtmlandini, 8, 108
- Rtmriverflux, 5, 109
- scatter\_1darray\_int, 392
- scatter\_1darray\_real, 392
- scatter\_2darray\_int, 393
- scatter\_2darray\_real, 393
- set\_dgvm\_filename, 79
- set\_filename, 278
- set\_hist\_filename, 298
- set\_init\_filename, 314
- set\_proc\_rof\_bounds, 114
- set\_restart\_filename, 387

set\_roflnd, 113  
set\_rofocn, 114  
shell\_cmd, 281  
shr\_orb\_params, 6  
SnowAge, 5, 145  
SnowAlbedo, 4, 6, 142  
SnowCompaction, 5, 124  
snowdp2lev, 389  
SnowWater, 5, 123  
SoilAlbedo, 4, 6, 143  
SoilTemperature, 5, 135  
SoilThermProp, 136  
SoilWater, 5, 131  
spmd\_compute\_mpigs, 391  
spmd\_init, 399  
StabilityFunc, 94  
StabilityFunc2, 95  
Stomata, 4, 50  
subroutine scam\_field\_offsets, 372  
SurfaceAlbedo, 4, 6, 141  
SurfaceRadiation, 4, 147  
SurfaceRunoff, 5, 129  
surfrd, 7, 409  
system\_cmd, 413  
timemgr\_init, 7, 418  
timemgr\_print, 419  
timemgr\_read\_restart, 428  
timemgr\_restart, 419  
timemgr\_write\_restart, 428  
to\_upper, 429  
Tridiagonal, 4, 149  
Turnover, 5, 87  
TwoStream, 4, 6, 144  
update\_accum\_field\_ml, 162  
update\_accum\_field\_sl, 162  
update\_hbuf, 5, 292  
update\_hbuf\_field, 292  
updateAccFlds, 5, 156  
UpdateGlobal, 111  
UpdateInput, 110  
UpdateRunoff, 115  
varsur\_alloc, 7, 207  
varsur\_dealloc, 8, 208  
VOCEmission, 4, 151  
write\_diagnostic, 5, 276  
write\_rest\_pfile, 386  
wrtout, 19, 20  
wrtout\_1d\_array\_int, 327  
wrtout\_1d\_array\_real, 327  
wrtout\_2d\_array\_int, 328  
wrtout\_2darray\_real, 328  
runoff%lnd, 18  
runoff%ocn, 18  
set\_lake, 17  
typexy, 18  
units, 17



ORNL/TM-2004/119

**INTERNAL DISTRIBUTION**

- |                      |                                     |
|----------------------|-------------------------------------|
| 1. M. L. Branstetter | 10. P. H. Worley                    |
| 2. J. B. Drake       | 11. T. Zacharia                     |
| 3. M. W. Ham         | 12. Central Research Library        |
| 4-8. F. M. Hoffman   | 13. ORNL Laboratory Records RC      |
| 9. J. A. Nichols     | 14-26. ORNL Laboratory Records OSTI |

**EXTERNAL DISTRIBUTION**

- 27. Gordon Bonan  
bonan@ucar.edu
- 28. Tom Henderson  
hender@ucar.edu
- 29. Samuel Levis  
slevis@ucar.edu
- 30. Keith Oleson  
oleson@ucar.edu
- 31. Peter Thornton  
thornton@ucar.edu
- 32. Mariana Vertenstein  
mvertens@ucar.edu