LIS 4.0 Documentation

Submitted under Task Agreement GSFC-CT-2

Cooperative Agreement Notice (CAN) CAN-00OES-01

Increasing Interoperability and Performance of
Grand Challenge Applications in the Earth, Space, Life, and Microgravity Sciences

February 11, 2005

Revision 2.0

History:

Revision	Summary of Changes	Date
2.0	Milestone "K" submission	Feb 11, 2005
1.0	Milestone "G" submission	May 7, 2004

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

Contents

1 Routine/Function Prologues **13**

1.0.1 absoft_release_cache.F90 (Source File: absoft.F90) 13

1.1 Fortran: Module Interface baseforcing_module.F90 (Source File: baseforcing_module.F90) 13

1.1.1 LIS_get_base_forcing (Source File: baseforcing_module.F90) 13

1.1.2 LIS_baseforcing_init (Source File: baseforcing_module.F90) 13

1.1.3 forcing_init (Source File: baseforcing_module.F90) 14

1.1.4 init_baseforcing (Source File: baseforcing_module.F90) 14

1.1.5 get (Source File: baseforcing_module.F90) 15

1.1.6 time_interp (Source File: baseforcing_module.F90) 15

1.1.7 ld_getbaseforcing (Source File: baseforcing_module.F90) 16

1.1.8 scatter_data (Source File: baseforcing_module.F90) 16

1.1.9 scatter_elev (Source File: baseforcing_module.F90) 17

1.1.10 check_error (Source File: check_error.F90) 17

1.1.11 define_gds.F90 (Source File: define_gds.F90) 17

1.2 Fortran: Module Interface domain_module.F90 (Source File: domain_module.F90) 19

1.2.1 forcing_init (Source File: domain_module.F90) 19

1.2.2 (Source File: domain_module.F90) 19

1.2.3 read_domain (Source File: domain_module.F90) 20

1.3 Fortran: Module Interface driverpardef_module.F90 (Source File: driverpardef_module.F90) 20

1.3.1 def_driverpar_structs (Source File: driverpardef_module.F90) 21

1.4 Fortran: Module Interface drv_output_mod (Source File: drv_output_mod.F90) 21

1.4.1 drv_writevar_bin (Source File: drv_output_mod.F90) 21

1.4.2 t2gr.F90 (Source File: drv_output_mod.F90) 26

1.4.3 elevadjust.F90 (Source File: elevadjust.F90) 26

1.4.4 endrun.F90 (Source File: endrun.F90) 28

1.5 Fortran: Module Interface filename_mod.F90 (Source File: filename_mod.F90) 29

1.5.1 avhrr_file_1km (Source File: filename_mod.F90) 29

1.5.2 avhrr_file_1km (Source File: filename_mod.F90) 29

1.5.3 modis_file_1km (Source File: filename_mod.F90) 30

1.5.4 avhrr_laifilename (Source File: filename_mod.F90) 30

1.5.5 avhrr_laifilename (Source File: filename_mod.F90) 32

1.5.6 avhrr_file_5km (Source File: filename_mod.F90) 33

1.5.7 avhrr_file_5km (Source File: filename_mod.F90) 35

1.5.8 modis_file_2 (Source File: filename_mod.F90) 37

1.5.9 modis_file_5km (Source File: filename_mod.F90) 39

1.5.10 avhrr_g_file (Source File: filename_mod.F90) 41

1.5.11 modis_g_file (Source File: filename_mod.F90) 42

1.6 Fortran: Module Interface grid_module.F90 (Source File: grid_module.F90) 42

1.7 Fortran: Module Interface grid_spmdMod.F90 (Source File: grid_spmdMod.F90) 43

1.7.1 allocate_gdd (Source File: grid_spmdMod.F90) 43

1.7.2 grid_spmd_init (Source File: grid_spmdMod.F90) 44

1.8 Fortran: Module Interface gswp_module.F90 (Source File: gswp_module.F90) 44

1.9 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90) 48

1.9.1 ld_domain_init (Source File: lisdrv_module.F90) 49

1.9.2 setup_timeMgr (Source File: lisdrv_module.F90) 49
1.9.3 LIS_ticktime (Source File: lisdrv_module.F90) 49
1.9.4 LIS_domain_init (Source File: lisdrv_module.F90) 50
1.9.5 setnch (Source File: lisdrv_module.F90) 53
1.9.6 getdomain (Source File: lisdrv_module.F90) 53
1.9.7 getlsm (Source File: lisdrv_module.F90) 54
1.9.8 getnch (Source File: lisdrv_module.F90) 54
1.9.9 getnc (Source File: lisdrv_module.F90) 54
1.9.10 getnr (Source File: lisdrv_module.F90) 55
1.9.11 gettileindex (Source File: lisdrv_module.F90) 55
1.9.12 getmaxt (Source File: lisdrv_module.F90) 55
1.9.13 getforcing (Source File: lisdrv_module.F90) 56
1.9.14 getnmif (Source File: lisdrv_module.F90) 56
1.9.15 LIS_endofrun (Source File: lisdrv_module.F90) 56
1.9.16 dist_gindex (Source File: lisdrv_module.F90) 57
1.10 Fortran: Module Interface lis_indices_module.F90 (Source File: lis_indices_module.F90) 59
1.10.1 lis_set_indices (Source File: lis_indices_module.F90) 59
1.10.2 lis_get_run_slat (Source File: lis_indices_module.F90) 59
1.10.3 lis_get_run_wlon (Source File: lis_indices_module.F90) 59
1.10.4 lis_get_run_nlat (Source File: lis_indices_module.F90) 59
1.10.5 lis_get_run_elon (Source File: lis_indices_module.F90) 60
1.10.6 lis_get_run_lat_res (Source File: lis_indices_module.F90) 60
1.10.7 lis_get_run_lon_res (Source File: lis_indices_module.F90) 60
1.10.8 lis_get_data_slat (Source File: lis_indices_module.F90) 60
1.10.9 lis_get_data_wlon (Source File: lis_indices_module.F90) 60
1.10.10 lis_get_data_nlat (Source File: lis_indices_module.F90) 60
1.10.11 lis_get_data_elon (Source File: lis_indices_module.F90) 61
1.10.12 lis_get_data_lat_res (Source File: lis_indices_module.F90) 61
1.10.13 lis_get_data_lon_res (Source File: lis_indices_module.F90) 61
1.10.14 lis_global_to_local_row_offset (Source File: lis_indices_module.F90) . . 61
1.10.15 lis_global_to_local_col_offset (Source File: lis_indices_module.F90) . . 61
1.10.16 lis_log_msg.F90 (Source File: lis_log_msg.F90) 62
1.10.17 lis_log_blocked_msg (Source File: lis_log_msg.F90) 62
1.11 Fortran: Module Interface lis_module.F90 (Source File: lis_module.F90) . . 63
1.12 Fortran: Module Interface lis_openfileMod.F90 (Source File: lis_openfileMod.F90) 67
1.12.1 lis_set_filename (Source File: lis_openfileMod.F90) 67
1.12.2 lis_read_file (Source File: lis_openfileMod.F90) 68
1.12.3 retrieve_script (Source File: lis_openfileMod.F90) 70
1.13 Fortran: Module Interface lis_flush (Source File: lis_utilities.F90) 70
1.14 Fortran: Module Interface lsm_module.F90 (Source File: lsm_module.F90) . 70
1.14.1 LIS_setuplsm (Source File: lsm_module.F90) 70
1.14.2 LIS_lsm_main (Source File: lsm_module.F90) 70
1.14.3 LIS_force2tile (Source File: lsm_module.F90) 71
1.14.4 LIS_readrestart (Source File: lsm_module.F90) 71
1.14.5 LIS_writerestart (Source File: lsm_module.F90) 71
1.14.6 LIS_lsm_output (Source File: lsm_module.F90) 71
1.14.7 LIS_setDynlsm (Source File: lsm_module.F90) 71
1.14.8 lsm_tile_allocate (Source File: lsm_module.F90) 72

1.14.9 lsm_setup (Source File: lsm_module.F90) 72
1.14.10 run_lsm (Source File: lsm_module.F90) 73
1.14.11 lsm_readrestart (Source File: lsm_module.F90) 73
1.14.12 write_output (Source File: lsm_module.F90) 74
1.14.13 setDynParams (Source File: lsm_module.F90) 74
1.14.14 lsm_f2t (Source File: lsm_module.F90) 74
1.14.15 lsm_writerestart (Source File: lsm_module.F90) 75
1.14.16 makepdsn (Source File: makepdsn.F90) 75
1.15 Fortran: Module Interface mpishorthand.F90 (Source File: mpishorthand.F90) 75
1.16 Fortran: Module Interface obsprecipforcing_module.F90 (Source File: obsprecipforcing_module.F90) 76
1.16.1 LIS_obsprecipforcing_init (Source File: obsprecipforcing_module.F90) 76
1.16.2 LIS_get_obsprecip_forcing (Source File: obsprecipforcing_module.F90) 76
1.16.3 precip_forcing_init (Source File: obsprecipforcing_module.F90) 77
1.16.4 get_precip_forcing (Source File: obsprecipforcing_module.F90) 77
1.16.5 scatter_precip_data (Source File: obsprecipforcing_module.F90) . . . 78
1.17 Fortran: Module Interface obsradforcing_module.F90 (Source File: obsradforcing_module.F90) 78
1.17.1 LIS_obsradforcing_init (Source File: obsradforcing_module.F90) . . . 79
1.17.2 LIS_get_obsrad_forcing (Source File: obsradforcing_module.F90) . . . 79
1.17.3 rad_forcing_init (Source File: obsradforcing_module.F90) 79
1.17.4 get_obsrad_forcing (Source File: obsradforcing_module.F90) 80
1.17.5 scatter_rad_data (Source File: obsradforcing_module.F90) 81
1.18 Fortran: Module Interface opendap_module.F90 (Source File: opendap_module.F90) 81
1.18.1 opendap_init (Source File: opendap_module.F90) 82
1.18.2 opendap_readcard (Source File: opendap_module.F90) 83
1.18.3 reset_lis_filepaths (Source File: opendap_module.F90) 83
1.18.4 init_parm_vars (Source File: opendap_module.F90) 84
1.18.5 set_parm_lat (Source File: opendap_module.F90) 86
1.19 Fortran: Module Interface precision.F90 (Source File: precision.F90) 88
1.19.1 readcard.F90 (Source File: readcard.F90) 88
1.19.2 check_timestep (Source File: readcard.F90) 91
1.19.3 openfile (Source File: readcard.F90) 91
1.20 Fortran: Module Interface spmdMod.F90 (Source File: spmdMod.F90) . . . 93
1.20.1 spmd_init (Source File: spmdMod.F90) 93
1.20.2 stats.F90 (Source File: stats.F90) 94
1.21 Fortran: Module Interface string_utils (Source File: string_utils.F90) 95
1.21.1 to_upper (Source File: string_utils.F90) 95
1.21.2 tile_module.F90 (Source File: tile_module.F90) 96
1.22 Fortran: Module Interface tile_spmdMod.F90 (Source File: tile_spmdMod.F90) 97
1.22.1 allocate_tiledd (Source File: tile_spmdMod.F90) 97
1.22.2 tile_spmd_init (Source File: tile_spmdMod.F90) 97
1.23 Fortran: Module Interface time_manager.F90 (Source File: time_manager.F90) 98
1.23.1 timemgr_print (Source File: time_manager.F90) 99
1.23.2 timemgr_print (Source File: time_manager.F90) 100
1.23.3 advance_timestep (Source File: time_manager.F90) 100
1.23.4 get_step_size (Source File: time_manager.F90) 101
1.23.5 get_nstep (Source File: time_manager.F90) 101

1.23.6 get_curr_day (Source File: time_manager.F90) 102
1.23.7 get_prev_date (Source File: time_manager.F90) 102
1.23.8 get_start_date (Source File: time_manager.F90) 102
1.23.9 get_ref_date (Source File: time_manager.F90) 102
1.23.10 get_curr_time (Source File: time_manager.F90) 102
1.23.11 get_curr_calday (Source File: time_manager.F90) 103
1.23.12 is_last_step (Source File: time_manager.F90) 103
1.23.13 timemgr_write_restart (Source File: time_manager.F90) 103
1.23.14 timemgr_read_restart (Source File: time_manager.F90) 103
1.23.15 chkrc (Source File: time_manager.F90) 106
1.23.16 zterp.F90 (Source File: zterp.F90) 106
1.23.17 coszenith (Source File: zterp.F90) 111
1.23.18 localtime (Source File: zterp.F90) 113
1.24 Fortran: Module Interface lsm_pluginMod.F90 (Source File: lsm_pluginMod.F90) 114
1.24.1 lsm_plugin (Source File: lsm_pluginMod.F90) 115
1.25 Fortran: Module Interface baseforcing_pluginMod.F90 (Source File: baseforcing_pluginMod.F90) 117
1.25.1 baseforcing_plugin (Source File: baseforcing_pluginMod.F90) 117
1.26 Fortran: Module Interface precipforcing_pluginMod.F90 (Source File: precipforcing_pluginMod.F90) 118
1.26.1 precipforcing_plugin (Source File: precipforcing_pluginMod.F90) . . 119
1.27 Fortran: Module Interface radforcing_pluginMod.F90 (Source File: radforcing_pluginMod.F90) 120
1.27.1 radforcing_plugin (Source File: radforcing_pluginMod.F90) 120
1.28 Fortran: Module Interface domain_pluginMod.F90 (Source File: domain_pluginMod.F90) 121
1.28.1 domain_plugin (Source File: domain_pluginMod.F90) 121
1.28.2 calculate_domveg (Source File: calculate_domveg.F90) 122
1.28.3 create_vegtilespace (Source File: create_vegtilespace.F90) 126
1.29 Fortran: Module Interface elevdiff_pluginMod.F90 (Source File: elevdiff_pluginMod.F90) 135
1.29.1 elevdiff_plugin (Source File: elevdiff_pluginMod.F90) 135
1.30 Fortran: Module Interface lai_pluginMod.F90 (Source File: lai_pluginMod.F90) 135
1.30.1 lai_plugin (Source File: lai_pluginMod.F90) 136
1.31 Fortran: Module Interface landcover_pluginMod.F90 (Source File: landcover_pluginMod.F90) 136
1.31.1 landcover_plugin (Source File: landcover_pluginMod.F90) 137
1.32 Fortran: Module Interface soils_pluginMod.F90 (Source File: soils_pluginMod.F90) 137
1.32.1 soils_plugin (Source File: soils_pluginMod.F90) 138
1.32.2 read_color (Source File: read_color.F90) 138
1.32.3 read_faosoils (Source File: read_faoclay.F90) 139
1.32.4 read_faosoils (Source File: read_faosand.F90) 139
1.32.5 read_faosoils (Source File: read_faosilt.F90) 139
1.32.6 read_statsgosoils (Source File: read_statsgoclay.F90) 140
1.32.7 read_statsgosoils (Source File: read_statsgosand.F90) 140
1.32.8 read_statsgosoils (Source File: read_statsgosilt.F90) 140
1.32.9 read_umdavhrr_lc (Source File: read_umdavhrr_lc.F90) 141
1.32.10 read_umdavhrr_mask (Source File: read_umdavhrr_mask.F90) 141
1.32.11 read_elevdiff_gtopo30 (Source File: read_elevdiff_gtopo30.F90) 141
1.32.12 climatologylairead.F90 (Source File: climatologylairead.F90) 142
1.32.13 climatologysairead.F90 (Source File: climatologysairead.F90) 142

1.32.14 read_avhrrlai (Source File: read_avhrrlai.F90) 143
1.32.15 clmlairead.F90: (Source File: read_avhrrsai.F90) 146
1.32.16 bilinear_interp.F90 (Source File: bilinear_interp.F90) 149
1.33 Fortran: Module Interface bilinear_interpMod.F90 (Source File: bilinear_interpMod.F90) 150
1.33.1 allocate_bilinear_interp (Source File: bilinear_interpMod.F90) 151
1.33.2 bilinear_interp_input (Source File: bilinear_interpMod.F90) 152
1.33.3 compute_coord.F90 (Source File: compute_coord.F90) 153
1.33.4 compute_coord_gauss (Source File: compute_coord_gauss.F90) 154
1.33.5 conserv_interp.F90 (Source File: conserv_interp.F90) 156
1.34 Fortran: Module Interface conserv_interpMod.F90 (Source File: conserv_interpMod.F90) 157
1.34.1 allocate_interp (Source File: conserv_interpMod.F90) 157
1.34.2 get_field_pos.F90 (Source File: get_fieldpos.F90) 158
1.34.3 polfixs.F90 (Source File: polfixs.F90) 158
1.35 Fortran: Module Interface geosdomain_module.F90 (Source File: geosdomain_module.F90) 159
1.35.1 defnatgeos.F90 (Source File: geosdomain_module.F90) 159
1.36 Fortran: Module Interface geosdrv_module.F90 (Source File: geosdrv_module.F90) 160
1.37 Fortran: Module Interface geosopendap_module.F90 (Source File: geosopendap_module.F90) 160
1.37.1 reset_geos_filepaths (Source File: geosopendap_module.F90) 162
1.37.2 init_geos_vars (Source File: geosopendap_module.F90) 162
1.37.3 def_kgds (Source File: geosopendap_module.F90) 164
1.37.4 def_gridDesc (Source File: geosopendap_module.F90) 165
1.37.5 init_geos_ref_date (Source File: geosopendap_module.F90) 165
1.37.6 get_geos_index (Source File: geosopendap_module.F90) 166
1.37.7 set_geos_lat (Source File: geosopendap_module.F90) 166
1.37.8 getgeos.F90 (Source File: getgeos.F90) 168
1.38 Core Functions of getgeos 168
1.38.1 geosfile (Source File: getgeos.F90) 172
1.38.2 readgeoscrd.F90 (Source File: readgeoscrd.F90) 174
1.38.3 readgeos.F90 (Source File: readgeos.F90) 175
1.39 Core Functions of readgeos 175
1.39.1 time_interp_geos.F90 (Source File: time_interp_geos.F90) 181
1.40 Core Functions of time_interp_geos 182
1.41 Fortran: Module Interface gdasdomain_module.F90 (Source File: gdasdomain_module.F90) 185
1.41.1 defnatgdas.F90 (Source File: gdasdomain_module.F90) 185
1.42 Fortran: Module Interface gdasdrv_module.F90 (Source File: gdasdrv_module.F90) 186
1.43 Fortran: Module Interface gdasopendap_module.F90 (Source File: gdasopendap_module.F90) 187
1.43.1 reset_gdas_filepaths (Source File: gdasopendap_module.F90) 188
1.43.2 init_gdas_vars (Source File: gdasopendap_module.F90) 188
1.43.3 def_kgds (Source File: gdasopendap_module.F90) 189
1.43.4 set_gdas_lat (Source File: gdasopendap_module.F90) 189
1.43.5 twotoone (Source File: gdasopendap_module.F90) 190
1.43.6 get_kpds (Source File: gdasopendap_module.F90) 191
1.43.7 get_kgds (Source File: gdasopendap_module.F90) 192
1.43.8 get_gridDesc (Source File: gdasopendap_module.F90) 192

1.43.9 set_lb (Source File: gdasopendap_module.F90) 193
1.43.10 getgdas.F90 (Source File: getgdas.F90) 193
1.44 Core Functions of getgdas 194
1.44.1 gdasfile (Source File: getgdas.F90) 201
1.44.2 gdasfilef06 (Source File: getgdas.F90) 203
1.44.3 readgdascrd.F90 (Source File: readgdascrd.F90) 205
1.44.4 retgdas.F90 (Source File: retgdas.F90) 205
1.44.5 interp_gdas (Source File: retgdas.F90) 209
1.44.6 time_interp_gdas (Source File: time_interp_gdas.F90) 211
1.45 Core Functions of time_interp_gdas 211
1.46 Fortran: Module Interface ecmwfdomain_module.F90 (Source File: ecmwfdomain_module.F90) 214
1.46.1 defnatecmwf.F90 (Source File: ecmwfdomain_module.F90) 214
1.47 Fortran: Module Interface ecmwfdrv_module.F90 (Source File: ecmwfdrv_module.F90) 215
1.48 Fortran: Module Interface ecmwfopendap_module.F90 (Source File: ecmwfopendap_module.F90) 215
1.48.1 reset_ecmwf_filepaths (Source File: ecmwfopendap_module.F90) . . . 217
1.48.2 init_ecmwf_vars (Source File: ecmwfopendap_module.F90) 217
1.48.3 def_gridDesc (Source File: ecmwfopendap_module.F90) 219
1.48.4 init_ecmwf_ref_date (Source File: ecmwfopendap_module.F90) 219
1.48.5 get_ecmwf_index (Source File: ecmwfopendap_module.F90) 220
1.48.6 set_ecmwf_lat (Source File: ecmwfopendap_module.F90) 220
1.48.7 getecmwf.F90: (Source File: getecmwf.F90) 222
1.48.8 readecmwfcrd.F90 (Source File: readecmwfcrd.F90) 222
1.48.9 retecmwf (Source File: retecmwf.F90) 223
1.48.10 ret_inst3 (Source File: retecmwf.F90) 224
1.48.11 time_interp_ecmwf.F90 (Source File: time_interp_ecmwf.F90) 224
1.49 Core Functions of time_interp_geos 224
1.50 Fortran: Module Interface bergdomain_module.F90 (Source File: bergdomain_module.F90) 227
1.50.1 defnatberg.F90 (Source File: bergdomain_module.F90) 228
1.51 Fortran: Module Interface bergdrv_module.F90 (Source File: bergdrv_module.F90) 228
1.51.1 geogfill2 (Source File: geogfill2.F90) 229
1.51.2 getberg.F90 (Source File: getberg.F90) 229
1.51.3 readbergcrd.F90 (Source File: readbergcrd.F90) 230
1.51.4 recmwfmask.F90: (Source File: readbergmask.F90) 230
1.51.5 retberg.F90 (Source File: retberg.F90) 231
1.51.6 berggrid_2_gldasgrid (Source File: retberg.F90) 232
1.51.7 fillgaps (Source File: retberg.F90) 232
1.52 Fortran: Module Interface time_interp_berg (Source File: time_interp_berg.F90) 233
1.52.1 getnldas.F90 (Source File: getnldas.F90) 234
1.52.2 ncepfile.f: (Source File: getnldas.F90) 235
1.53 Fortran: Module Interface nldasdomain_module.F90 (Source File: nldasdomain_module.F90) 235
1.53.1 defnatnldas.F90 (Source File: nldasdomain_module.F90) 235
1.54 Fortran: Module Interface nldasdrv_module.F90 (Source File: nldasdrv_module.F90) 236
1.54.1 readnldascrd.F90 (Source File: readnldascrd.F90) 236
1.54.2 retnldas.F90 (Source File: retnldas.F90) 237

1.54.3 interp_nldas (Source File: retnldas.F90) 238
1.54.4 time_interp_geos.F90 (Source File: time_interp_geos.F90) 240
1.55 Core Functions of time_interp_geos 240
1.55.1 time_interp_nldas.F90 (Source File: time_interp_nldas.F90) 243
1.56 Core Functions of time_interp_nldas 243
1.56.1 getgswp.F90 (Source File: getgswp.F90) 243
1.57 Core Functions of getgswp 244
1.58 Fortran: Module Interface gswpdomain_module.F90 (Source File: gswpdomain_module.F90) 246
1.58.1 defnatgswp.F90 (Source File: gswpdomain_module.F90) 247
1.59 Fortran: Module Interface gswpdrv_module.F90 (Source File: gswpdrv_module.F90) 247
1.59.1 readgswp.F90 (Source File: readgswp.F90) 248
1.59.2 time_interp_gswp.F90 (Source File: time_interp_gswp.F90) 248
1.60 Core Functions of time_interp_gswp 248
1.61 Fortran: Module Interface cmapdomain_module.F90 (Source File: cmapdomain_module.F90) 251
1.61.1 defnatcmap.F90 (Source File: cmapdomain_module.F90) 251
1.61.2 allocate_cmap_ip (Source File: cmapdomain_module.F90) 252
1.61.3 def_cmap_ip_input (Source File: cmapdomain_module.F90) 253
1.62 Fortran: Module Interface cmapdrv_module.F90 (Source File: cmapdrv_module.F90) 255
1.62.1 getcmap.F90 (Source File: getcmap.F90) 256
1.62.2 cmapfile (Source File: getcmap.F90) 258
1.62.3 glbprecip_cmap.F90 (Source File: glbprecip_cmap.F90) 260
1.62.4 interp_cmap.F90 (Source File: interp_cmap.F90) 261
1.62.5 readcmapcrd.F90 (Source File: readcmapcrd.F90) 263
1.62.6 time_interp_cmap.F90 (Source File: time_interp_cmap.F90) 264
1.62.7 gethuff.F90 (Source File: gethuff.F90) 265
1.62.8 hufffile (Source File: gethuff.F90) 268
1.62.9 glbprecip_huff.F90 (Source File: glbprecip_huff.F90) 269
1.63 Fortran: Module Interface huffdomain_module.F90 (Source File: huffdomain_module.F90) 271
1.63.1 defnathuff.F90 (Source File: huffdomain_module.F90) 271
1.63.2 allocate_huff_ip (Source File: huffdomain_module.F90) 272
1.64 Fortran: Module Interface huffdrv_module.F90 (Source File: huffdrv_module.F90) 272
1.64.1 readhuffcrd.F90 (Source File: readhuffcard.F90) 273
1.64.2 time_interp_huff.F90 (Source File: time_interp_huff.F90) 273
1.64.3 getpers.F90 (Source File: getpers.F90) 274
1.64.4 persfile (Source File: getpers.F90) 275
1.64.5 glbprecip_pers.F90 (Source File: glbprecip_pers.F90) 277
1.65 Fortran: Module Interface persdomain_module.F90 (Source File: persdomain_module.F90) 278
1.65.1 defnatpers.F90 (Source File: persdomain_module.F90) 278
1.65.2 allocate_pers_ip (Source File: persdomain_module.F90) 279
1.66 Fortran: Module Interface persdrv_module.F90 (Source File: persdrv_module.F90) 279
1.66.1 readperscrd.F90 (Source File: readperscard.F90) 280
1.66.2 time_interp_pers.F90 (Source File: time_interp_pers.F90) 280
1.66.3 agrlwdn.F90 (Source File: agrlwdn.F90) 281
1.67 Fortran: Module Interface agrmetdomain_module.F90 (Source File: agrmetdomain_module.F90) 283

1.67.1 defnatagrmet.F90 (Source File: agrmetdomain_module.F90) 283
1.67.2 allocate_agr_ip (Source File: agrmetdomain_module.F90) 284
1.67.3 def_agr_ip_input (Source File: agrmetdomain_module.F90) 284
1.68 Fortran: Module Interface agrmetdrv_module.F90 (Source File: agrmetdrv_module.F90) 286
1.69 Fortran: Module Interface agrmetopendap_module.F90 (Source File: agrmetopendap_module.F90) 286
1.69.1 reset_agrmet_filepaths (Source File: agrmetopendap_module.F90) . . 287
1.69.2 init_agrmet_ref_date (Source File: agrmetopendap_module.F90) . . . 288
1.69.3 set_agrmet_index (Source File: agrmetopendap_module.F90) 288
1.69.4 getgrad.F90 (Source File: getgrad.F90) 289
1.69.5 agrSWfile: (Source File: getgrad.F90) 292
1.69.6 subroutine interp_agrmet_lw.F90 (Source File: interp_agrmet_lw.F90) 293
1.69.7 interp_agrmet_sw.F90 (Source File: interp_agrmet_sw.F90) 295
1.69.8 readagrmetcrd.F90 (Source File: readagrmetcrd.F90) 297
1.69.9 retagrlw.F90 (Source File: retagrlw.F90) 298
1.69.10 retglbSW.F90 (Source File: retglbSW.F90) 301
1.70 Fortran: Module Interface time_interp_agrmet.F90 (Source File: time_interp_agrmet.F90) 302
1.70.1 readtemplatecrd.F90 (Source File: readtemplatecrd.F90) 307
1.70.2 template_binout.F90 (Source File: template_binout.F90) 308
1.71 Fortran: Module Interface templatedrv_module.F90 (Source File: templatedrv_module.F90) 309
1.71.1 template_dynsetup.F90 (Source File: template_dynsetup.F90) 309
1.71.2 template_f2t.F90 (Source File: template_f2t.F90) 309
1.71.3 template_main.F90 (Source File: template_main.F90) 310
1.72 Fortran: Module Interface template_module.F90 (Source File: template_module.F90) 310
1.72.1 template_almaout.F90 (Source File: template_out.F90) 311
1.72.2 template_output.F90 (Source File: template_output.F90) 313
1.72.3 templaterst.F90 (Source File: templaterst.F90) 314
1.72.4 template_setup.F90 (Source File: template_setup.F90) 314
1.72.5 template_varder_ini (Source File: template_varder.F90) 315
1.72.6 template_writerst.F90 (Source File: template_writerst.F90) 316
1.72.7 template_writevars.F90 (Source File: template_writestats.F90) . . . 316
1.72.8 mapvegc.F90 (Source File: mapvegc.F90) 318
1.72.9 noah_alb.F90: (Source File: noah_alb.F90) 319
1.72.10 noah_almaout.F90 (Source File: noah_almaout.F90) 323
1.72.11 noah_atmdrv.F90 (Source File: noah_atmdrv.F90) 326
1.72.12 noah_binout.F90 (Source File: noah_binout.F90) 326
1.72.13 noah_coldstart.F90 (Source File: noah_coldstart.F90) 329
1.73 Fortran: Module Interface noahdrv_module.F90 (Source File: noahdrv_module.F90) 330
1.73.1 noah_dynsetup.F90 (Source File: noah_dynsetup.F90) 331
1.73.2 noah_gather.F90 (Source File: noah_gather.F90) 331
1.73.3 noah_gfrac.F90 (Source File: noah_gfrac.F90) 332
1.73.4 noah_binout.F90 (Source File: noah_gribout.F90) 335
1.73.5 noah_main.F90 (Source File: noah_main.F90) 340
1.74 Fortran: Module Interface noah_module.F90 (Source File: noah_module.F90) 355
1.74.1 noah_binout.F90 (Source File: noah_netcdfout.F90) 356
1.74.2 noah_output.F90 (Source File: noah_output.F90) 361
1.75 Fortran: Module Interface noahpardef_module.F90 (Source File: noahpardef_module.F90) 362

1.75.1 def_noahpar_struct (Source File: noahpardef_module.F90) 363
1.75.2 noahrst.F90 (Source File: noahrst.F90) 393
1.75.3 noah_scatter.F90 (Source File: noah_scatter.F90) 396
1.75.4 set_mxsnalb (Source File: noah_setmxsnalb.F90) 396
1.75.5 noahsetsoils (Source File: noah_setsoils.F90) 397
1.75.6 setnoahp.F90 (Source File: noah_settbot.F90) 400
1.75.7 noah_setup.F90 (Source File: noah_setup.F90) 401
1.75.8 noah_setvegparms.F90 (Source File: noah_setvegparms.F90) 402
1.75.9 noah_singlegather.F90 (Source File: noah_singlegather.F90) 404
1.75.10 noah_singleout.F90 (Source File: noah_singleout.F90) 407
1.75.11 noah_soiltype.F90 (Source File: noah_soil_typ.F90) 411
1.75.12 noah_totinit.F90 (Source File: noah_totinit.F90) 413
1.75.13 noah_varder.F90 (Source File: noah_varder.F90) 414
1.75.14 noah_varder_ini (Source File: noah_varder.F90) 414
1.75.15 noah_writerst.F90 (Source File: noah_writerst.F90) 415
1.75.16 noah_dump_restart (Source File: noah_writerst.F90) 418
1.75.17 noah_gather_restart (Source File: noah_writerst.F90) 419
1.75.18 readkpds.F90 (Source File: readkpds.F90) 424
1.75.19 readnoahcrd.F90 (Source File: readnoahcrd.F90) 424
1.75.20 sh2o_init.F90 (Source File: sh2o_init.F90) 425
1.75.21 noah_atmdrv.F90 (Source File: mos_atmdrv.F90) 426
1.75.22 mos_binout.F90 (Source File: mos_binout.F90) 427
1.75.23 mos_coldstart.F90 (Source File: mos_coldstart.F90) 429
1.76 Fortran: Module Interface mosdrv_module.F90 (Source File: mosdrv_module.F90) 430
1.76.1 noah_dynsetup.F90 (Source File: mos_dynsetup.F90) 430
1.76.2 mos_gather.F90 (Source File: mos_gather.F90) 431
1.77 Fortran: Module Interface mos_module.F90: (Source File: mos_module.F90) 432
1.77.1 mos_output.F90 (Source File: mos_output.F90) 433
1.78 Fortran: Module Interface mospardef_module.F90 (Source File: mospardef_module.F90) 434
1.78.1 mosrst.F90 (Source File: mosrst.F90) 435
1.78.2 mos_scatter.F90 (Source File: mos_scatter.F90) 437
1.78.3 mos_setup.F90 (Source File: mos_setup.F90) 437
1.78.4 mos_totinit.F90 (Source File: mos_totinit.F90) 438
1.78.5 mos_varder.F90 (Source File: mos_varder.F90) 440
1.78.6 mos_varder_ini (Source File: mos_varder.F90) 440
1.78.7 mos_writerst.F90 (Source File: mos_writerst.F90) 441
1.78.8 mos_writevars.F90 (Source File: mos_writestats.F90) 443
1.78.9 setnoahp.F90 (Source File: setmosp.F90) 447
1.78.10 hyssib_albedo.F90: (Source File: hyssib_albedo.F90) 448
1.78.11 hyssib_almaout.F90 (Source File: hyssib_almaout.F90) 452
1.78.12 hyssib_atmdrv.f (Source File: hyssib_atmdrv.F90) 454
1.78.13 hyssib_gridout.F90 (Source File: hyssib_binout.F90) 455
1.78.14 hyssib_coldstart.F90 (Source File: hyssib_coldstart.F90) 460
1.79 Fortran: Module Interface hyssibdrv_module.f: (Source File: hyssibdrv_module.F90) 461
1.79.1 hyssib_dynsetup.F90 (Source File: hyssib_dynsetup.F90) 462
1.79.2 hyssib_gather.F90 (Source File: hyssib_gather.F90) 462
1.79.3 hyssib_gfrac.F90 (Source File: hyssib_gfrac.F90) 463
1.79.4 hyssib_main.f (Source File: hyssib_main.F90) 463

1.79.5 hyssib_mapvegc.F90 (Source File: hyssib_mapvegc.F90) 471
1.80 Fortran: Module Interface hyssib_module.f: (Source File: hyssib_module.F90) 471
1.80.1 hyssib_output.F90 (Source File: hyssib_output.F90) 473
1.81 Fortran: Module Interface hyssibpardef_module.F90 (Source File: hyssibpardef_module.F90) 473
1.81.1 noah_totinit.F90 (Source File: hyssib_totinit.F90) 477
1.81.2 hyssib_writerst.F90 (Source File: hyssib_writerst.F90) 479
1.81.3 hyssib_writestats (Source File: hyssib_writestats.F90) 480
1.81.4 sethyssibp.f (Source File: sethyssibp.F90) 486
1.81.5 readssibcrd.F90 (Source File: readssibcrd.F90) 487
1.81.6 setssibp.f (Source File: setssibp.F90) 487
1.81.7 ssib_almaout.F90 (Source File: ssib_almaout.F90) 488
1.81.8 ssib_atmdrv.f (Source File: ssib_atmdrv.F90) 490
1.81.9 ssib_gridout.F90 (Source File: ssib_binout.F90) 491
1.81.10 ssib_coldstart.F90 (Source File: ssib_coldstart.F90) 495
1.81.11 ssib_com.F90 (Source File: ssib_com.F90) 499
1.82 Fortran: Module Interface ssibdrv_module.f: (Source File: ssibdrv_module.F90) 499
1.82.1 ssib_dynsetup.F90 (Source File: ssib_dynsetup.F90) 500
1.82.2 ssib_gather.F90 (Source File: ssib_gather.F90) 501
1.82.3 ssib_gfrac.F90 (Source File: ssib_gfrac.F90) 502
1.82.4 ssib_main.f (Source File: ssib_main.F90) 517
1.82.5 ssib_mapvegc.F90 (Source File: ssib_mapvegc.F90) 531
1.83 Fortran: Module Interface ssib_module.f: (Source File: ssib_module.F90) . . 532
1.83.1 ssib_output.F90 (Source File: ssib_output.F90) 534
1.84 Fortran: Module Interface ssibpardef_module.F90 (Source File: ssibpardef_module.F90) 535
1.84.1 ssibrst.F90 (Source File: ssibrst.F90) 536
1.84.2 ssib_scatter.F90 (Source File: ssib_scatter.F90) 539
1.84.3 ssib_setup.F90 (Source File: ssib_setup.F90) 539
1.84.4 ssib_singlegather.F90 (Source File: ssib_singlegather.F90) 541
1.84.5 ssib_singleout.F90 (Source File: ssib_singleout.F90) 545
1.84.6 ssib_totinit.F90 (Source File: ssib_totinit.F90) 548
1.84.7 ssib_varder.F90 (Source File: ssib_varder.F90) 549
1.84.8 ssib_varder_ini (Source File: ssib_varder.F90) 550
1.84.9 ssib_writerst.F90 (Source File: ssib_writerst.F90) 551
1.84.10 ssib_tileout.F90 (Source File: ssib_writestats.F90) 554
1.84.11 clm2_almaout.F90 (Source File: clm2_almaout.F90) 560
1.84.12 clm2_binout.F90 (Source File: clm2_binout.F90) 563
1.85 Fortran: Module Interface clm2drv_module.F90 (Source File: clm2drv_module.F90) 569
1.85.1 clm2_dynsetup.F90 (Source File: clm2_dynsetup.F90) 570
1.85.2 clm2_gather (Source File: clm2_gather.F90) 571
1.85.3 clm2_gribout.F90 (Source File: clm2_gribout.F90) 571
1.85.4 clmlairead.F90: (Source File: clm2lairead.F90) 580
1.85.5 clm2_output.F90 (Source File: clm2_output.F90) 581
1.86 Fortran: Module Interface clm2pardef_module.F90 (Source File: clm2pardef_module.F90) 582
1.86.1 def_clmpar_struct (Source File: clm2pardef_module.F90) 582
1.86.2 clm2_restart.F90 (Source File: clm2_restart.F90) 583
1.86.3 clm2_scatter (Source File: clm2_scatter.F90) 583
1.86.4 clm2_scatterlai (Source File: clm2_scatter.F90) 584

1.86.5 clm2_setup.F90 (Source File: clm2_setup.F90) 584
1.86.6 clm2_singlegather.F90 (Source File: clm2_singlegather.F90) 586
1.86.7 clm2_singleout.F90 (Source File: clm2_singleout.F90) 590
1.86.8 clm2_totinit.F90 (Source File: clm2_totinit.F90) 594
1.86.9 clm2_writestats.F90 (Source File: clm2_writestats.F90) 595
1.86.10 clm_varder.F90 (Source File: clm_varder.F90) 602
1.86.11 clm_varder_ini (Source File: clm_varder.F90) 602
1.86.12 clm_varder_init (Source File: clm_varder.F90) 603
1.86.13 iniTimeConst.F90 (Source File: iniTimeConst.F90) 604
1.86.14 iniTimeVar.F90 (Source File: iniTimeVar.F90) 611
1.86.15 readkpdsclm.F90 (Source File: readkpdsclm.F90) 617

1 Routine/Function Prologues

1.0.1 absoft_release_cache.F90 (Source File: absoft.F90)

This routine forces the explicit clearing of memory caches when using the absoft compiler.

REVISION HISTORY:

```
10 Sept 03    Jim Geiger  Initial Specification
```

INTERFACE:

```
subroutine absoft_release_cache()
```

1.1 Fortran: Module Interface baseforcing_module.F90 (Source File: baseforcing_module.F90)

This module contains interfaces and subroutines that controls the incorporation of model forcing

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

1.1.1 LIS_get_base_forcing (Source File: baseforcing_module.F90)

INTERFACE:

```
interface LIS_get_base_forcing
   module procedure ld_getbaseforcing
end interface
```

1.1.2 LIS_baseforcing_init (Source File: baseforcing_module.F90)

INTERFACE:

```
interface LIS_baseforcing_init
   module procedure init_baseforcing
end interface
```

1.1.3 forcing_init (Source File: baseforcing_module.F90)

Sets up functions for reading model forcing

INTERFACE:

```
subroutine forcing_init()
```

USES:

```
use baseforcing_pluginMod
```

1.1.4 init_baseforcing (Source File: baseforcing_module.F90)

Initializes model forcing variables and allocates memory

INTERFACE:

```
subroutine init_baseforcing()
```

USES:

```
use bilinear_interpMod, only: bilinear_interp_input, &
     allocate_bilinear_interp
use conserv_interpMod, only : conserv_interp_input, &
     allocate_conserv_interp
use lisdrv_module, only: lis, getforcing
use lis_indices_module, only: lis_nc_working, lis_nr_working
real :: gridDesci(50)
```

CONTENTS:

```
    gridDesci = 0
    call forcing_init()
    call allocate_forcing_mem()

#if ( ! defined OPENDAP )
    if ( masterproc ) then
#endif
       call defnatres(getforcing(),gridDesci)

       if(lis%f%interp.eq.1) then
          call allocate_bilinear_interp(lis_nc_working*lis_nr_working)
          call bilinear_interp_input(gridDesci,lis%d%gridDesc,&
               lis_nc_working*lis_nr_working)
       elseif(lis%f%interp.eq.2) then
          call allocate_bilinear_interp(lis_nc_working*lis_nr_working)
          call bilinear_interp_input(gridDesci,lis%d%gridDesc,&
               lis_nc_working*lis_nr_working)
          call allocate_conserv_interp(lis_nc_working*lis_nr_working)
```

```
          call conserv_interp_input(gridDesci,lis%d%gridDesc,&
               lis_nc_working*lis_nr_working)
       endif
#if ( ! defined OPENDAP )
    endif
#endif
```

1.1.5 get (Source File: baseforcing_module.F90)

Retrieves and interpolates model forcing

INTERFACE:

```
subroutine get()
```

USES:

```
use lisdrv_module, only: getforcing
```

CONTENTS:

```
call getf(getforcing())
```

1.1.6 time_interp (Source File: baseforcing_module.F90)

Computes temporal interpolation

INTERFACE:

```
subroutine time_interp()
```

USES:

```
use lisdrv_module,only :getforcing
```

CONTENTS:

```
call timeinterp(getforcing())
```

CONTENTS:

```
nmif = getnmif()
if(masterproc) then
   allocate(glbdata1(nmif,lis%d%glbngrid))
   allocate(glbdata2(nmif,lis%d%glbngrid))
else
   allocate(glbdata1(nmif,gdi(iam)))
   allocate(glbdata2(nmif,gdi(iam)))
endif
```

1.1.7 ld_getbaseforcing (Source File: baseforcing_module.F90)

Retrieves model forcing and invokes spatial and interpolation routines

INTERFACE:

```
subroutine ld_getbaseforcing()
```

USES:

```
 use lisdrv_module, only: lis, grid
 use grid_spmdMod, only : gdisp,gdi
```

CONTENTS:

```
#if ( defined OPENDAP )
    call get()
#else
    if ( masterproc ) then
       call get()
    endif
#if (defined SPMD)
    call MPI_BCAST(lis%f%findtime1,1,MPI_INTEGER,0, &
         MPI_COMM_WORLD, ier)
    call MPI_BCAST(lis%f%findtime2,1,MPI_INTEGER,0, &
         MPI_COMM_WORLD, ier)
    call MPI_BCAST(lis%f%gridchange,1,MPI_INTEGER,0, &
         MPI_COMM_WORLD, ier)
#endif

    if(lis%f%findtime1 ==1 .or.  &
         lis%f%findtime2==1) then
       if(npes > 1) then
          call scatter_data()
       endif
    endif
    if ( lis%f%gridchange == 1 ) then ! grid HAS changed
       if(npes>1) then
          call scatter_elev()
       endif
       lis%f%gridchange = 0
    endif
#endif
    call time_interp()
```

1.1.8 scatter_data (Source File: baseforcing_module.F90)

Distributes the forcing data on compute nodes

INTERFACE:

```
subroutine scatter_data()
```

USES:

```
use lisdrv_module, only : lis
```

1.1.9 scatter_elev (Source File: baseforcing_module.F90)

Distributes the elevation difference correction on compute nodes

INTERFACE:

```
subroutine scatter_elev()
```

USES:

```
use lisdrv_module, only : tile
use driverpardef_module
use tile_spmdMod
```

1.1.10 check_error (Source File: check_error.F90)

Error check; Program exits in case of error

INTERFACE:

```
subroutine check_error(ierr,msg,iam)
```

CONTENTS:

```
if ( ierr /= 0 ) then
   print*,'ERR: ',msg,' Stopping.',' (',iam,')'
   call endrun
endif
```

1.1.11 define_gds.F90 (Source File: define_gds.F90)

Assigns a grid definition section (GDS) array appropriate to the global resolution used.

REVISION HISTORY:

```
20 Jul 2001: Urszula Jambor; Initial code
12 Feb 2002: Urszula Jambor; Added latmax variable assignment
06 Mar 2002: Urszula Jambor; Added 1 & 1/2 degree resolution GDS arrays
24 Feb 2004: James Geiger; Stripped routine down so it only updates
                           values needed by the GrADS-DODS server
```

INTERFACE:

```
subroutine define_gds ( lis )
```

USES:

```
  use lis_module      ! LDAS non-model-specific 1-D variables
#if ( defined OPENDAP )
  use opendap_module, only : parm_nc, parm_nr,         &
                             output_slat, output_nlat, &
                             output_wlon, output_elon, &
                             ciam
  implicit none
```

ARGUMENTS:

```
  type (lisdec):: lis
```

CONTENTS:

```
!-----------------------------------------------------------------------
!      kgds(1) = 4 !Input grid type (4=Gaussian)
!      kgds(2) = 128 !Number of points on a lat circle
!      kgds(3) = 64 !Number of points on a meridian
!      kgds(4) = -87864 !Latitude of origin x1000
!      kgds(5) = 0 !Longitude of origin x1000
!      kgds(6) = 128 !8 bits (1 byte) related to resolution
! !(recall that 10000000 = 128), Table 7
!      kgds(7) = 87864 !Latitude of extreme point x1000
!      kgds(8) = -2812 !Longitude of extreme point x1000
!      kgds(9) = 2812 !N/S direction increment x1000
!      kgds(10) = 32 !(Gaussian) # lat circles pole-equator
!      kgds(11) = 64 !8 bit scanning mode flag (Table 8)
!-----------------------------------------------------------------------
     lis%d%gridDesc(2) = parm_nc
     lis%d%gridDesc(3) = parm_nr
     lis%d%gridDesc(4) = output_slat
     lis%d%gridDesc(5) = output_wlon
     lis%d%gridDesc(7) = output_nlat
     lis%d%gridDesc(8) = output_elon

     print*, 'DBG: define_gds -- lis%d%gridDesc(2)', lis%d%gridDesc(2), &
             ' ( ',ciam,' )'
     print*, 'DBG: define_gds -- lis%d%gridDesc(3)', lis%d%gridDesc(3), &
             ' ( ',ciam,' )'
     print*, 'DBG: define_gds -- lis%d%gridDesc(4)', lis%d%gridDesc(4), &
             ' ( ',ciam,' )'
     print*, 'DBG: define_gds -- lis%d%gridDesc(5)', lis%d%gridDesc(5), &
             ' ( ',ciam,' )'
     print*, 'DBG: define_gds -- lis%d%gridDesc(7)', lis%d%gridDesc(7), &
             ' ( ',ciam,' )'
```

```
     print*, 'DBG: define_gds -- lis%d%gridDesc(8)', lis%d%gridDesc(8), &
             ' ( ',ciam,' )'
#endif
```

1.2 Fortran: Module Interface domain_module.F90 (Source File: domain_module.F90)

This module contains interfaces and subroutines that controls the incorporation of new domains

REVISION HISTORY:

```
17Feb04    Sujay Kumar  Initial Specification
```

1.2.1 forcing_init (Source File: domain_module.F90)

Sets up functions for defining domain initializations

INTERFACE:

```
subroutine define_domains()
```

USES:

```
use domain_pluginMod, only : domain_plugin
use landcover_pluginMod, only : landcover_plugin
use elevdiff_pluginMod, only : elevdiff_plugin
```

1.2.2 (Source File: domain_module.F90)

Makes the domain

INTERFACE:

```
subroutine domain_init(domain)
```

USES:

```
     integer, intent(in) :: domain
```

CONTENTS:

```
    call makedomain(domain)
  end subroutine domain_init
```

1.2.3 read_domain (Source File: domain_module.F90)

calls the appropriate domain

INTERFACE:

```
subroutine read_domain(domain)
```

USES:

```
integer, intent(in) :: domain
```

CONTENTS:

```
call readinput(domain)
end subroutine read_domain
```

end module domain_module
!————————————————————————————————————- ! NASA Goddard Space Flight Center Land Information System (LIS) V3.0 ! Released May 2004 ! ! See SOFTWARE DISTRIBUTION POLICY for software distribution policies ! ! The LIS source code and documentation are in the public domain, ! available without fee for educational, research, non-commercial and ! commercial purposes. Users may distribute the binary or source ! code to third parties provided this statement appears on all copies and ! that no charge is made for such copies. ! ! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE ! SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED AS IS WITHOUT EXPRESS OR ! IMPLIED WARRANTY. NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE ! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE. ! ! See COPYRIGHT.TXT for copyright details. ! !————————————————————————————————————- #include "misc.h"

1.3 Fortran: Module Interface driverpardef_module.F90 (Source File: driverpardef_module.F90)

This module contains routines that defines MPI derived data types for LIS driver specific variables

REVISION HISTORY:

```
06 Oct 2003; Sujay Kumar  Initial Specification
```

INTERFACE:

```
module driverpardef_module
```

USES:

```
use tile_module
use lis_module
use grid_module
use spmdMod

implicit none
```

ARGUMENTS:

```
#if (defined SPMD)
  integer:: MPI_TILE_STRUCT  !MPI derived type for tile$_-$module
  integer:: MPI_GRID_STRUCT  !MPI derived type for grid$_-$module
  integer:: MPI_LD_STRUCT    !MPI derived type for lisdomain
  integer:: MPI_LF_STRUCT    !MPI derived type for lisforcing
  integer:: MPI_LP_STRUCT    !MPI derived type for lisparameters
  integer:: MPI_LT_STRUCT    !MPI derived type for listime
  integer:: MPI_LO_STRUCT    !MPI derived type for lisoutput
  integer:: MPI_LA_STRUCT    !MPI derived type for lisassimil
```

1.3.1 def_driverpar_structs (Source File: driverpardef_module.F90)

! ! ! Routine that defines MPI derived data types ! INTERFACE:

```
subroutine def_driverpar_structs()
```

1.4 Fortran: Module Interface drv_output_mod (Source File: drv_output_mod.F90)

Module containing output methods for different file formats

REVISION HISTORY:

```
10Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
module drv_output_mod
```

1.4.1 drv_writevar_bin (Source File: drv_output_mod.F90)

! ! Write the variable to an output file in different formats ! REVISION HISTORY:

```
!  10Feb2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine drv_writevar_bin(ftn, var)

  use lisdrv_module, only : lis,tile,gindex

  integer, intent(in) :: ftn
  real, intent(in)    :: var(lis%d%glbnch)
  real, allocatable :: ttmp(:)
```

```
    real, allocatable :: gtmp(:,:)

    if(lis%o%wtil.eq.1) then
       allocate(ttmp(lis%d%glbnch))
       call t2gr(var,ttmp,lis%d%glbngrid,lis%d%glbnch,tile)
       write(ftn) ttmp
       deallocate(ttmp)
    else
       allocate(gtmp(lis%d%lnc,lis%d%lnr))
       call tile2grid(var,gtmp,lis%d%glbnch,lis%d%lnc,&
            lis%d%lnr,lis%d%udef,tile)
       write(ftn) gtmp
       deallocate(gtmp)
    endif
  end subroutine drv_writevar_bin

  subroutine drv_writevar_netcdf(ftn,var,dim1,varid)

    use lisdrv_module, only : lis,tile,gindex
    integer,intent(in) :: ftn,dim1
    real, intent(in)   :: var(lis%d%glbnch)
#if ( defined USE_NETCDF )
    integer :: varid, iret
    iret =nf90_put_var(ftn, varid, var, (/1,dim1/),(/lis%d%glbnch,1/))
#endif
  end subroutine drv_writevar_netcdf

  subroutine drv_writevar_netcdf3d(ftn,var,dim1,soill,varid)

    use lisdrv_module, only : lis,tile,gindex

    integer,intent(in) :: ftn,dim1,soill
     real, intent(in)   :: var(dim1,lis%d%glbnch)
    real, intent(in)   :: var(lis%d%glbnch)
#if ( defined USE_NETCDF )
     real, allocatable :: ttmp(:)
    integer :: varid, iret
     allocate(ttmp(lis%d%glbnch))
     call t2gr(var,ttmp,lis%d%glbngrid,lis%d%glbnch,tile)
    iret =nf90_put_var(ftn, varid, var, (/1,dim1,soill/),(/lis%d%glbnch,1,1/))
     call drv_writenf3(ftn,varid,1,var,dim1,lis%d%glbnch)
     deallocate(ttmp)
#endif
  end subroutine drv_writevar_netcdf3d


  subroutine drv_writevar_grib(ftn, var, kpds, lismask,writeint,&
       today, yesterday)
```

```
  use lisdrv_module, only : lis,tile,gindex

  integer, intent(in)      :: ftn
  integer, intent(in)      :: kpds(25)
  character*8,intent(in)   :: today, yesterday
  real, intent(in)         :: writeint
   logical*1, intent(in)    :: lismask(lis%d%lnc,lis%d%lnr)
  logical*1                :: lismask(lis%d%lnc,lis%d%lnr)
  logical*1                :: lismask1(lis%d%lnc*lis%d%lnr)
  real, intent(in)         :: var(lis%d%glbnch)
  integer                  :: kpds_t(200),gridDesc(200)
  integer                  :: ngrid
  integer                  :: c,r,count
  real, allocatable        :: ttmp(:)
  real, allocatable        :: gtmp(:,:)

  gridDesc = 0
  gridDesc(1) = nint(lis%d%gridDesc(1))
  gridDesc(2) = nint(lis%d%gridDesc(2))
  gridDesc(3) = nint(lis%d%gridDesc(3))
  gridDesc(4) = nint(lis%d%gridDesc(4)*1000)
  gridDesc(5) = nint(lis%d%gridDesc(5)*1000)
  gridDesc(6) = nint(lis%d%gridDesc(6))
  gridDesc(7) = nint(lis%d%gridDesc(7)*1000)
  gridDesc(8) = nint(lis%d%gridDesc(8)*1000)
  gridDesc(9) = nint(lis%d%gridDesc(9)*1000)
  gridDesc(10) = nint(lis%d%gridDesc(10)*1000)
  gridDesc(11) = nint(lis%d%gridDesc(11))
  gridDesc(20) = nint(lis%d%gridDesc(20))

  kpds_t= 0
  do i=1,25
     kpds_t(i) = kpds(i)
  enddo

  ngrid = lis%d%lnc*lis%d%lnr
  allocate(ttmp(ngrid))
  allocate(gtmp(lis%d%lnc,lis%d%lnr))
  ttmp = 0
  gtmp = 0
  count = 1

Convert 1D tiled input array, var, to 2D output grid, gtmp
  call tile2grid(var,gtmp,lis%d%glbnch,lis%d%lnc,&
       lis%d%lnr,lis%d%udef,tile)

Prepare LIS mask for writing of ttmp (or var) to putgb
   lismask = .false.  ! 2D mask
```

```
     lismask1 = .false.

     do r=1,lis%d%lnr
        do c=1,lis%d%lnc
           ttmp(count)=gtmp(c,r)
            write(45,*) c,r,count,ttmp,gtmp
           if(gindex(c,r).ne.-1) then
               ttmp(count) = var(gindex(c,r))
              lismask1(count) = .true.
           endif
           count = count+1
        enddo
     enddo

     call makepdsn(today, yesterday, kpds_t, lis%t%hr,writeint)
     call putgb(ftn,ngrid,kpds_t,gridDesc,lismask1,ttmp,iret)

     if (iret .ne. 0) then
        print*, 'putgb failed for hour = ',lis%t%hr,',','field =',ttmp
        call endrun
     end if

     deallocate(ttmp)
     deallocate(gtmp)

   end subroutine drv_writevar_grib


%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{tile2grid (Source File: drv\_output\_mod.F90)}


   !
   !  Transfer variables from tile space to grid space
   !
\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}   !  20 Oct 2003; Sujay Kumar, Initial Code
   !  23 Jun 2004; Sujar Kumar, Corrected handling of udef
   !  13 Aug 2004; James Geiger, Corrected indices for GDS-based running mode
```

INTERFACE:

```
   subroutine tile2grid(t,g,nch,nc,nr,udef,tile)
```

USES:

```
     use tile_module
```

```
     use lisdrv_module,      only : glbgindex, lis
     use lis_indices_module, only : lis_g2l_row_offset, &
                                    lis_g2l_col_offset, &
                                    lis_tnroffset
```

CONTENTS:

```
    integer             :: c,r,i
    g=0
    do r=1,nr
       do c=1,nc
          if(glbgindex(c,r).eq.-1) then
             g(c,r) = udef
!             print*,  glbgindex(c,r),c,r,udef
          endif
       enddo
    enddo
!for non-1km resolutions.
!    if(lis%d%domain.ne.8) then
!    if(lis%d%gridDesc(9) .ne.0.01) then
!       do i=1, nch
!          c=tile(i)%col
!          r=tile(i)%row! + lis_tnroffset
          !rindex = r - nint((lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
          !     /lis%d%gridDesc(9))
          !cindex = c - nint((lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
          !     /lis%d%gridDesc(10))
!          rindex = r - lis_g2l_row_offset
!          cindex = c - lis_g2l_col_offset
!          g(cindex,rindex) = g(cindex,rindex) + t(i)*tile(i)%fgrd
!       end do
!    else
       do i=1, nch
          c=tile(i)%col
          r=tile(i)%row
          g(c,r) = g(c,r) + t(i)*tile(i)%fgrd
       end do
!    endif
!    do r=1,nr
!       do c=1,nc
!          g(c,r) = g(c,r) + t(glbgindex(c,r))*tile(glbgindex(c,r))%fgrd
!          print*, c,r,t(glbgindex(c,r)), tile(glbgindex(c,r))%fgrd, g(c,r)
!       enddo
!    enddo
    return
```

!

1.4.2 t2gr.F90 (Source File: drv_output_mod.F90)

! ! Aggregate variables for all tiles at each grid point ! REVISION HISTORY:

```
!  15 Oct 1999: Paul Houser; Initial Code
```

INTERFACE:

```
subroutine t2gr(t,g,ngrid, nch,tile)
```

USES:

```
 use tile_module
```

CONTENTS:

```
integer       :: c,r,i
g=0.0
do i=1,nch
   g(tile(i)%index)=g(tile(i)%index)+t(i)*tile(i)%fgrd
enddo
return
```

1.4.3 elevadjust.F90 (Source File: elevadjust.F90)

Corrects Temperature, Pressure, Humidity and Longwave Radiation values for differences in elevation between EDAS and LDAS grids.

REVISION HISTORY:

```
 11 Apr 2000: Brian Cosgrove; Initial Code
 12 May 2000: Brian Cosgrove; Corrected for zero humidities
 09 Aug 2000: Brian Cosgrove; Corrected program so that
              it only performs calculations if both
              the elevation difference file and the forcing
              data file (use temperature data as check for all
              fields) contain defined values
 25 Jan 2001: Matt Rodell; Compute number of input and output
grid points, use to allocate local arrays
 27 Feb 2001: Brian Cosgrove; Added statement to check for use of
              catchment data so that correct elevation correction
              files is used
 15 Mar 2001: Jon Gottschalck; if-then to handle negative vapor
pressures in long wave correction
 15 Mar 2001: Matt Rodell; merge NLDAS and GLDAS versions
 14 Nov 2003: Sujay Kumar; Adopted in LIS
```

INTERFACE:

```
subroutine elevadjust(t,f,fforce,force_tmp,force_hum,force_lwd, &
     force_prs)
```

USES:

```
use lisdrv_module, only: grid
implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: f, t
```

OUTPUT PARAMETERS:

```
real, intent(inout) :: fforce,force_tmp,force_hum,&
     force_lwd,force_prs
```

CONTENTS:

```
grav = 9.81
rdry = 287.
lapse = -0.0065
tcforce=force_tmp+(lapse*grid(t)%elev)
tbar=(force_tmp+tcforce)/2.
pcforce=force_prs/(exp((grav*grid(t)%elev)/(rdry*tbar)))
if (force_hum .eq. 0) force_hum=1e-08
ee=(force_hum*force_prs)/0.622
esat=611.2*(exp((17.67*(force_tmp-273.15))/&
     ((force_tmp-273.15)+243.5)))
qsat=(0.622*esat)/(force_prs-(0.378*esat))
rh=(force_hum/qsat)*100.
fesat=611.2*(exp((17.67*(tcforce-273.15))/ &
     ((tcforce-273.15)+243.5)))
fqsat=(0.622*fesat)/(pcforce-(0.378*fesat))
hcforce=(rh*fqsat)/100.
fe=(hcforce*pcforce)/0.622
mee=ee/100.
mfe=fe/100.
!-----------------------------------------------------------------------
! correct for negative vapor pressure at very low temperatures at
! high latitudes
!-----------------------------------------------------------------------
if (mee .le. 0) mee = 1e-08
if (mfe .le. 0) mfe = 1e-08
emiss  =1.08*(1-exp(-mee**(force_tmp/bb)))
femiss =1.08*(1-exp(-mfe**(tcforce/bb)))
ratio=(femiss*(tcforce**4))/(emiss*(force_tmp**4))
lcforce=force_lwd*ratio

select case (f)
case(1)
```

```
     fforce=tcforce
  case(2)
     fforce=hcforce
  case(4)
     fforce=lcforce
  case(7)
     fforce=pcforce
  case default
     print*, "not a valid forcing type for elevation adjustment"
     call endrun
  end select
  return
```

1.4.4 endrun.F90 (Source File: endrun.F90)

Routine to be called to terminate the program. This routines flushes the output streams and aborts the mpi processes.

REVISION HISTORY:

```
    14Nov02    Sujay Kumar  Initial Specification
 #include <misc.h>
```

INTERFACE:

```
 subroutine endrun
```

USES:

```
 #if (defined SPMD)
   use mpishorthand, only: MPI_COMM_WORLD
 #endif
```

CONTENTS:

```
  write(6,*)'endrun is being called'
  call lis_flush( 6 )   ! Flush all output to standard output
#if (defined SPMD)
  call mpi_abort (MPI_COMM_WORLD, 1)
#else
  call abort
#endif
```

1.5 Fortran: Module Interface filename_mod.F90 (Source File: filename_mod.F90)

Generates filenames for a number of routines.

REVISION HISTORY:

```
13 Mar 2004   Sujay Kumar  Initial Specification
```

1.5.1 avhrr_file_1km (Source File: filename_mod.F90)

Generates the 1km filenames for AVHHR LAI data.

INTERFACE:

```
subroutine avhrr_laifile_1km (name9,name10,name11,name12,&
     avhrrdir, cyr1, cyr2, cmo1, cmo2 )

  implicit none
```

ARGUMENTS:

```
  character(len=80), intent(out) :: name9,  name10, name11, name12
  character(len=40), intent(in)  :: avhrrdir
  character(len=4), intent(in)   :: cyr1, cyr2
  character(len=2), intent(in)   :: cmo1, cmo2
```

1.5.2 avhrr_file_1km (Source File: filename_mod.F90)

Generates the 1km filenames for AVHHR LAI data.

INTERFACE:

```
subroutine avhrr_saifile_1km (name13,&
     name14,name15,name16, &
     avhrrdir, cyr1, cyr2, cmo1, cmo2 )

  implicit none
```

ARGUMENTS:

```
  character(len=80), intent(out) :: name13, name14, name15, name16
  character(len=40), intent(in)  :: avhrrdir
  character(len=4), intent(in)   :: cyr1, cyr2
  character(len=2), intent(in)   :: cmo1, cmo2
```

1.5.3 modis_file_1km (Source File: filename_mod.F90)

Generates the 1km filenames for MODIS LAI data.

INTERFACE:

```
subroutine modis_file_1km (name9,name10,name11,name12,name13,&
     name14,name15,name16, &
     modisdir, cyr1, cyr2, cmo1, cmo2 )

  implicit none
```

ARGUMENTS:

```
  character(len=80), intent(out) :: name9,  name10, name11, name12
  character(len=80), intent(out) :: name13, name14, name15, name16
  character(len=40), intent(in)  :: modisdir
  character(len=4), intent(in)   :: cyr1, cyr2
  character(len=2), intent(in)   :: cmo1, cmo2
```

1.5.4 avhrr_laifilename (Source File: filename_mod.F90)

!This subroutine puts together AVHRR file name ! INTERFACE:

```
subroutine avhrr_laifilename ( &
     name9,name10,name11,name12, &
     avhrrdir, cyr1, cyr2, cmo1, cmo2 )

  implicit none
```

ARGUMENTS:

```
  character(len=80), intent(out) :: name9,  name10, name11, name12
  character(len=40), intent(in)  :: avhrrdir
  character(len=4),  intent(in)  :: cyr1, cyr2
  character(len=2),  intent(in)  :: cmo1, cmo2
```

CONTENTS:

```
write(unit=temp1,fmt='(a40)') avhrrdir
write(unit=temp2,fmt='(a40)') avhrrdir
write(unit=temp3,fmt='(a40)') avhrrdir
write(unit=temp4,fmt='(a40)') avhrrdir

read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)
```

```
write(unit=temp1,fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2,fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4,fmt='(7a1)') fdir_4

do i = 1, 7
   if ( fdir(i) == ' ' ) fdir(i) = '0'
   if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
   if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
   if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp2, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp3, fmt='(a15)') '_AVHRRLAI_0.125'
write(unit=temp4, fmt='(a15)') '_AVHRRLAI_0.125'
read (unit=temp1, fmt='(80a1)') (fsubsn(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_2(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_3(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_4(i), i=1,15)

c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
     (fsubsn(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
     (fsubsn_2(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
     (fsubsn_3(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
     (fsubsn_4(i), i=1,15)
read(unit=temp1, fmt='(a80)') name9
read(unit=temp2, fmt='(a80)') name10
read(unit=temp3, fmt='(a80)') name11
read(unit=temp4, fmt='(a80)') name12

print*, 'lai1 : rt ',name9
print*, 'lai2 : rt ',name10
```

```
print*, 'lai1 : clim ',name11
print*, 'lai2 : clim ',name12


return
```

1.5.5 avhrr_laifilename (Source File: filename_mod.F90)

!This subroutine puts together AVHRR file name ! INTERFACE:

```
subroutine avhrr_saifilename ( &
     name13,name14,name15,name16, &
     avhrrdir, cyr1, cyr2, cmo1, cmo2 )

 implicit none
```

ARGUMENTS:

```
 character(len=80), intent(out) :: name13, name14, name15, name16
 character(len=40), intent(in)  :: avhrrdir
 character(len=4),  intent(in)  :: cyr1, cyr2
 character(len=2),  intent(in)  :: cmo1, cmo2
```

CONTENTS:

```
write(unit=temp1,fmt='(a40)') avhrrdir
write(unit=temp2,fmt='(a40)') avhrrdir
write(unit=temp3,fmt='(a40)') avhrrdir
write(unit=temp4,fmt='(a40)') avhrrdir

read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)

write(unit=temp1,fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2,fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4,fmt='(7a1)') fdir_4

do i = 1, 7
   if ( fdir(i) == ' ' ) fdir(i) = '0'
   if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
```

```
      if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
      if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
   enddo

   write(unit=temp1, fmt='(a15)') '_AVHRRSAI_0.125'
   write(unit=temp2, fmt='(a15)') '_AVHRRSAI_0.125'
   write(unit=temp3, fmt='(a15)') '_AVHRRSAI_0.125'
   write(unit=temp4, fmt='(a15)') '_AVHRRSAI_0.125'
   read (unit=temp1, fmt='(80a1)') (fsubsn_5(i), i=1,15)
   read (unit=temp2, fmt='(80a1)') (fsubsn_6(i), i=1,15)
   read (unit=temp3, fmt='(80a1)') (fsubsn_7(i), i=1,15)
   read (unit=temp4, fmt='(80a1)') (fsubsn_8(i), i=1,15)

   c = 0
   do i = 1, 80
      if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
      if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
      if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
      if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
   end do

   write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
        (fsubsn_5(i), i=1,15)
   write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
        (fsubsn_6(i), i=1,15)
   write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
        (fsubsn_7(i), i=1,15)
   write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
        (fsubsn_8(i), i=1,15)
   read(unit=temp1, fmt='(a80)') name13
   read(unit=temp2, fmt='(a80)') name14
   read(unit=temp3, fmt='(a80)') name15
   read(unit=temp4, fmt='(a80)') name16

   print*, 'sai1: real ',name13
   print*, 'sai2: real ',name14
   print*, 'sai1: clim ',name15
   print*, 'sai2: clim ',name16

   return
```

!

1.5.6 avhrr_file_5km (Source File: filename_mod.F90)

!This subroutine puts together AVHRR file name ! INTERFACE:

```
subroutine avhrr_laifile_5km ( &
     name9,name10,name11,name12, &
     avhrrdir, cyr1, cyr2, cmo1, cmo2 )
  implicit none

  character(len=40), intent(in) :: avhrrdir
  character(len=80), intent(out) :: name9,  name10, name11, name12
  character(len=4), intent(in)  :: cyr1, cyr2
  character(len=2), intent(in)  :: cmo1, cmo2
```

CONTENTS:

```
write(unit=temp1, fmt='(a40)') avhrrdir
write(unit=temp2, fmt='(a40)') avhrrdir
write(unit=temp3, fmt='(a40)') avhrrdir
write(unit=temp4, fmt='(a40)') avhrrdir
read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)

write(unit=temp1, fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2, fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4, fmt='(7a1)') fdir_4

do i = 1, 7
   if ( fdir(i) == ' ' ) fdir(i) = '0'
   if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
   if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
   if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a8)') '_5KM.bin'
write(unit=temp2, fmt='(a8)') '_5KM.bin'
write(unit=temp3, fmt='(a8)') '_5KM.bin'
write(unit=temp4, fmt='(a8)') '_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_2(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_3(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_4(i), i=1,15)

c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
```

```
      if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
      if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
      if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
   end do

   write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
        (fsubsn(i), i=1,15)
   write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
        (fsubsn_2(i), i=1,15)
   write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
        (fsubsn_3(i), i=1,15)
   write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
        (fsubsn_4(i), i=1,15)
   read(unit=temp1, fmt='(a80)') name9
   read(unit=temp2, fmt='(a80)') name10
   read(unit=temp3, fmt='(a80)') name11
   read(unit=temp4, fmt='(a80)') name12

   return
```

!

1.5.7 avhrr_file_5km (Source File: filename_mod.F90)

!This subroutine puts together AVHRR file name ! INTERFACE:

```
  subroutine avhrr_saifile_5km ( &
       name13,name14,name15,name16, &
       avhrrdir, cyr1, cyr2, cmo1, cmo2 )
    implicit none

    character(len=40), intent(in) :: avhrrdir
    character(len=80), intent(out) :: name13, name14, name15, name16
    character(len=4), intent(in)  :: cyr1, cyr2
    character(len=2), intent(in)  :: cmo1, cmo2
```

CONTENTS:

```
   write(unit=temp1, fmt='(a40)') avhrrdir
   write(unit=temp2, fmt='(a40)') avhrrdir
   write(unit=temp3, fmt='(a40)') avhrrdir
   write(unit=temp4, fmt='(a40)') avhrrdir
   read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
   read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
   read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
   read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)
```

```
write(unit=temp1, fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2, fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4, fmt='(7a1)') fdir_4

do i = 1, 7
   if ( fdir(i) == ' ' ) fdir(i) = '0'
   if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
   if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
   if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp2, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp3, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp4, fmt='(a12)') '_SAI_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn_5(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_6(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_7(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do
write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
     (fsubsn_5(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
     (fsubsn_6(i), i=1,15)
write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
     (fsubsn_7(i), i=1,15)
write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
     (fsubsn_8(i), i=1,15)
read(unit=temp1, fmt='(a80)') name13
read(unit=temp2, fmt='(a80)') name14
read(unit=temp3, fmt='(a80)') name15
read(unit=temp4, fmt='(a80)') name16

return
```

!

1.5.8 modis_file_2 (Source File: filename_mod.F90)

! This subroutine puts together MODIS file name ! INTERFACE:

```
subroutine modis_file_2 (name9,name10,name11,name12,&
     name13,name14,name15,name16, &
     modisdir, cyr1, cyr2, cmo1, cmo2 )
```

CONTENTS:

```
92  format (80a1)
93  format (a80)
94  format (i4, i2, i2, i2)
95  format (10a1)
96  format (a40)
97  format (a9)
67  format (a15)
98  format (a1, a4, a2)
66  format (a5,a2)
99  format (7a1)

    write(90, 96, rec=1) modisdir
    write(91, 96, rec=1) modisdir
    write(92, 96, rec=1) modisdir
    write(93, 96, rec=1) modisdir
    read(90, 92, rec=1) (fbase(i), i=1,80)
    read(91, 92, rec=1) (fbase_2(i), i=1,80)
    read(92, 92, rec=1) (fbase_3(i), i=1,80)
    read(93, 92, rec=1) (fbase_4(i), i=1,80)

    write(90, 98, rec=1) '/', cyr1, cmo1
    read(90, 99, rec=1) fdir
    write(91, 98, rec=1) '/', cyr2, cmo2
    read(91, 99, rec=1) fdir_2
    write(92, 66, rec=1) '/CLIM', cmo1
    read(92, 99, rec=1) fdir_3
    write(93, 66, rec=1) '/CLIM', cmo2
    read(93, 99, rec=1) fdir_4

    do i = 1, 7
       if ( fdir(i) == ' ' ) fdir(i) = '0'
       if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
       if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
       if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
    enddo
```

```
write(90, 67, rec=1) '_MODISLAI_0.125'
write(91, 67, rec=1) '_MODISLAI_0.125'
write(92, 67, rec=1) '_MODISLAI_0.125'
write(93, 67, rec=1) '_MODISLAI_0.125'
read (90, 92, rec=1) (fsubsn(i), i=1,15)
read (91, 92, rec=1) (fsubsn_2(i), i=1,15)
read (92, 92, rec=1) (fsubsn_3(i), i=1,15)
read (93, 92, rec=1) (fsubsn_4(i), i=1,15)
write(90, 67, rec=1) '_MODISSAI_0.125'
write(91, 67, rec=1) '_MODISSAI_0.125'
write(92, 67, rec=1) '_MODISSAI_0.125'
write(93, 67, rec=1) '_MODISSAI_0.125'
read (90, 92, rec=1) (fsubsn_5(i), i=1,15)
read (91, 92, rec=1) (fsubsn_6(i), i=1,15)
read (92, 92, rec=1) (fsubsn_7(i), i=1,15)
read (93, 92, rec=1) (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
     (fsubsn(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
     (fsubsn_2(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
     (fsubsn_3(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
     (fsubsn_4(i), i=1,15)
read(90, 93, rec=1) name9
read(91, 93, rec=1) name10
read(92, 93, rec=1) name11
read(93, 93, rec=1) name12

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,7),  &
     (fsubsn_5(i), i=1,15)
write(91, 92, rec=1) (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
     (fsubsn_6(i), i=1,15)
write(92, 92, rec=1) (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
     (fsubsn_7(i), i=1,15)
write(93, 92, rec=1) (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
     (fsubsn_8(i), i=1,15)
read(90, 93, rec=1) name13
read(91, 93, rec=1) name14
```

```
read(92, 93, rec=1) name15
read(93, 93, rec=1) name16

close(90)
close(91)
close(92)
close(93)

!  print*, 'n9 ',name9
!  print*, 'n10 ',name10
!  print*, 'n11 ',name11
!  print*, 'n12 ',name12
!  print*, 'n13 ',name13
!  print*, 'n14 ',name14
!  print*, 'n15 ',name15
!  print*, 'n16 ',name16
return
```

!

1.5.9 modis_file_5km (Source File: filename_mod.F90)

! This subroutine puts together MODIS file name ! INTERFACE:

```
subroutine modis_file_5km(name9,name10,name11,name12,name13,&
     name14,name15,name16, &
     modisdir, cyr1, cyr2, cmo1, cmo2 )
```

CONTENTS:

```
92  format (80a1)
93  format (a80)
94  format (i4, i2, i2, i2)
95  format (10a1)
96  format (a40)
97  format (a9)
67  format (a8)
68  format (a12)
98  format (a1, a4, a2)
66  format (a5,a2)
99  format (7a1)

    write(unit=temp1, fmt='(a40)') modisdir
    write(unit=temp2, fmt='(a40)') modisdir
    write(unit=temp3, fmt='(a40)') modisdir
    write(unit=temp4, fmt='(a40)') modisdir
    read(unit=temp1, fmt='(80a1)') (fbase(i), i=1,80)
```

```
read(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,80)
read(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,80)
read(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,80)

write(unit=temp1, fmt='(a1,a4,a2)') '/', cyr1, cmo1
read(unit=temp1, fmt='(7a1)') fdir
write(unit=temp2, fmt='(a1,a4,a2)') '/', cyr2, cmo2
read(unit=temp2, fmt='(7a1)') fdir_2
write(unit=temp3, fmt='(a5,a2)') '/CLIM', cmo1
read(unit=temp3, fmt='(7a1)') fdir_3
write(unit=temp4, fmt='(a5,a2)') '/CLIM', cmo2
read(unit=temp4, fmt='(7a1)') fdir_4

do i = 1, 7
   if ( fdir(i) == ' ' ) fdir(i) = '0'
   if ( fdir_2(i) == ' ' ) fdir_2(i) = '0'
   if ( fdir_3(i) == ' ' ) fdir_3(i) = '0'
   if ( fdir_4(i) == ' ' ) fdir_4(i) = '0'
enddo

write(unit=temp1, fmt='(a8)') '_5KM.bin'
write(unit=temp2, fmt='(a8)') '_5KM.bin'
write(unit=temp3, fmt='(a8)') '_5KM.bin'
write(unit=temp4, fmt='(a8)') '_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_2(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_3(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_4(i), i=1,15)
write(unit=temp1, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp2, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp3, fmt='(a12)') '_SAI_5KM.bin'
write(unit=temp4, fmt='(a12)') '_SAI_5KM.bin'
read (unit=temp1, fmt='(80a1)') (fsubsn_5(i), i=1,15)
read (unit=temp2, fmt='(80a1)') (fsubsn_6(i), i=1,15)
read (unit=temp3, fmt='(80a1)') (fsubsn_7(i), i=1,15)
read (unit=temp4, fmt='(80a1)') (fsubsn_8(i), i=1,15)

c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_2(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_3(i) == ' ') .and. (c == 0) ) c = i-1
   if ( (fbase_4(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
     (fsubsn(i), i=1,15)
write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
```

```
        (fsubsn_2(i), i=1,15)
    write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
        (fsubsn_3(i), i=1,15)
    write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
        (fsubsn_4(i), i=1,15)
    read(unit=temp1, fmt='(a80)') name9
    read(unit=temp2, fmt='(a80)') name10
    read(unit=temp3, fmt='(a80)') name11
    read(unit=temp4, fmt='(a80)') name12
    write(unit=temp1, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,7),  &
        (fsubsn_5(i), i=1,15)
    write(unit=temp2, fmt='(80a1)') (fbase_2(i), i=1,c), (fdir_2(i), i=1,7),  &
        (fsubsn_6(i), i=1,15)
    write(unit=temp3, fmt='(80a1)') (fbase_3(i), i=1,c), (fdir_3(i), i=1,7),  &
        (fsubsn_7(i), i=1,15)
    write(unit=temp4, fmt='(80a1)') (fbase_4(i), i=1,c), (fdir_4(i), i=1,7),  &
        (fsubsn_8(i), i=1,15)
    read(unit=temp1, fmt='(a80)') name13
    read(unit=temp2, fmt='(a80)') name14
    read(unit=temp3, fmt='(a80)') name15
    read(unit=temp4, fmt='(a80)') name16
    print*, 'n11',name11
    print*, 'n15',name15
    return
  end subroutine modis_file_5KM
```

1.5.10 avhrr_g_file (Source File: filename_mod.F90)

! ! This subroutine puts together AVHRR LAI file name ! INTERFACE:

```
   subroutine avhrr_g_file (name9,name10,name11,name12,name13,name14, &
        name15,name16,avhrrdir,cyr1,cyr2,cmo1,cmo2 )

     implicit none
```

ARGUMENTS:

```
     character(len=40), intent(in)  :: avhrrdir
     character(len=80), intent(out) :: name9,  name10, name11, name12
     character(len=80), intent(out) :: name13, name14, name15, name16
     character(len=4), intent(in)   :: cyr1, cyr2
     character(len=2), intent(in)   :: cmo1, cmo2
```

1.5.11 modis_g_file (Source File: filename_mod.F90)

! ! This subroutine puts together MODIS LAI file name ! INTERFACE:

```
subroutine modis_g_file (name9,name10,name11,name12,name13,name14, &
     name15,name16,modisdir,cyr1,cyr2,cmo1,cmo2 )

  implicit none
```

ARGUMENTS:

```
  character(len=40), intent(in)  :: modisdir
  character(len=80), intent(out) :: name9,  name10, name11, name12
  character(len=80), intent(out) :: name13, name14, name15, name16
  character(len=4), intent(in)   :: cyr1, cyr2
  character(len=2), intent(in)   :: cmo1, cmo2
```

1.6 Fortran: Module Interface grid_module.F90 (Source File: grid_module.F90)

LIS non-model-specific grid variables only.
FORCING() ARRAY:
1. T 2m Temperature interpolated to 2 metres [K]
2. q 2m Instantaneous specific humidity interpolated to 2 metres[kg/kg]
3. radswg Downward shortwave flux at the ground [W/m^2]
4. lwgdwn Downward longwave radiation at the ground [W/m^2]
5. u 10m Instantaneous zonal wind interpolated to 10 metres [m/s]
6. v 10m Instantaneous meridional wind interpolated to 10 metres[m/s]
7. ps Instantaneous Surface Pressure [Pa]
8. preacc Total precipitation [mm/s]
9. precon Convective precipatation [mm/s]
10. albedo Surface albedo (0-1)

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial code
11 Apr 2000: Brian Cosgrove; Added Forcing Mask variables
23 Feb 2001: Urszula Jambor; Added GEOS & GDAS forcing variables
27 Feb 2001: Brian Cosgrove; Added Catchment forcing data variables
23 Mar 2001: Jon Radakovich; Added variables for PSAS assimilation
04 Sep 2001: Brian Cosgrove; Added variabes for humidity, precip,par
             brightness temp,precip mask, removed awips2lis and
             pinker2lis variables, GRIB interp. package used now
15 Oct 2001: Jesse Meng; Revised doc block with forcing array definition
15 Oct 2001: Jesse Meng; Added oblwdata1 and oblwdata2
14 Nov 2002: Sujay Kumar; Optimized version of grid_module
```

INTERFACE:

```
module grid_module
  implicit none
  public griddec
```

ARGUMENTS:

```
  type griddec
     real    :: lat            !latitude of grid point
     real    :: lon            !longitude of grid point
     real    :: elev
     real    :: forcing(10)    !interpolated LIS forcing array
     real    :: fgrd(13)  !fraction of vegetation class in grid
  end type griddec
```

1.7 Fortran: Module Interface grid_spmdMod.F90 (Source File: grid_spmdMod.F90)

This module computes domain decomposition on the grid domain

REVISION HISTORY:

```
  14 Nov 2002 Sujay Kumar Initial Specification
```

INTERFACE:

```
module grid_spmdMod
```

USES:

```
  use spmdMod
```

ARGUMENTS:

```
  integer, allocatable :: gdi(:), gdisp(:)
  integer, allocatable :: g2di(:), g2disp(:)
```

1.7.1 allocate_gdd (Source File: grid_spmdMod.F90)

Allocates memory for arrays that contain domain decomposition information

INTERFACE:

```
  subroutine allocate_gdd()
```

DESCRIPTION:

Allocates memory for arrays that contain domain decomposition information CONTENTS:

```
   allocate(gdi(0:npes-1))
   allocate(gdisp(0:npes-1))
   allocate(g2di(0:npes-1))
   allocate(g2disp(0:npes-1))
```

1.7.2 grid_spmd_init (Source File: grid_spmdMod.F90)

Computes domain decomposition based on the number of processors

INTERFACE:

```
subroutine grid_spmd_init(tile,nch,nmif,ngrid)
```

USES:

```
  use tile_module
  use tile_spmdMod, only : displs
!INPUT ARGUMENTS:
  integer, intent(in) :: nch, ngrid, nmif
!OUTPUT ARGUMENTS:
  type(tiledec)::tile(nch)
```

CONTENTS:

```
gdisp(0) = 0
do p=1, npes-1
   gdisp(p) = tile(displs(p))%index
enddo
do p = 0 , npes-2
   gdi(p) = gdisp(p+1)-gdisp(p)
enddo
gdi(npes-1) = ngrid - gdisp(npes-1)
do p=0, npes-1
   g2di(p) = gdi(p)*nmif
enddo
g2disp(0) = 0
do p=1, npes-1
   g2disp(p) = g2disp(p-1)+g2di(p-1)
enddo
```

1.8 Fortran: Module Interface gswp_module.F90 (Source File: gswp_module.F90)

This module contains useful routines that generates indices for reading the GSWP data

REVISION HISTORY:

```
24Feb04    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module gswp_module

contains
```

```
subroutine getgswp_monindex(yr,mo,index)
  integer, intent(out) :: index
  integer, intent(in) :: yr, mo
  integer :: k
  logical :: leap
  index = 0
  index = index + (yr-1982)*12 + mo
end subroutine getgswp_monindex

subroutine getgswp_timeindex(yr,mo,da,hr,index)

  implicit none
  integer, intent(in)  :: yr, mo, da, hr
  integer, intent(out) :: index
  integer :: ryr, rmo, rda, rhr, days1(12),days2(12),yrs(13)
  integer :: k
  data yrs /184,365,366,365,365,365,366,365,365,365,366,365,365/ !from 1983-94
  data days1 /31,28,31,30,31,30,31,31,30,31,30,31/
  data days2 /31,29,31,30,31,30,31,31,30,31,30,31/
  logical :: leap
  ryr = 1982
  rmo = 7
  rda = 1
  rhr = 0

  index = 0
  if(yr.gt.1982) then
     if((mod(yr,4).eq.0.and.mod(yr,100).ne.0) &     !correct for leap year
          .or.(mod(yr,400).eq.0))then             !correct for y2k
        leap = .true.
     else
        leap = .false.
     endif
     k = mo
     index = index + hr+ (da-1)*24
     do while(k.gt.1)
        if(leap) then
           index = index + days2(k-1) *24  !current year's
        else
           index = index + days1(k-1) *24  !current year's
        endif
        k = k-1
     enddo
     ! now add the missing years..
     k = yr-1982
     do while(k.ge.1)
        index = index + yrs(k)*24
        k = k-1
```

```
        enddo
        index = index/3
     else
        if(mo.gt.7) then
           k=mo
           index = index+hr+(da-1)*24
           do while(k .gt.7)
              index = index + days1(k-1)*24
              k = k-1
           end do
        else
           index = hr+(da-1)*24
        endif
        index = index/3
     endif
     index = index+1
     !convert times to 3 hour index

   end subroutine getgswp_timeindex

 end module gswp_module
\markboth{Left}{Source File: lisdrv.F90,  Date: Fri Feb 11 15:41:02 EST 2005
}


  ------------------------------------------------------------------------
   NASA Goddard Space Flight Center Land Information System (LIS) V3.0
   Released May 2004
   See SOFTWARE DISTRIBUTION POLICY for software distribution policies
   The LIS source code and documentation are in the public domain,
   available without fee for educational, research, non-commercial and
   commercial purposes.  Users may distribute the binary or source
   code to third parties provided this statement appears on all copies and
   that no charge is made for such copies.
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See COPYRIGHT.TXT for copyright details.
  ------------------------------------------------------------------------
 #include "misc.h"
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection{lisdrv.F90 - Main program for LIS (Source File: lisdrv.F90)}
```

```
    Main driver program for LIS. It initializes the boundary conditions,
    allocates memory for the required variables, sets up appropriated
    model parameters, performs the I/O for forcing and land models,
    in addition to calling the appropriate land model over different
    time steps and geographical domains.
    \subsection{Function calls for main routines: }

    The function calls are:
    \begin{description}
    \item[LIS\_domain\_init]
        Initializes the domain variables
    \item[LIS\_allocate\_memory]
        Allocates memory for modules, variables
    \item[LIS\_lsm\_init]
       Initializes land surface model run parameters
    \item[LIS\_baseforcing\_init]
       Initializes model forcing variables
    \item[LIS\_readrestart]
       Reads the restart files
    \item[LIS\_setuplsm]
       Completes initialization of the land surface model
    \item[LIS\_ticktime]
       Manages the advancement of time
    \item[LIS\_endofrun]
       Checks if the end of simulation is reached
    \item[LIS\_get\_base\_forcing]
       Reads, interpolates model forcing
    \item[LIS\_force2tile]
       Transfers grid forcing to model tiles.
    \item[LIS\_lsm\_main]
       Executes land surface model runs.
    \item[LIS\_lsm\_output]
       Writes land surface model output
    \item[LIS\_write\_restart]
       Writes restart files
    \end{description}

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}    14Nov02    Sujay Kumar  Initial Specification
```

USES:

```
   use precision
   use lisdrv_module
```

```
use lsm_module
use baseforcing_module
use obsprecipforcing_module
use obsradforcing_module
use spmdMod
```

CONTENTS:

```
call LIS_config
call LIS_domain_init
call LIS_lsm_init
call LIS_baseforcing_init
call LIS_obsprecipforcing_init
call LIS_obsradforcing_init
call LIS_setuplsm
call LIS_readrestart

do while (.NOT. LIS_endofrun())
   call LIS_ticktime
   call LIS_setDynlsm
   call LIS_get_base_forcing
   call LIS_get_obsprecip_forcing
   call LIS_get_obsrad_forcing
   call LIS_force2tile
   call LIS_lsm_main
   call LIS_lsm_output
   call LIS_writerestart
enddo
```

1.9 Fortran: Module Interface lisdrv_module.F90 (Source File: lisdrv_module.F90)

Main program for LIS This module contains interfaces and subroutines that control program execution.

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

USES:

```
use lis_module
use grid_module
use time_manager
use tile_spmdMod
use tile_module
```

1.9.1 ld_domain_init (Source File: lisdrv_module.F90)

Calls routines to read the card file, and initialize the time manager

INTERFACE:

```
subroutine LIS_config()
```

CONTENTS:

```
    call spmd_init()
#if ( ! defined OPENDAP )
    if ( masterproc ) then
#endif
       call readcard()
#if ( ! defined OPENDAP )
    endif
#endif
!-----------------------------------------------------------------------
! we have read in the time parameters. Use them to initialize ESMF
! time manager
!-----------------------------------------------------------------------
    if ( masterproc ) then
       call setup_timeMgr()
       lis%t%endtime = 0
    endif
```

1.9.2 setup_timeMgr (Source File: lisdrv_module.F90)

Initializes the ESMF time manager

INTERFACE:

```
subroutine setup_timeMgr()
```

CONTENTS:

```
    call timemgr_init(lis%t)
    print*, 'time manager initialized..'
```

1.9.3 LIS_ticktime (Source File: lisdrv_module.F90)

Uses the ESMF time manager to handle model timestepping.

INTERFACE:

```
subroutine LIS_ticktime()
```

USES:

```
 #if (defined SPMD)
     use driverpardef_module, only: MPI_LT_STRUCT
 #endif
```

CONTENTS:

```
    if(masterproc) then
       call advance_timestep(lis%t)
    endif
#if (defined SPMD)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0, &
         MPI_COMM_WORLD, ierr)
#endif
```

1.9.4 LIS_domain_init (Source File: lisdrv_module.F90)

Allocates memory for the domain variables, initializes MPI data structures, and computes domain decomposition

INTERFACE:

```
   subroutine LIS_domain_init
```

CONTENTS:

```
#if ( defined SPMD )
    call def_driverpar_structs()
#endif

#if ( defined OPENDAP )
#if ( defined SPMD )
    call MPI_BCAST(lis%d, 1, MPI_LD_STRUCT, 0, MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%f, 1, MPI_LF_STRUCT, 0, MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p, 1, MPI_LP_STRUCT, 0, MPI_COMM_WORLD, ierr)
#endif

    call define_domains()
    call read_domain(lis%d%domain)
#else
    if ( masterproc ) then
       call define_domains()
       call read_domain(lis%d%domain)
    endif
#endif
```

```
    if ( masterproc ) then
       nc = getnc()
       nr = getnr()
       maxt = getmaxt()
       call setnch(nc, nr, maxt)
    endif

#if 0
#if ( defined OPENDAP )
#if (defined SPMD)
    call MPI_BCAST(lis%d%ic, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%ir, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%lnc, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%lnr, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%nt, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%mina, 1, MPI_REAL, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%d%maxt, 1, MPI_INTEGER, 0, &
         MPI_COMM_WORLD, ierr)
!    call MPI_BCAST(lis%p%koster, 1, MPI_INTEGER, 0, &
!         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%mfile, 50, MPI_CHARACTER, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p%vfile, 50, MPI_CHARACTER, 0, &
         MPI_COMM_WORLD, ierr)
#endif
#endif
#endif

#if ( ! defined OPENDAP )
    if(masterproc)  then
#endif
       call domain_init(lis%d%domain)
#if ( ! defined OPENDAP )
    endif
#endif
#if (defined SPMD)
    !call def_driverpar_structs()
    call MPI_BCAST(lis%d, 1, MPI_LD_STRUCT, 0, &
          MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%f, 1, MPI_LF_STRUCT, 0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%p, 1, MPI_LP_STRUCT, 0, &
```

```
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%o, 1, MPI_LO_STRUCT,0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%a, 1, MPI_LA_STRUCT,0, &
         MPI_COMM_WORLD, ierr)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT,0, &
         MPI_COMM_WORLD, ierr)
#endif
#if ( ! defined OPENDAP )
    lis%d%ngrid = lis%d%glbngrid
    lis%d%nch   = lis%d%glbnch
#endif
    call allocate_tiledd
    if(masterproc) then
       call tile_spmd_init(tile, lis%d%glbnch,lis%f%nmif)
    endif
    call spread_tdds()
    if(.NOT.masterproc) then
       allocate(tile(di_array(iam)))
    endif
#if (defined SPMD)
    if(npes>1) then
       call MPI_SCATTERV(tile,di_array,displs, &
            MPI_TILE_STRUCT,tile,di_array(iam),MPI_TILE_STRUCT, &
            0,MPI_COMM_WORLD,ierr)
    endif
#endif
    call allocate_gdd
    if(masterproc) then
       call grid_spmd_init(tile,lis%d%glbnch, &
            lis%f%nmif, lis%d%glbngrid)
    endif
    call spread_gdds()
    if(.NOT.masterproc) then
       allocate(grid(gdi(iam)))
    endif
#if (defined SPMD)
    if(npes >1) then
       call MPI_SCATTERV(grid,gdi,gdisp, &
            MPI_GRID_STRUCT,grid,gdi(iam),MPI_GRID_STRUCT, &
            0,MPI_COMM_WORLD,ierr)
    endif
#endif
    call dist_gindex()
#if 0
!<kluge>
    if(masterproc) then
       call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
```

```
       call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
       call updatetime(lis%t)
    endif
!</kluge>
#endif
#if(defined SPMD)
    call MPI_BCAST(lis%t, 1, MPI_LT_STRUCT, 0,  &
         MPI_COMM_WORLD, ierr)
#endif
```

1.9.5 setnch (Source File: lisdrv_module.F90)

Computes an estimate of the total number of tiles

INTERFACE:

```
subroutine setnch(nc, nr, maxt)
```

ARGUMENTS:

```
integer::nc
integer::nr
integer::maxt
```

CONTENTS:

```
lis%d%glbnch = nc*nr*maxt !!may be too big
```

1.9.6 getdomain (Source File: lisdrv_module.F90)

Returns the domain resolution

INTERFACE:

```
function getdomain() result(d)
```

ARGUMENTS:

```
integer :: d
```

CONTENTS:

```
d = lis%d%domain
```

1.9.7 getlsm (Source File: lisdrv_module.F90)

Returns which land surface model is executed

INTERFACE:

```
function getlsm() result(lsmno)
```

ARGUMENTS:

```
integer :: lsmno
```

CONTENTS:

```
lsmno = lis%d%lsm
```

1.9.8 getnch (Source File: lisdrv_module.F90)

Returns the number of model tiles

INTERFACE:

```
function getnch() result(n)
```

ARGUMENTS:

```
integer :: n
```

CONTENTS:

```
n = lis%d%glbnch
```

1.9.9 getnc (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnc() result(ncol)
```

ARGUMENTS:

```
integer :: ncol
```

CONTENTS:

```
ncol = lis%d%lnc
```

1.9.10 getnr (Source File: lisdrv_module.F90)

Returns the number of columns

INTERFACE:

```
function getnr() result(nrow)
```

ARGUMENTS:

```
integer :: nrow
```

CONTENTS:

```
nrow = lis%d%lnr
```

1.9.11 gettileindex (Source File: lisdrv_module.F90)

Returns the index of a tile given lat lon

INTERFACE:

```
function gettileindex(lat,lon) result(k)
```

ARGUMENTS:

```
real :: lat, lon
integer :: k, t
```

CONTENTS:

```
k = -1
do t=1,lis%d%nch
   if((grid(tile(t)%index)%lat .eq. lat) .and. &
        (grid(tile(t)%index)%lon .eq. lon)) then
      k = t
   endif
enddo
```

1.9.12 getmaxt (Source File: lisdrv_module.F90)

Returns the maximum number of tiles

INTERFACE:

```
function getmaxt() result(maxt)
```

ARGUMENTS:

```
integer :: maxt
```

CONTENTS:

```
maxt = lis%d%maxt
```

1.9.13 getforcing (Source File: lisdrv_module.F90)

Returns the type of forcing used

INTERFACE:

```
function getforcing() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%force
```

1.9.14 getnmif (Source File: lisdrv_module.F90)

Returns the number of forcing variables for model initialization

INTERFACE:

```
function getnmif() result(f)
```

ARGUMENTS:

```
integer:: f
```

CONTENTS:

```
f = lis%f%nmif
```

1.9.15 LIS_endofrun (Source File: lisdrv_module.F90)

Returns if the end of simulation has reached

INTERFACE:

```
function LIS_endofrun() result(finish)
```

ARGUMENTS:

```
logical :: finish
integer :: ierr
```

CONTENTS:

```
    if(masterproc) then
       finish = is_last_step(lis%t)
    endif
#if (defined SPMD)
    call MPI_BCAST(finish, 1, MPI_LOGICAL, 0, &
         MPI_COMM_WORLD, ierr)
#endif
```

1.9.16 dist_gindex (Source File: lisdrv_module.F90)

Distributes the mask indices on compute nodes for a GDS-based execution

INTERFACE:

```
   subroutine dist_gindex
```

USES:

```
     use grid_spmdMod

     implicit none
```

ARGUMENTS:

```
     integer :: findex, lindex, nr_count, t
     integer :: ierr
     integer :: grid_offset, grid_lb, grid_ub
     integer :: i, j
 #if (defined SPMD)
     integer :: status(MPI_STATUS_SIZE)
 #endif
```

CONTENTS:

```
#if ( defined OPENDAP )
#if ( defined SPMD )
    if ( npes > 1 ) then
       if ( masterproc ) then
          do t = 1, npes-1
             findex = tile( gdisp(t) + 1 )%row
             lindex = tile( gdisp(t) + gdi(t) )%row
             nr_count = (lindex-findex+1)
             print*,'DBG: lisdrv_module -- ', &
                  't, findex,lindex,lis%d%gnc,nr_count', &
                  t, findex,lindex,lis%d%gnc,nr_count,' (', iam,')'
             call mpi_send(nr_count,1,MPI_INTEGER,t,t, &
                  MPI_COMM_WORLD,ierr)
             call mpi_send(glbgindex(:,findex:lindex), &
                  lis%d%gnc*nr_count, &
                  MPI_INTEGER,t,t,MPI_COMM_WORLD,ierr)
          enddo
       else
          call mpi_recv(nr_count,1,MPI_INTEGER,0,MPI_ANY_TAG, &
               MPI_COMM_WORLD,status,ierr)
          allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
          call check_error(ierr,'lisdrv_module', &
               'Error allocating gindex.',iam)
          call mpi_recv(gindex,lis%d%gnc*nr_count,MPI_INTEGER,0, &
               MPI_ANY_TAG, &
```

```
               MPI_COMM_WORLD,status,ierr)
       endif
    endif
#endif

    if ( masterproc ) then
       findex = tile( gdisp(0) + 1 )%row
       lindex = tile( gdisp(0) + gdi(0) )%row
       nr_count = (lindex-findex+1)
       print*,'DBG: lisdrv_module -- ', &
            'findex,lindex,lis%d%gnc,nr_count', &
            findex,lindex,lis%d%gnc,nr_count,' (', iam,')'
       allocate(gindex(lis%d%gnc,nr_count),stat=ierr)
       call check_error(ierr,'lisdrv_module', &
            'Error allocating gindex.',iam)
       print*,'DBG: lisdrv_module -- size(gindex),size(glbgindex)', &
            size(gindex(1:lis%d%gnc,1:nr_count)), &
            size(glbgindex(1:lis%d%gnc,findex:lindex))
       gindex = glbgindex(:,findex:lindex)
    endif

    grid_lb = gdisp(iam) + 1
    grid_ub = gdisp(iam) + gdi(iam)
    grid_offset = gdisp(iam)
    do j = 1, nr_count
       do i = 1, lis%d%gnc
          if ( gindex(i,j) /= -1 ) then
             if ( ( gindex(i,j) < grid_lb ) .or. &
                  ( gindex(i,j) > grid_ub ) ) then
                gindex(i,j) = - 1
             else
                gindex(i,j) = gindex(i,j) - grid_offset
             endif
          endif
       enddo
    enddo
#else
    if ( masterproc ) then
       allocate(gindex(lis%d%lnc,lis%d%lnr),stat=ierr)
       call check_error(ierr,'lisdrv_module', &
            'Error allocating gindex.',iam)
       gindex=glbgindex
    endif
#endif
```

1.10 Fortran: Module Interface lis_indices_module.F90 (Source File: lis_indices_module.F90)

This module contains generic indices for performing loops over rows and columns. Depending on how LIS is run (e.g., when using a GrADS-DODS server (GDS) or not), the number of rows and columns for a given domain may change. This module determines how LIS is run and which set of indices to use.

REVISION HISTORY:

```
08 Apr 2004  James Geiger; Initial Specification
```

1.10.1 lis_set_indices (Source File: lis_indices_module.F90)

This routine sets the generic indices – lis_nc_working, lis_nr_working, lis_tnroffset, lis_nc_data, lis_nr_data, lis_g2l_row_offset, and lis_g2l_col_offset – depending on how the LIS executable was compiled.

INTERFACE:

```
subroutine lis_set_indices()
```

1.10.2 lis_get_run_slat (Source File: lis_indices_module.F90)

This routine returns the value of the southern latitude for the running domain.

INTERFACE:

```
function lis_get_run_slat()
```

1.10.3 lis_get_run_wlon (Source File: lis_indices_module.F90)

This routine returns the value of the western longitude for the running domain.

INTERFACE:

```
function lis_get_run_wlon()
```

1.10.4 lis_get_run_nlat (Source File: lis_indices_module.F90)

This routine returns the value of the northern latitude for the running domain.

INTERFACE:

```
function lis_get_run_nlat()
```

1.10.5 lis_get_run_elon (Source File: lis_indices_module.F90)

This routine returns the value of the eastern longitude for the running domain.

INTERFACE:

```
function lis_get_run_elon()
```

1.10.6 lis_get_run_lat_res (Source File: lis_indices_module.F90)

This routine returns the value of the resolution (latitude) for the running domain.

INTERFACE:

```
function lis_get_run_lat_res()
```

1.10.7 lis_get_run_lon_res (Source File: lis_indices_module.F90)

This routine returns the value of the resolution (longitude) for the running domain.

INTERFACE:

```
function lis_get_run_lon_res()
```

1.10.8 lis_get_data_slat (Source File: lis_indices_module.F90)

This routine returns the value of the southern latitude for the parameter data domain.

INTERFACE:

```
function lis_get_data_slat()
```

1.10.9 lis_get_data_wlon (Source File: lis_indices_module.F90)

This routine returns the value of the western longitude for the parameter data domain.

INTERFACE:

```
function lis_get_data_wlon()
```

1.10.10 lis_get_data_nlat (Source File: lis_indices_module.F90)

This routine returns the value of the northern latitude for the parameter data domain.

INTERFACE:

```
function lis_get_data_nlat()
```

1.10.11 lis_get_data_elon (Source File: lis_indices_module.F90)

This routine returns the value of the eastern longitude for the parameter data domain.

INTERFACE:

```
function lis_get_data_elon()
```

1.10.12 lis_get_data_lat_res (Source File: lis_indices_module.F90)

This routine returns the value of the resolution (latitude) for the parameter data domain.

INTERFACE:

```
function lis_get_data_lat_res()
```

1.10.13 lis_get_data_lon_res (Source File: lis_indices_module.F90)

This routine returns the value of the resolution (longitude) for the parameter data domain.

INTERFACE:

```
function lis_get_data_lon_res()
```

1.10.14 lis_global_to_local_row_offset (Source File: lis_indices_module.F90)

This routine returns the offset needed to adjust a global row index value into its corresponding local row index value.

INTERFACE:

```
function lis_global_to_local_row_offset(offset)
```

1.10.15 lis_global_to_local_col_offset (Source File: lis_indices_module.F90)

This routine returns the offset needed to adjust a global column index value into its corresponding local column index value.

INTERFACE:

```
function lis_global_to_local_col_offset()
```

1.10.16 lis_log_msg.F90 (Source File: lis_log_msg.F90)

This routine formats a given message by prepending a time stamp and appending the process id number. This newly formatted message is then written to standard out.

REVISION HISTORY:

```
12 Mar 2004: James Geiger; Initial version
```

INTERFACE:

```
subroutine lis_log_msg(msg)
```

USES:

```
use spmdMod, only : iam

implicit none
```

INPUT PARAMETERS:

```
character(len=*), intent(in) :: msg
```

LOCAL VARIABLES:

```
character(len=8)  :: date
character(len=10) :: time
character(len=5)  :: zone
integer, dimension(8) :: values
```

CONTENTS:

```
call date_and_time(date,time,zone,values)

print*,date(1:4),'-',date(5:6),'-',date(7:8),'T',  &
       time(1:2),':',time(3:4),':',time(5:10),' ', &
       trim(msg),' (',iam,')'
```

1.10.17 lis_log_blocked_msg (Source File: lis_log_msg.F90)

This routine call lis_log_msg to print a time-stamped message, and then it waits at an mpi_barrier.

REVISION HISTORY:

```
16 Aug 2004: James Geiger; Initial version
```

INTERFACE:

```
subroutine lis_log_blocked_msg(msg)

#include "misc.h"
```

USES:

```
   use spmdMod

   implicit none
```

INPUT PARAMETERS:

```
   character(len=*), intent(in) :: msg

   integer :: ierr

   call lis_log_msg(msg)

#if ( defined SPMD )
   if ( npes > 1 ) then
      call lis_log_msg('DBG: lis_log_blocked_msg -- waiting at barrier')
      call MPI_BARRIER(MPI_COMM_WORLD,ierr)
      call lis_log_msg('DBG: lis_log_blocked_msg -- passed barrier')
   endif
#endif
```

1.11 Fortran: Module Interface lis_module.F90 (Source File: lis_module.F90)

Module for LDAS variable specification. This file will contain no tile space or grid space variables.

REVISION HISTORY:

```
   15 Oct 1999: Paul Houser; Initial code
   4  Apr 2000: Jeffrey Walker; Added some catchment model variables
   11 Apr 2000: Brian Cosgrove; Elevation correction and Forcing Mask
                variables added
   6  Jun 2000: Jon Radakovich; Updated for new version of CLM
   23 Feb 2001: Urszula Jambor; Updated for GEOS & GDAS forcing in GLDAS
   27 Feb 2001: Brian Cosgrove; Added Catchment forcing data variables
   15 Mar 2001: Jon Gottschalck; Updated for GDAS initialization of Mosaic
   12 Apr 2001: Urszula Jambor; Added domain,lsm,& force namefile paramters
   30 Apr 2001: Jon Radakovich; Update for PSAS temperature assimilation
   17 Jul 2001: Jon Gottschalck; Update for global precipitation variables
   30 Jul 2001: Matt Rodell; Add new soil parameter variables
   05 Sep 2001: Brian Cosgrove; Add variables for PAR and BRTTMP, remove
```

```
              1/4 to 1/8 interp variables
 15 Oct 2001: Jesse Meng; Replace agrmet flag by agrmetsw and agrmetlw
 27 Nov 2001: Jon Gottschalck; Added variables for AVHRR LAI data
 07 Dec 2001: Urszula Jambor; Added LDAS_KGDS array
 03 Feb 2002: Jon Gottschalck; Added Koster tilespace variables
 05 Feb 2002: Brian Cosgrove; Added NLDAS 11 Layer soil class file from Yun Duan
              and ftype variable to indicate NLDAS ETA forcing source used
 08 Feb 2002: Urszula Jambor; Added latmax for AGRMET use.
 15 Apr 2002: Urszula Jambor; Added ECMWF forcing options.
 28 Apr 2002: Kristi Arsenault; Added NOAH LSM parameters and code
 14 Nov 2002; Sujay Kumar; Optimized version for LIS
 14 Oct 2003; Sujay Kumar; Removed LSM specific variables.
 17 Oct 2003: Yudong Tian; Added IC, IR for regional runs
```

INTERFACE:

```
module lis_module
```

USES:

```
   implicit none
```

ARGUMENTS:

```
   type lisdomain
      integer :: nch            !actual number of tiles
      integer :: lsm            !land surface model (2=clm,4=noah)
      integer :: soil           !soil parameter scheme (1=original veg-based, 2=reynolds soils)
      integer :: elev           !elevation difference base
      integer :: glbnch          !actual global number of tiles
      integer :: ngrid           !actual number of grids
      integer :: glbngrid        !actual global number of grids
      integer :: domain         !model domain, (1=nldas, 2=gldas)
      integer :: landcover      ! landcover type
      integer :: gnc            !global array (different if subsetting is used)
      integer :: gnr            !global array (different if subsetting is used)
      integer :: lnc            !local number of columns in grid
      integer :: lnr            !local number of rows in grid
      integer :: ic             ! column index of sub-domain block
      integer :: ir             ! row index of sub-domain block
      integer :: maxt           !maximum tiles per grid
      real :: mina              !min grid area for tile (%)
      real :: udef              !undefined value
      real :: gridDesc(50)      !grid definition array
      real :: soil_gridDesc(6)   !grid definition for soil dataset
      real :: elev_gridDesc(6)   !grid definition for elev dataset
      real :: lc_gridDesc(6)   !grid definition for landcover dataset
   end type lisdomain

   type lisforcing
```

```
   integer :: force          !forcing data type (1=gdas,2=geos)
   integer :: ecor           !use elevation correction
   integer :: nforce
   integer :: nf             !number of forcing variables
   integer :: nmif           !number of forcing variables for model initialization option
   integer :: rstflag        !0=use only 1 forcing time, 1=find two forcing times upon resta
   integer :: gridchange
   integer :: interp
   integer :: latmax         !per hemisphere, for agrmet intepolation

   integer :: shortflag      !shortwave radiation source flag
                              !0=no radiation
                              !1=instantaneous sw
                              !2=time averaged sw
   integer :: longflag       !longwave radiation source flag
                              !0=no radiation
                              !1=instantaneous lw
                              !2=time averaged lw
   integer :: findtime1,findtime2
   integer :: findagrtime1,findagrtime2
   integer :: f00_flag, f06_flag
   integer :: gpcpsrc     !global precipitation flags
   integer :: radsrc
end type lisforcing

type lisparameters
   integer :: lai            !lai data source (1=original,
                             !2=avhrr satellite data
                             !3=modis satellite data)
   integer :: nt             !number of vegetation types
   integer :: vclass         !vegetation classification (1=umd)
   integer  :: laiflag        !satellite lai time
   integer  :: saiflag        !satellite lai time
   character*50 :: mfile     !land/water mask file for modelling (avhrr)
   character*50 :: vfile     !vegetation classification file (avhrr)!
   character*40 :: safile    !sand fraction map file
   character*40 :: clfile    !clay fraction map file
   character*40 :: po1file    !porosity map file
   character*40 :: po2file    !porosity map file
   character*40 :: po3file    !porosity map file
   character*40 :: slfile    !slope map file
   character*40 :: sifile    !silt map file
   character*40 :: avhrrdir     !avhrr data directory
   character*40 :: modisdir  !modis data directory
   character*40 :: iscfile   !soil color map file
   character*40 :: elevfile
   real*8  :: laitime        !satellite lai time
   real*8  :: saitime        !satellite sai time
```

```
end type lisparameters

type lisoutput
   integer :: wfor           !write forcing (0=no,1=yes)
   integer :: wtil           !write tile space data (0=no, 1=yes)
   integer :: wout           !output format option (1-binary, 2-grib)
   integer :: wsingle        !write one variable per file
   integer :: wparam         !write parameter file output
   integer :: expcode        !3 digit experiment code
   integer :: startcode      !0=restart date, 1=card date
   integer :: foropen
   integer :: numoutf        !counts number of output times for forcing data
   integer :: fidgm,rcgm,fidtm,rctm
   character*40 :: odir      !output data base directory
   character*40 :: dfile     !runtime diagnostics file
end type lisoutput

type listime
   integer :: sss            !starting second
   integer :: sdoy           !starting day of year
   integer :: smn            !starting minute
   integer :: shr            !starting hour
   integer :: sda            !starting day
   integer :: smo            !starting month
   integer :: syr            !starting year
   integer :: endcode        !0=realtime, 1=specific date
   integer :: ess            !ending second
   integer :: emn            !ending minute
   integer :: edoy           !ending day of year
   integer :: ehr            !ending hour
   integer :: eda            !ending day
   integer :: emo            !ending month
   integer :: eyr            !ending year
   integer :: ts             !timestep (seconds)
   integer :: tscount        !timestep count
   integer :: yyyymmdd,hhmmss
   integer :: doy,yr,mo,da,hr,mn,ss !lis current model timing variables
   integer :: endtime        !lis stop (0=continue time looping)
   integer :: pda            !lis previous timestep day
   real*8 :: time            !lis current model time in years
   real*8 :: etime           !lis end time in years
   real :: gmt,egmt,sgmt
end type listime

type lisassimil
   integer :: rpsas, rbias,ribc,rdbc,rsdbc
end type lisassimil
```

```
   type lisdec
      type(lisdomain)     :: d
      type(lisforcing)    :: f
      type(lisparameters) :: p
      type(listime)       :: t
      type(lisoutput)     :: o
      type(lisassimil)    :: a
   end type lisdec
```

1.12 Fortran: Module Interface lis_openfileMod.F90 (Source File: lis_openfileMod.F90)

This module contains interfaces and subroutines for opening data files.

REVISION HISTORY:

```
    08Apr04    James Geiger Initial Specification
```

1.12.1 lis_set_filename (Source File: lis_openfileMod.F90)

This routine overwrites the path for a GDS run

INTERFACE:

```
 subroutine lis_set_filename(file,time_offset)
 #if ( defined OPENDAP )
    use opendap_module, only : opendap_data_prefix, ciam
 #endif
   character(len=*), intent(inout):: file
   character(len=*), optional :: time_offset
 #if ( defined OPENDAP )
   if ( use_opendap_server ) then
      if ( PRESENT(time_offset) ) then
         file = trim(opendap_data_prefix)//'/'// &
              trim(adjustl(ciam))//'/'//"var_"//time_offset//".bin"
      else
         file = trim(opendap_data_prefix)//'/'// &
              trim(adjustl(ciam))//'/'//"var.bin"
      endif
   endif
 #endif
 end subroutine lis_set_filename
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\
```

```
\subsubsection{lis\_open\_file  (Source File: lis\_openfileMod.F90)}



   This routine is a generic open routine.  It parses its optional input
   arguments and builds an approriate open call.  It also determines
   whether or not data must be retrieve via a GraDS-DODS data server (GDS).
   If so, it calls the specified GDS script.

\bigskip{\sf INTERFACE:}
\begin{verbatim} subroutine lis_open_file(unit, file, form, status, access, recl, script, time_

    implicit none
```

INPUT PARAMETERS:

```
    integer,          intent(in) :: unit
    character(len=*), intent(in) :: file
    character(len=*), optional   :: form
    character(len=*), optional   :: status
    character(len=*), optional   :: access
    integer,          optional   :: recl
    character(len=*), optional   :: script
    character(len=*), optional   :: time_offset
```

LOCAL VARIABLES:

```
    integer                      :: ios
    character(len=11)            :: form_use
    character(len=7)             :: status_use
    character(len=10)            :: access_use
    character(len=15)            :: script_use
    character(len=4)             :: cunit
```

1.12.2 lis_read_file (Source File: lis_openfileMod.F90)

This routine is a generic read routine. It parses its optional input arguments and builds an approriate read call.

INTERFACE:

```
 subroutine lis_read_file(unit, array)
   implicit none
   integer,          intent(in) :: unit
   real,          intent(inout) :: array(lis_nc_data, lis_nr_data)
```

```
   integer :: line1, line2, line
   integer :: c,r, glnc, glnr

 #if ( defined OPENDAP )
   line = 1
   do r=1,lis_nr_data
      do c=1,lis_nc_data
         read(unit,rec=line) array(c,r)
         line = line + 1
      enddo
   enddo
 #else
    if(lis%d%domain.eq.8) then
    if(lis%d%gridDesc(9) .eq. 0.01) then
      line1 = nint((lis%d%gridDesc(4)-lis%d%gridDesc(44))/lis%d%gridDesc(9))+1
      line2 = nint((lis%d%gridDesc(5)-lis%d%gridDesc(45))/lis%d%gridDesc(10))+1
      do r=1,lis%d%lnr
         do c=1,lis%d%lnc
            glnc = line2+c-1
            glnr = line1+r-1
            line = (glnr-1)*nint(lis%d%gridDesc(42))+glnc
            read(unit,rec=line) array(c,r)
         enddo
      enddo
    else
       read(unit) array
    endif
 #endif
 end subroutine lis_read_file

%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{retrieve\_data  (Source File: lis\_openfileMod.F90)}



   This routine retrieves data from a GDS.  It will make 3 attempts to
   retrieve data from the server.  If the data cannot be retrieved, this
   routine aborts by calling endrun.

\bigskip{\sf INTERFACE:}
\begin{verbatim} subroutine retrieve_data(file, script,time_offset)
```

1.12.3 retrieve_script (Source File: lis_openfileMod.F90)

This routine retrieves makes the system call that executes the GrADS script that retrieves data from a GDS.

INTERFACE:

```
subroutine retrieve_script(file, script, time_offset)
```

1.13 Fortran: Module Interface lis_flush (Source File: lis_utilities.F90)

This routine is a generic interface to the flush routine.

REVISION HISTORY:

```
16 Nov 2004    James Geiger Initial Specification
```

1.14 Fortran: Module Interface lsm_module.F90 (Source File: lsm_module.F90)

This module contains interfaces and subroutines that control land surface model initialization, execution, reading and writing of restart files and other relevant land surface model computations.

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module lsm_module
```

1.14.1 LIS_setuplsm (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_setuplsm
```

1.14.2 LIS_lsm_main (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_lsm_main
```

1.14.3 LIS_force2tile (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_force2tile
```

1.14.4 LIS_readrestart (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_readrestart
```

1.14.5 LIS_writerestart (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_writerestart
```

1.14.6 LIS_lsm_output (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_lsm_output
```

1.14.7 LIS_setDynlsm (Source File: lsm_module.F90)

INTERFACE:

```
interface LIS_setDynlsm
```

INTERFACE:

```
subroutine LIS_lsm_init()
```

DESCRIPTION:

Setup functions for each land surface model

USES:

```
     use lisdrv_module, only: lis
     use lsm_pluginMod
 #if ( defined OPENDAP )
     use opendap_module
 #endif
     use lis_indices_module
```

CONTENTS:

```
! These (opendap_init, lis_set_indices) are done here to prevent a
! circular module dependency with lisdrv_module
#if ( defined OPENDAP )
    call opendap_init()
#endif
    call lis_set_indices()

    call lsm_plugin

    if ( lis%o%wsingle == 1 ) then
       call lsmini(lis%d%lsm,lis%d%nch)
    else
       call lsmini(lis%d%lsm,lis%d%glbnch)
    endif
```

1.14.8 lsm_tile_allocate (Source File: lsm_module.F90)

Allocates memory for land surface model variables

INTERFACE:

```
   subroutine lsm_tile_allocate()
```

USES:

CONTENTS:

1.14.9 lsm_setup (Source File: lsm_module.F90)

Completes land surface model initilaization

INTERFACE:

```
subroutine lsm_setup()
```

USES:

```
use lisdrv_module, only : lis
use tile_spmdMod, only : masterproc
```

CONTENTS:

```
call lsmsetup(lis%d%lsm)
```

1.14.10 run_lsm (Source File: lsm_module.F90)

Executes land surface model runs

INTERFACE:

```
subroutine run_lsm()
```

USES:

```
use lisdrv_module, only: lis
```

CONTENTS:

```
call lsmrun(lis%d%lsm)
```

1.14.11 lsm_readrestart (Source File: lsm_module.F90)

Reads restart files

INTERFACE:

```
subroutine lsm_readrestart()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmrestart(lis%d%lsm,1)
```

1.14.12 write_output (Source File: lsm_module.F90)

Writes output of land surface model runs

INTERFACE:

```
subroutine write_output()
```

USES:

```
use lisdrv_module, only: lis
```

CONTENTS:

```
call lsmoutput(lis%d%lsm)
```

1.14.13 setDynParams (Source File: lsm_module.F90)

Updates time dependent land surface model parameters

INTERFACE:

```
subroutine setDynParams()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmdynsetup(lis%d%lsm)
```

1.14.14 lsm_f2t (Source File: lsm_module.F90)

Transfers grid forcing to model tiles.

INTERFACE:

```
subroutine lsm_f2t()
```

USES:

```
use lisdrv_module, only: lis, grid,tile
use tile_spmdMod
use grid_spmdMod, only : gdisp
```

CONTENTS:

```
do t=1, di_array(iam)
   index = tile(t)%index -gdisp(iam)
   call lsmf2t(lis%d%lsm, t, grid(index)%forcing)
enddo
```

1.14.15 lsm_writerestart (Source File: lsm_module.F90)

Writes restart files

INTERFACE:

```
subroutine lsm_writerestart()
```

USES:

```
use lisdrv_module, only : lis
```

CONTENTS:

```
call lsmwrst(lis%d%lsm)
```

1.14.16 makepdsn (Source File: makepdsn.F90)

This routine computes the coefficients and terms required for grib output

REVISION HISTORY:

```
3-19-02  Cosgrove; Altered code for use with CLM writing
         intervals
5-17-02  Cosgrove; Changed code to use ldas%lsm instead of
         ldas%rmos and ldas%rclm so that forcing generation
         would still work when rmos set to zero in card
28-5-02  Arsenault; Altered code for use with NOAH writing
         intervals
```

INTERFACE:

```
subroutine makepdsn(yesterday, beforeyester, kpds, hour, writeint)
```

1.15 Fortran: Module Interface mpishorthand.F90 (Source File: mpishorthand.F90)

Data and parameters used for MPI. Some shorthand variables with shorter names than the standard MPI parameters. Also some variables used for heap management. Adopted from CLM

REVISION HISTORY:

```
02 Jan 2002 : Sujay Kumar: Initial Version
```

INTERFACE:

```
module mpishorthand
```

USES:

```
#if (defined SPMD)
#if (defined OSF1)
  include 'mpif.h'
#else
  use mpi
#endif
#endif
```

1.16 Fortran: Module Interface obsprecipforcing_module.F90 (Source File: obsprecipforcing_module.F90)

This module contains interfaces and subroutines that controls the incorporation of observed precipitation forcing

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module obsprecipforcing_module
```

ARGUMENTS:

```
  real, allocatable :: obsprecip(:)
```

1.16.1 LIS_obsprecipforcing_init (Source File: obsprecipforcing_module.F90)

INTERFACE:

```
  interface LIS_obsprecipforcing_init
     module procedure precip_forcing_init
  end interface
```

1.16.2 LIS_get_obsprecip_forcing (Source File: obsprecipforcing_module.F90)

INTERFACE:

```
  interface LIS_get_obsprecip_forcing
     module procedure get_obsprecip_forcing
  end interface
```

1.16.3 precip_forcing_init (Source File: obsprecipforcing_module.F90)

Initializes the variables required for the spatial interpolation of observed precipitation data

INTERFACE:

```
subroutine precip_forcing_init()
```

USES:

```
 use lisdrv_module, only :lis
  use def_ipMod, only : allocate_ip, def_ip_input
 use precipforcing_pluginMod, only : precipforcing_plugin
 use spmdMod, only : masterproc, iam
 use grid_spmdMod, only : gdi
```

CONTENTS:

```
if(lis%f%gpcpsrc.gt.0) then
   call precipforcing_plugin
   if(masterproc) then
      call defnatrespcp(lis%f%gpcpsrc)
      allocate(obsprecip(lis%d%ngrid))
   else
      allocate(obsprecip(gdi(iam)))
   endif
endif
```

1.16.4 get_precip_forcing (Source File: obsprecipforcing_module.F90)

Calls the routines to open, read, and interpolate observed precipitation forcing data

INTERFACE:

```
subroutine get_obsprecip_forcing
```

USES:

```
 use lisdrv_module, only: lis, gindex
 use grid_spmdMod
```

CONTENTS:

```
if(lis%f%gpcpsrc.gt.0) then
   if(masterproc) then
      call glbprecip(lis%f%gpcpsrc)
   endif
   if(lis%f%findtime1 ==1 .or.  &
        lis%f%findtime2==1) then
      if(npes > 1) then
```

```
            call scatter_precip_data()
         endif
      endif
      call timeinterppcp(lis%f%gpcpsrc)
   endif
```

1.16.5 scatter_precip_data (Source File: obsprecipforcing_module.F90)

Distributes the precipitation forcing data onto the compute nodes

INTERFACE:

```
subroutine scatter_precip_data()
```

USES:

```
use grid_spmdMod
```

1.17 Fortran: Module Interface obsradforcing_module.F90 (Source File: obsradforcing_module.F90)

This module contains interfaces and subroutines that controls the incorporation of observed radiation forcing

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module obsradforcing_module
  implicit none
```

ARGUMENTS:

```
real, pointer :: obswdata1(:)
real, pointer :: obswdata2(:)
real, pointer :: oblwdata1(:)
real, pointer :: oblwdata2(:)
integer :: sstat1, sstat2, lstat1,lstat2
```

1.17.1 LIS_obsradforcing_init (Source File: obsradforcing_module.F90)

INTERFACE:

```
interface LIS_obsradforcing_init
   module procedure rad_forcing_init
end interface
```

1.17.2 LIS_get_obsrad_forcing (Source File: obsradforcing_module.F90)

INTERFACE:

```
interface LIS_get_obsrad_forcing
   module procedure get_obsrad_forcing
end interface
```

1.17.3 rad_forcing_init (Source File: obsradforcing_module.F90)

Allocates memory for variables required for radiation forcing interpolation

INTERFACE:

```
subroutine rad_forcing_init()
```

USES:

```
 use lisdrv_module, only:lis
 use grid_spmdMod
 use radforcing_pluginMod, only :radforcing_plugin
```

CONTENTS:

```
    if(lis%f%radsrc.gt.0) then
       call radforcing_plugin
#if ( defined OPENDAP )
          call defnatresrad(lis%f%radsrc)
          allocate(obswdata1(gdi(iam)))
          allocate(obswdata2(gdi(iam)))
          allocate(oblwdata1(gdi(iam)))
          allocate(oblwdata2(gdi(iam)))
#else
       if(masterproc) then
          call defnatresrad(lis%f%radsrc)
          allocate(obswdata1(lis%d%ngrid))
          allocate(obswdata2(lis%d%ngrid))
          allocate(oblwdata1(lis%d%ngrid))
          allocate(oblwdata2(lis%d%ngrid))
```

```
       else
          allocate(obswdata1(gdi(iam)))
          allocate(obswdata2(gdi(iam)))
          allocate(oblwdata1(gdi(iam)))
          allocate(oblwdata2(gdi(iam)))
       endif
#endif
    endif
```

1.17.4 get_obsrad_forcing (Source File: obsradforcing_module.F90)

Calls the routines that read observed radiation forcing methods

INTERFACE:

```
   subroutine get_obsrad_forcing
```

USES:

```
     use lisdrv_module, only: lis, grid
     use grid_spmdMod
     use driverpardef_module
```

CONTENTS:

```
!    sstat1 = 0
!    sstat2 = 0
!    lstat1 = 0
!    lstat2 = 0
    if(lis%f%radsrc.gt.0) then
#if ( defined OPENDAP )
       call getrad(lis%f%radsrc)
#else
#if (defined SPMD)
       call MPI_GATHERV(grid(1:gdi(iam)),gdi(iam), &
            MPI_GRID_STRUCT,grid,gdi,gdisp,MPI_GRID_STRUCT, &
            0,MPI_COMM_WORLD, ier)
#endif
       if(masterproc) then
          call getrad(lis%f%radsrc)
       endif
#endif
#if ( ! defined OPENDAP )
#if (defined SPMD)
       call MPI_BCAST(sstat1, 1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
       call MPI_BCAST(sstat2, 1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
```

```
       call MPI_BCAST(lstat1, 1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
       call MPI_BCAST(lstat2, 1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
       call MPI_BCAST(lis%f%findagrtime1,1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
       call MPI_BCAST(lis%f%findagrtime2,1,MPI_INTEGER,0, &
            MPI_COMM_WORLD, ier)
       if ( lis%f%findagrtime1 == 1 .or. lis%f%findagrtime2 == 1 ) then
          if ( npes > 1 ) then
             call scatter_rad_data()
          endif
       endif
#endif
#endif
       call timeinterprad(lis%f%radsrc)
    endif
```

1.17.5 scatter_rad_data (Source File: obsradforcing_module.F90)

Distributes radiation forcing data on to the compute node.

INTERFACE:

```
subroutine scatter_rad_data()
```

USES:

```
use grid_spmdMod
use lisdrv_module, only : lis
```

1.18 Fortran: Module Interface opendap_module.F90 (Source File: opendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O

REVISION HISTORY:

```
10 Jul 2003; James Geiger  Initial Specification
```

INTERFACE:

```
module opendap_module
#if ( defined OPENDAP )
```

USES:

```
use time_manager
use lisdrv_module,  only : lis, grid, gindex
use grid_spmdMod,   only : gdi, gdisp
use tile_spmdMod,   only : di_array
use spmdMod

implicit none
```

ARGUMENTS:

```
integer      :: tnroffset    ! Global to local row mapping offset
integer      :: grid_offset  ! Global to local grid mapping offset
real         :: output_slat  ! Southern latitude boundary for the
                             ! interpolated domain
real         :: output_nlat  ! Northern latitude boundary for the
                             ! interpolated domain
real         :: output_wlon  ! Western longitude boundary for the
                             ! interpolated domain
real         :: output_elon  ! Eastern longitude boundary for the
                             ! interpolated domain
integer      :: parm_slat    ! Southern latitude boundary for each
                             ! subdomain (for parameter data)
integer      :: parm_nlat    ! Northern latitude boundary for each
                             ! subdomain (for parameter data)
integer      :: parm_wlon    ! Western longitude boundary for each
                             ! subdomain (for parameter data)
integer      :: parm_elon    ! Eastern longitude boundary for each
                             ! subdomain (for parameter data)
integer      :: parm_nc      ! Number of columns for each subdomain
                             ! (for parameter data)
integer      :: parm_nr      ! Number of rows for each subdomain
                             ! (for parameter data)
   integer :: domain

character*3  :: ciam         ! character representation of processor id
character*3  :: cdom         ! character representation of domain no
character*5  :: cparm_slat   ! character representation of cparm$_-$slat
character*5  :: cparm_nlat   ! character representation of cparm$_-$nlat
character*5  :: cparm_wlon   ! character representation of cparm$_-$wlon
character*5  :: cparm_elon   ! character representation of cparm$_-$elon
character*40 :: opendap_data_prefix ! prefix to prepend to data paths
```

1.18.1 opendap_init (Source File: opendap_module.F90)

Initializes the GDS variables

INTERFACE:

```
subroutine opendap_init()
```

CONTENTS:

```
call opendap_readcard()
call init_parm_vars()
call reset_lis_filepaths()
```

1.18.2 opendap_readcard (Source File: opendap_module.F90)

Reads the opendap namelist from the lis.crd card file

INTERFACE:

```
subroutine opendap_readcard()
```

CONTENTS:

```
open(10,file='lis.crd',form='formatted',status='old')
read(unit=10,nml=opendap)
print*,'MSG: opendap_readcard -- Using data prefix ', &
       opendap_data_prefix, ' (', iam, ')'
close(10)
```

1.18.3 reset_lis_filepaths (Source File: opendap_module.F90)

Resets various input data filenames for execution through GDS

INTERFACE:

```
subroutine reset_lis_filepaths()
```

CONTENTS:

```
lis%p%clfile       = trim(opendap_data_prefix)//'/'// &
                     trim(adjustl(ciam))//'/'//lis%p%clfile
lis%p%safile       = trim(opendap_data_prefix)//'/'// &
                     trim(adjustl(ciam))//'/'//lis%p%safile
lis%p%iscfile      = trim(opendap_data_prefix)//'/'// &
                     trim(adjustl(ciam))//'/'//lis%p%iscfile
```

1.18.4 init_parm_vars (Source File: opendap_module.F90)

Computes domain decomposition for native as well as interpolated domains for input data

INTERFACE:

```
subroutine init_parm_vars()
```

CONTENTS:

```
if(lis%d%gridDesc(9) .eq. 0.01) then
   domain = 8
elseif(lis%d%gridDesc(9).eq.0.05) then
   domain = 7
elseif(lis%d%gridDesc(9) .eq. 0.125) then
   domain = 6
elseif(lis%d%gridDesc(9) .eq. 0.25) then
   domain = 5
elseif(lis%d%gridDesc(9) .eq. 0.50) then
   domain = 4
elseif(lis%d%gridDesc(9) .eq. 1.0) then
   domain = 3
elseif((lis%d%gridDesc(9) .eq. 2) .and.  &
     (lis%d%gridDesc(10) .eq. 2.5)) then
   domain = 2
endif

select case (domain)
case(1)
   print*,'Error!  Cannot handle nldas.'
   stop 999
case(2)
   print*,'Error!  Cannot handle 2x2.5.'
   stop 999
case(3)  ! 1 deg
   res = 1000
   center = 500
   s_origin = -59500
   w_origin = -179500
case(4) ! 1/2 deg
   res = 500
   center = 750
   s_origin = -59750
   w_origin = -179750
case(5) ! 1/4 deg
   res = 250
   center = 875
   s_origin = -59875
   w_origin = -179875
case(6) ! 5km
```

```
      res = 50
      center = 975
      s_origin = -59975
      w_origin = -179975
   case(7) ! 5km
      res = 50
      center = 975
      s_origin = -59975
      w_origin = -179975
   case(8) ! 1km
      res = 10
      center = 995
      s_origin = -59995
      w_origin = -179995
   case DEFAULT
      print*, "Select domain size (1,2,3,4,5,6,7,8)"
      stop 999
   end select

   call set_parm_lat(parm_slat,parm_nlat,output_slat,output_nlat,&
                     parm_wlon,parm_elon,output_wlon,output_elon,&
                     res,s_origin,w_origin)
   !parm_wlon = 1
   !parm_elon = lis%d%gnc
   !parm_wlon = 1 + lis%d%gnc * (lis%d%ic - 1)
   !parm_elon = lis%d%gnc + lis%d%gnc * (lis%d%ic - 1)

   lis%d%ngrid = gdi(iam)
   lis%d%nch   = di_array(iam)

   !parm_nc    = lis%d%gnc
   parm_nc    = ( parm_elon - parm_wlon + 1 )
   parm_nr    = ( parm_nlat - parm_slat + 1 )

   if ( lis%d%ir > 1 ) then ! running the special 1km regional domain
      tnroffset  = parm_slat - 1 - (lis%d%ir - 1)*lis%d%gnr
   else ! running the ``normal'' way
      tnroffset  = parm_slat - 1
   endif
   grid_offset = tile(1)%index-1

   write(ciam, '(i3)') iam
   write(cdom, '(i3)') domain
   write(cparm_slat, '(i5)') parm_slat
   write(cparm_nlat, '(i5)') parm_nlat
   write(cparm_wlon, '(i5)') parm_wlon
   write(cparm_elon, '(i5)') parm_elon
```

```
print*,'DBG: parm_init -- parm_slat', parm_slat, ' (', iam, ')'
print*,'DBG: parm_init -- parm_nlat', parm_nlat, ' (', iam, ')'
print*,'DBG: parm_init -- parm_wlon', parm_wlon, ' (', iam, ')'
print*,'DBG: parm_init -- parm_elon', parm_elon, ' (', iam, ')'
print*,'DBG: parm_init -- output_slat', output_slat, ' (', iam, ')'
print*,'DBG: parm_init -- output_nlat', output_nlat, ' (', iam, ')'
print*,'DBG: parm_init -- output_wlon', output_wlon, ' (', iam, ')'
print*,'DBG: parm_init -- output_elon', output_elon, ' (', iam, ')'
print*,'DBG: parm_init -- ngrid', lis%d%ngrid, ' (', iam, ')'
print*,'DBG: parm_init -- glbngrid', lis%d%glbngrid, ' (', iam, ')'
print*,'DBG: parm_init -- lnc', lis%d%lnc, ' (', iam, ')'
print*,'DBG: parm_init -- lnr', lis%d%lnr, ' (', iam, ')'
print*,'DBG: parm_init -- gnc', lis%d%gnc, ' (', iam, ')'
print*,'DBG: parm_init -- gnr', lis%d%gnr, ' (', iam, ')'
print*,'DBG: parm_init -- tnroffset', tnroffset, ' (', iam, ')'
print*,'DBG: parm_init -- grid_offset', grid_offset, ' (', iam, ')'
print*,'DBG: parm_init -- nch', lis%d%nch, ' (', iam, ')'
print*,'DBG: parm_init -- ngrid', lis%d%ngrid, ' (', iam, ')'
print*,'DBG: parm_init -- glbnch', lis%d%glbnch, ' (', iam, ')'
print*,'DBG: parm_init -- glbngrid', lis%d%glbngrid, ' (', iam, ')'
print*,'DBG: parm_init -- cparm_slat ', cparm_slat, ' (', iam, ')'
print*,'DBG: parm_init -- cparm_nlat ', cparm_nlat, ' (', iam, ')'
print*,'DBG: parm_init -- ciam ', ciam, ' (', iam, ')'
print*,'DBG: parm_init -- parm_nc', parm_nc, ' (', iam, ')'
print*,'DBG: parm_init -- parm_nr', parm_nr, ' (', iam, ')'
print*,'DBG: parm_init -- gdi(iam)', gdi(iam), ' (', iam, ')'
print*,'DBG: parm_init -- tile(1)%row', tile(1)%row, ' (', iam, ')'
print*,'DBG: parm_init -- tile(1)%col', tile(1)%col, ' (', iam, ')'
print*,'DBG: parm_init -- tile(gdi(iam))%row', tile(gdi(iam))%row, &
       ' (', iam, ')'
print*,'DBG: parm_init -- tile(gdi(iam))%col', tile(gdi(iam))%col, &
       ' (', iam, ')'

call define_gds(lis)
```

1.18.5 set_parm_lat (Source File: opendap_module.F90)

Computes the latitudes of the decomposed domain for the parameter data

INTERFACE:

```
subroutine set_parm_lat(slat, nlat, oslat, onlat, &
                        wlon, elon, owlon, oelon, &
                        res, s_origin, w_origin)
```

USES:

```
      use lisdrv_module, only : tile
      implicit none
```

INPUT PARAMETERS:

```
      integer, intent(in) :: res, s_origin, w_origin
```

OUTPUT PARAMETERS:

```
      integer, intent(out) :: slat, nlat, wlon, elon
      real, intent(out)    :: oslat, onlat, owlon, oelon
```

CONTENTS:

```
      ! Set southern latitude index
      !slat = tile( 1 )%row
      slat = tile( 1 )%row  + (lis%d%ir-1)*lis%d%gnr

      ! Set southern latitude boundary
      oslat = s_origin + (slat-1)*res
      oslat = oslat / 1000.

      ! Set northern latitude index
      !nlat = tile( gdi(iam) )%row
      nlat = tile( gdi(iam) )%row  + (lis%d%ir-1)*lis%d%gnr

      ! Set northern latitude boundary
      onlat = s_origin + (nlat-1)*res
      onlat = onlat / 1000.

      ! Set western longitude index
      wlon = 1 + lis%d%gnc * (lis%d%ic - 1)

      ! Set western longitude boundary
      owlon = w_origin + (wlon-1)*res
      owlon = owlon / 1000.

      ! Set eastern longitude index
      elon = lis%d%gnc + lis%d%gnc * (lis%d%ic - 1)

      ! Set eastern longitude boundary
      oelon = w_origin + (elon-1)*res
      oelon = oelon / 1000.

    end subroutine set_parm_lat
#endif
```

1.19 Fortran: Module Interface precision.F90 (Source File: precision.F90)

Define the precision to use for floating point and integer operations throughout the model.

REVISION HISTORY:

```
14 Nov 2002; Sujay Kumar  Initial Specification
```

INTERFACE:

```
module precision
```

ARGUMENTS:

```
integer, parameter :: r4 = selected_real_kind(5)
integer, parameter :: r8 = selected_real_kind(6)
integer, parameter :: i8 = selected_int_kind(13)
```

1.19.1 readcard.F90 (Source File: readcard.F90)

Reads in LIS run specifics from lis.crd

REVISION HISTORY:

```
REVISION HISTORY:
 15 Oct 1999: Paul Houser; Initial code
 4  Apr 2000: Jeffrey Walker; Added catchment model output interval
 11 Apr 2000: Brian Cosgrove; Added Elevation correction and Forcing
              Mask read statements
 6  Jun 2000: Jon Radakovich; Updated for new version of CLM
 23 Feb 2001: Urszula Jambor; Added GEOS or GDAS forcing option
 27 Mar 2001: Jon Gottschalck; Revision of subroutine by implementing namelists
 05 Sep 2001: Brian Cosgrove; Altered forcing logfile output to include
              more precip types
 04 Feb 2002: Jon Gottschalck; Added section to set to Koster tilespace files if necessary
 15 Apr 2002: Urszula Jambor; Added ECMWF forcing options, also
              adding 1 & 1/2 degree GLDAS domain options.
 28 Apr 2002: Kristi Arsenault; Added NOAH LSM code
 14 Nov 2003: Sujay Kumar; Modified card file that includes regional
              modeling options
```

INTERFACE:

```
subroutine readcard
```

USES:

```
use lisdrv_module, only : lis
use time_manager, only : date2time
use soils_pluginMod, only : soils_plugin
use lai_pluginMod, only : lai_plugin
implicit none
```

CONTENTS:

```
  open(10,file='lis.crd',form='formatted',status='old')
!-----------------------------------------------------------------------
! Reading in parameters that need to be initialized
! to avoid any problems later
! Reading in parameters that are used by all LDAS runs
!-----------------------------------------------------------------------
  read(unit=10,nml=driver)
  read(unit=10,nml=lis_run_inputs)

!YDT: get command line arguments and convert to IC, IR
    lis%d%IC = 1
    lis%d%IR = 1
    ttmp=''
    Do i=1, iargc()
     call getarg(i, stmp)
     if (trim(stmp).eq.'-ic') then
      call getarg(i+1, stmp)
      Read(Unit=stmp, fmt='(i3)') lis%d%IC!fmt='(i30)' syntax is needed for IBM
      ttmp=trim(stmp)//'-'
     end if
     if (trim(stmp).eq.'-ir') then
      call getarg(i+1, stmp)
      Read(Unit=stmp, fmt='(i3)') lis%d%IR!fmt='(i30)' syntax is needed for IBM
      ttmp=trim(ttmp)//trim(stmp)
     end if
    End Do

!    if ( lis%o%odirn == 1 ) then
!       lis%o%odir_array(1) = trim(lis%o%odir_array(1))//trim(ttmp)
!    endif
!    lis%o%odir = lis%o%odir_array(1)
!-----------------------------------------------------------------------
! Open runtime diagnostics file
!-----------------------------------------------------------------------
  call openfile(name,lis%o%odir,lis%o%expcode,lis%o%dfile)
88 format(a4,25x,a3,5x,16a)
89 format(20x,a49)
!-----------------------------------------------------------------------
! Impose limit on time step
!-----------------------------------------------------------------------
  call check_timestep(lis%t%ts)
!-----------------------------------------------------------------------
! Set Time
!-----------------------------------------------------------------------
  call set_time(lis%t)
```

```
  call set_output_counters(lis%o%numoutf)

  call date2time(lis%t%time,lis%t%doy,lis%t%gmt, &
       lis%t%yr,lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)

  print*
  print*,'********** GSFC-LIS driver **********'
  print*,'experiment code: ','-',lis%o%expcode,'-'
  print*,'starting time: ',lis%t%smo,'/',lis%t%sda,'/',lis%t%syr
  print*,'ending time: ',lis%t%emo,'/',lis%t%eda,'/',lis%t%eyr
  print*

  print*,'forcing details:'
  read(unit=10,nml=landcover)
  read(unit=10,nml=elevation)
  read(unit=10,nml=soils)
  read(unit=10,nml=lai)

  call soils_plugin
  call lai_plugin

!-----------------------------------------------------------------------
!  Setting Satellite LAI variables
!-----------------------------------------------------------------------
  lis%p%laitime = 0.0
  lis%p%saitime = 0.0
  if(lis%p%lai.eq.2) then
     print*, "Using AVHRR Satellite LAI"
  endif
  if(lis%p%lai.eq.3) then
     print*, "Using MODIS Satellite LAI"
  endif

  lis%f%rstflag = 1
  lis%f%gridchange = 1
!-----------------------------------------------------------------------
! Initialize Statistics files conditional
!-----------------------------------------------------------------------
  lis%o%foropen=0
!-----------------------------------------------------------------------
! Fix MAXT out of bounds problems
!-----------------------------------------------------------------------
  if(lis%d%maxt.gt.lis%p%nt) lis%d%maxt=lis%p%nt
  if(lis%d%maxt.lt.1      ) lis%d%maxt=1
!-----------------------------------------------------------------------
! Select which vegetation tile space and mask files
!-----------------------------------------------------------------------
  print*, 'miscellaneous details:'
```

```
  if(lis%p%vclass.eq.1) print*, 'avhrr umd vegetation', lis%p%vclass
  if(lis%p%vclass.eq.2) print*, 'modis umd vegetation', lis%p%vclass
  print*,'mask file: ', lis%p%mfile
  print*,'vegetation file: ', lis%p%vfile
  if (lis%d%soil.eq.1) print *,'original vegetation-based soil scheme'
  if (lis%d%soil.eq.2) print *,'reynolds soils'
  if (lis%d%soil.eq.1) print *,'original vegetation-based soil scheme'
  if (lis%d%soil.eq.2) print *,'reynolds soils'

  close(10)
  return
```

1.19.2 check_timestep (Source File: readcard.F90)

check timestep to make sure it is between 1 and 3600

INTERFACE:

```
 subroutine check_timestep(timestep)
```

CONTENTS:

```
  if (timestep.gt.3600) then
     print *,'-------------------------------------'
     print *,'error, user timestep > 3600 seconds!!!'
     print *,'resetting lis%ts to 3600!!!!'
     print *,'-------------------------------------'
     timestep=3600
  endif
  if(timestep.lt.1) then
     print*,'Timestep can not be less than 1 minute, reset to 15 min.'
     timestep=15*60
  endif
  return
```

1.19.3 openfile (Source File: readcard.F90)

This subroutine puts together a filename

REVISION HISTORY:

```
    4 Jan 2000: Paul Houser; Original Code
```

INTERFACE:

```
 subroutine openfile(name, odir, expcode, ffile)
   implicit none
```

OUTPUT PARAMETERS:

```
character*80, intent(out):: name
```

INPUT PARAMETERS:

```
character*40, intent(in):: odir, ffile
integer, intent(in) :: expcode
character*80 mkdir
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Put together filename
!-----------------------------------------------------------------------
  write(unit=temp,fmt='(a40)') odir
  read(unit=temp,fmt='(80a1)') (fbase(i),i=1,80)
  c=0
  do i=1,80
     if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
  enddo

  write(unit=temp,fmt='(a4,i3,a1)')'/EXP',expcode,'/'
  read(unit=temp,fmt='(80a1)')(fcode(i),i=1,80)
  d=0
  do i=1,80
     if(fcode(i).eq.(' ').and.d.eq.0)d=i-1
  enddo
  write(unit=temp,fmt='(a40)') ffile
  read(unit=temp,fmt='(80a1)')(fname(i),i=1,80)
  e=0
  do i=1,80
     if(fname(i).eq.(' ').and.e.eq.0)e=i-1
  enddo

  write(unit=temp,fmt='(80a1)')(fbase(i),i=1,c), &
       (fcode(i),i=1,d), (fname(i),i=1,e)
  read(unit=temp,fmt='(a80)') name
  write(unit=temp,fmt='(a9)')'mkdir -p '
  read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)
!-----------------------------------------------------------------------
! Make the directories for the output files
!-----------------------------------------------------------------------
  write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9), &
       (fbase(i),i=1,c),(fcode(i),i=1,d)
  read(unit=temp,fmt='(a80)')mkdir
  call system(mkdir)
  return
```

1.20 Fortran: Module Interface spmdMod.F90 (Source File: spmdMod.F90)

MPI routines for initialization and computing arguments for different operations.

INTERFACE:

```
module spmdMod
```

ARGUMENTS:

```
#if (!defined SPMD)
  logical :: masterproc = .true. ! proc 0 logical for printing msgs
  integer :: iam = 0
  integer :: npes = 1
#endif

#if (defined SPMD)

#if (defined OFFLINE)
  use mpishorthand
#endif

#if (defined OFFLINE)
  integer :: npes        !number of processors
  integer :: iam         !proc number
  logical :: masterproc  !proc 0 logical for printing msgs
#endif
```

1.20.1 spmd_init (Source File: spmdMod.F90)

MPI initialization (number of cpus, processes, tids, etc)

INTERFACE:

```
  subroutine spmd_init
 DIR$ NAME (release_cache="_f90a_free_all")
```

CONTENTS:

```
    print*, 'in spmd mod'
#if (defined SPMD)
#if (defined OFFLINE)

    call mpi_init(ier)

#endif
    call mpi_comm_rank(MPI_COMM_WORLD, iam, ier)

    if (iam==0) then
```

```
      masterproc = .true.
   else
      masterproc = .false.
   end if
   call mpi_comm_size(MPI_COMM_WORLD, npes, ier)
   print*, '** Number of Procs **',npes
   return
```

1.20.2 stats.F90 (Source File: stats.F90)

Calculates statistics for a given variable

REVISION HISTORY:

```
Nov 11 1999:  Jon Radakovich; Initial code
```

INTERFACE:

```
subroutine stats(var,udef,nch,mean,stdev,min,max)
```

ARGUMENTS:

```
integer, intent(in) :: nch
real, intent(in)    :: var(nch), udef
real, intent(out)   :: mean,stdev,min,max
```

CONTENTS:

```
vsum=0.
mean=0.
dev=0.
stdev=0.
min=100000.
max=-100000.
count = 0
do t=1,nch
   if(var(t).ne.udef)then
      count = count +1
      vsum=vsum+var(t)
      if(var(t).gt.max)max=var(t)
      if(var(t).lt.min)min=var(t)
   endif
enddo
if(vsum.eq.0.)then
   max=0.
   min=0.
endif
```

```
if(count .ge.1) then
   mean=vsum/float(count)
else
   mean = 0
endif
count = 0
do t=1,nch
   if(var(t).ne.udef)then
      count = count + 1
      dev=dev+(var(t)-mean)**2
   endif
enddo
if(count .gt.1) then
   stdev=(dev*(float(count)-1)**(-1))**(0.5)
else
   stdev = 0
endif
return
```

Released May 2004
See SOFTWARE DISTRIBUTION POLICY for software distribution policies
The LIS source code and documentation are in the public domain, available without fee for educational, research, non-commercial and commercial purposes. Users may distribute the binary or source code to third parties provided this statement appears on all copies and that no charge is made for such copies.
NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED AS IS WITHOUT EXPRESS OR IMPLIED WARRANTY. NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
See COPYRIGHT.TXT for copyright details.

1.21 Fortran: Module Interface string_utils (Source File: string_utils.F90)

This module contains routines that perform string manipulations

REVISION HISTORY:

```
14 Nov 2002: Sujay Kumar: Initial Version, adopted from CLM
```

INTERFACE:

```
module string_utils
```

1.21.1 to_upper (Source File: string_utils.F90)

Convert character string to upper case.
Method: Use achar and iachar intrinsics to ensure use of ascii collating sequence.

INTERFACE:

```
function to_upper(str)
  implicit none
```

ARGUMENTS:

```
  character(len=*), intent(in) :: str      ! String to convert to upper case
  character(len=len(str))      :: to_upper
```

CONTENTS:

```
 do i = 1, len(str)
    ctmp = str(i:i)
    aseq = iachar(ctmp)
    if ( aseq >= 97  .and.  aseq <= 122 ) ctmp = achar(aseq - 32)
    to_upper(i:i) = ctmp
 end do
```

1.21.2 tile_module.F90 (Source File: tile_module.F90)

LIS non-model-specific tile variables only.

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial code
22 Aug 2000: Brian Cosgrove; Modified code for output of
             standard LDAS output variables--added CC and AC,
             the canopy and aerodynamic conductance
```

INTERFACE:

```
module tile_module

  implicit none
  public tiledec
```

ARGUMENTS:

```
  type tiledec
     integer :: col        !Grid Column of Tile
     integer :: row        !Grid Row of Tile
     integer :: index      !Index of corresponding grid
     integer :: vegt       !Vegetation Type of Tile
     real    :: fgrd       !Fraction of Grid covered by tile
  end type tiledec
```

1.22 Fortran: Module Interface tile_spmdMod.F90 (Source File: tile_spmdMod.F90)

This module contains routines for domain decomposition in tile space

REVISION HISTORY:

```
14 Nov 2002;  Sujay Kumar  Initial Specification
```

INTERFACE:

```
module tile_spmdMod
```

USES:

```
use spmdMod
```

ARGUMENTS:

```
integer, allocatable :: di_array(:) !array containing the sizes of the decomposed tile space
integer, allocatable :: displs(:) !array containing relative displacements fo the tile space
```

1.22.1 allocate_tiledd (Source File: tile_spmdMod.F90)

Allocates memory for arrays containing tile decomposition information

INTERFACE:

```
subroutine allocate_tiledd()
```

1.22.2 tile_spmd_init (Source File: tile_spmdMod.F90)

Performs domain decomposition in tile space

INTERFACE:

```
subroutine tile_spmd_init(tile, nch, nmif)
```

USES:

```
use tile_module
```

ARGUMENTS:

```
type(tiledec) :: tile(nch)
integer :: nch, nmif
```

CONTENTS:

```
ntiles = 1
deltax = nch/npes
do p=0,npes-2
   nti(p) = ntiles
   tindex = ntiles+deltax-1
   gind  = tile(tindex)%index
   do while(tile(tindex+1)%index ==gind)
      tindex = tindex+1
   enddo
   ntf(p) = tindex
   ntiles = tindex+1
enddo
nti(npes-1) = ntiles
ntf(npes-1) = nch
do i=0,npes-1
   di_array(i) = ntf(i)-nti(i)+1
enddo

displs(0) = 0
do i = 1, npes-1
   displs(i) = displs(i-1)+di_array(i-1)
enddo
print*, 'MSG: tile_spmd_init -- domain decomp', di_array, displs
```

1.23 Fortran: Module Interface time_manager.F90 (Source File: time_manager.F90)

This module contains wrapper functions that uses the ESMF time manager for time managment

INTERFACE:

```
module time_manager

   use spmdMod, only: masterproc
   use lis_module
   use precision
```

ARGUMENTS:

```
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 timestep
      get_prev_date,           &! return date components at beginning of current timestep
      get_start_date,          &! return date components of the start date
      get_ref_date,            &! return date components of the reference date
      get_curr_time,           &! return components of elapsed time since reference date
      get_curr_calday,         &! return calendar day at end of current timestep
      is_last_step,            &! return true on last timestep
      timemgr_write_restart,   &! write info to file needed to restart the time manager
      timemgr_read_restart,    &! read info from file needed to restart the time manager
      timemgr_restart,         &! restart the time manager
      tick,                    &
      date2time


  ==========================================================================================
 contains
  ==========================================================================================
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{timemgr\_init (Source File: time\_manager.F90)}




   Initialize the ESMF time manager.

   NOTE - This assumes that the namelist variables
   have been set before this routine is called.

\bigskip{\sf INTERFACE:}
\begin{verbatim} subroutine timemgr_init(lt)
```

1.23.1 timemgr_print (Source File: time_manager.F90)

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 a 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)

INTERFACE:

```
 subroutine timemgr_restart()
```

1.23.2 timemgr_print (Source File: time_manager.F90)

Prints the time manager information

INTERFACE:

```
subroutine timemgr_print(lt)
```

1.23.3 advance_timestep (Source File: time_manager.F90)

Increment the timestep number.

INTERFACE:

```
subroutine advance_timestep(lt)
```

CONTENTS:

```
  lt%ss = lt%ss + lt%ts

  do while(lt%ss .gt. 59)
     lt%ss = lt%ss - 60
     lt%mn = lt%mn + 1
  enddo

  do while(lt%mn .gt.59)
     lt%mn = lt%mn -60
     lt%hr = lt%hr+1
  enddo

  do while(lt%hr .ge.24)
     lt%hr = lt%hr -24
     lt%da = lt%da +1
  enddo

  if((mod(lt%yr,4) .eq. 0 .and. mod(lt%yr, 100).ne.0) &!leap year
       .or.(mod(lt%yr,400) .eq.0)) then
     days(2) = 29
  else
     days(2) = 28
  endif

  do while(lt%da.gt.days(lt%mo))
     lt%da = lt%da - days(lt%mo)
     lt%mo = lt%mo + 1
  enddo

  do while(lt%mo .gt. 12)
     lt%mo = lt%mo-12
```

```
     lt%yr = lt%yr +1
  enddo

  call date2time(lt%time,lt%doy,lt%gmt,&
       lt%yr, lt%mo, lt%da, lt%hr, lt%mn, lt%ss)

  lt%tscount = lt%tscount + 1

  write(*,24)'GSFC-LIS time: ',lt%mo,'/',lt%da,'/', &
       lt%yr,lt%hr,':',lt%mn,':',lt%ss
24 format(a16,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)

  if(lt%endcode.eq.0)then  !end at real-time date (tbd)
     write(*,*)'warning: do not know how to stop in real-time'
  endif
  if(lt%endcode.eq.1)then  !end on date specified in lis.crd file
     call date2time(lt%etime,lt%edoy,lt%egmt, &
          lt%eyr,lt%emo,lt%eda,lt%ehr,lt%emn,lt%ess)
     if(lt%time.ge.lt%etime)then
        lt%endtime=1
        write(*,*) 'GSFC-LDAS run completed'
     endif
  endif
```

1.23.4 get_step_size (Source File: time_manager.F90)

Return the step size in seconds.

INTERFACE:

```
function get_step_size(lt)
```

1.23.5 get_nstep (Source File: time_manager.F90)

Return the timestep number.

INTERFACE:

```
function get_nstep(lt)
```

1.23.6 get_curr_day (Source File: time_manager.F90)

Return date components valid at end of current timestep with an optional offset (positive or negative) in seconds.

INTERFACE:

```
subroutine get_curr_date(lt, yr, mon, day, tod, offset)
```

1.23.7 get_prev_date (Source File: time_manager.F90)

Return date components valid at beginning of current timestep.

INTERFACE:

```
subroutine get_prev_date(yr, mon, day, tod)
```

1.23.8 get_start_date (Source File: time_manager.F90)

Return date components valid at beginning of initial run.

INTERFACE:

```
subroutine get_start_date(yr, mon, day, tod)
```

1.23.9 get_ref_date (Source File: time_manager.F90)

Return date components of the reference date.

INTERFACE:

```
subroutine get_ref_date(yr, mon, day, tod)
```

1.23.10 get_curr_time (Source File: time_manager.F90)

Return time components valid at end of current timestep. Current time is the time interval between the current date and the reference date.

INTERFACE:

```
subroutine get_curr_time(days, seconds)
```

1.23.11 get_curr_calday (Source File: time_manager.F90)

Return calendar day at end of current timestep with optional offset. Calendar day 1.0 = 0Z on Jan 1.

INTERFACE:

```
function get_curr_calday(lt,offset)
```

1.23.12 is_last_step (Source File: time_manager.F90)

Return true on last timestep.

INTERFACE:

```
function is_last_step(lt)
```

1.23.13 timemgr_write_restart (Source File: time_manager.F90)

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.

INTERFACE:

```
subroutine timemgr_write_restart(ftn_unit)
```

1.23.14 timemgr_read_restart (Source File: time_manager.F90)

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.

INTERFACE:

```
subroutine timemgr_read_restart(ftn_unit)
```

---

determines time in years, based on year, month, day hour etc.. or reverse (date2time).

REVISION HISTORY:

```
15 oct 1999: paul houser; initial code
21 feb 2002: brian cosgrove; corrected leap year code line.  days(2)
             was not being reset to 28 after leaving a leap year,
             it was staying 29
```

INTERFACE:

```
subroutine date2time(time,doy,gmt,yr,mo,da,hr,mn,ss)

  implicit none
```

ARGUMENTS:

```
  integer yr,mo,da,hr,mn,ss,yrdays,doy,days(13),k
  real*8 time
  real gmt
```

CONTENTS:

```
if((mod(yr,4).eq.0.and.mod(yr,100).ne.0) &     !correct for leap year
     .or.(mod(yr,400).eq.0))then             !correct for y2k
   yrdays=366
else
   yrdays=365
endif

doy=0
do k=1,(mo-1)
   doy=doy+days(k)
enddo
doy=doy+da

if(yrdays.eq.366.and.mo.gt.2)doy=doy+1

time=(dfloat(yr)+((((((dfloat(ss)/60.d0)+dfloat(mn))/60.d0)+ &
     dfloat(hr))/24.d0)+dfloat(doy-1))/dfloat(yrdays))

gmt=( ( (float(ss)/60.0) +float(mn)) /60.0)+float(hr)
return
```

---

advance (or retract) time variables a specified amount (a nonmodular version of ticktime.f.)

REVISION HISTORY:

```
1  oct 1999: jared entin; initial code
15 oct 1999: paul houser; significant f90 revision
```

INTERFACE:

```
subroutine tick(time,doy,gmt,yr,mo,da,hr,mn,ss,ts)
  implicit none
```

ARGUMENTS:

```
  real*8 time
  integer days(13)
  integer yr,mo,da,hr,mn,ss,ts,doy
  real gmt
```

CONTENTS:

```
143 format(a1,' yr',i6,' mo',i5,' dy',i5,' hr',i5, &
         ' mn',i6,' ss',i8,' ts',i8)
    ss=ss+ts
    do while(ss.gt.59)
       ss=ss-60
       mn=mn+1
    enddo
    do while(ss.lt.0)
       ss=ss+60
       mn=mn-1
    enddo
    do while(mn.gt.59)
       mn=mn-60
       hr=hr+1
    enddo

    do while(mn.lt.0)
       mn=mn+60
       hr=hr-1
    enddo
    do while(hr.gt.23)
       hr=hr-24
       da=da+1
    enddo

    do while(hr.lt.0)
       hr=hr+24
       da=da-1
    enddo

    if((mod(yr,4).eq.0.and.mod(yr,100).ne.0) &    !correct for leap year
         .or.(mod(yr,400).eq.0))then               !correct for y2k
       days(2)=29
    else
       days(2)=28
    endif

    do while(da.gt.days(mo))
       da=da-days(mo)
       mo=mo+1
    enddo

    do while(da.lt.1)

       prvmo=mo-1
       if(mo.eq.1) prvmo=12
```

```
      da=da+days(prvmo)

      if(prvmo.eq.12) then
         mo=prvmo
         yr=yr-1
      else
         mo=prvmo
      endif
   enddo
   do while(mo.gt.12)
      mo=mo-12
      yr=yr+1
   enddo

   do while(mo.lt.1)
      mo=mo+12
      yr=yr-1
   enddo
   call date2time(time,doy,gmt,yr,mo,da,hr,mn,ss)
   return
```

1.23.15 chkrc (Source File: time_manager.F90)

Checks the return code for errors

INTERFACE:

```
subroutine chkrc(rc, mes)
```

CONTENTS:

```
if ( rc == esmf_success ) return
write(6,*) mes
call endrun
```

1.23.16 zterp.F90 (Source File: zterp.F90)

This subroutine is based, in part, on modified subroutines from Jean C. Morrill of the GSWP project. The program temporally interpolates time averge or instantaneous data to that needed by the model at the current timestep. It does this by combining a linear interpolation approach with a solar zenith angle approach in a fashion suitable for use with data such as short wave radiation values. It cannot be used with input data points which are more than 24 hours apart. The program outputs two weights which can then be applied to the original data to arrive at the interpolated data. If IFLAG=0, then WEIGHT1 is

the weight which should be applied to the time averaged data (from the time period which the model is currently in) to arrive at the interpolated value and weight 2 is not used at all. If IFLAG=1, then WEIGHT1 should be applied to the original instantaneous data located just prior to the model time step, and WEIGHT2 should be applied to the original instantaneous data located just after the model time step.
i.e. (IF IFLAG=0) interpolated data = (WEIGHT1 * time averaged data from time period that model is currently in)
i.e. (IF IFLAG=1) interp. data = (WEIGHT1*past data)+(WEIGHT2*future data)

REVISION HISTORY:

```
10/2/98 Brian Cosgrove
6/28/00 Brian Cosgrove; changed code so that it uses  LDAS%UDEF and
        not a hard-wired undefined value of -999.999.  Also changed
        call to ZTERP subroutine so that LDAS and GRID mod files
        are brought in and can be accessed in this subroutine
2/27/01 Brian Cosgrove; Added czmodel into call for ZTERP subroutine
10/7/01 Urszula Jambor; Added conditional to prevent stop in code when
        AGRMET data are available, but both endtime cos(zen.) are small.
```

INTERFACE:

```
subroutine zterp (iflag,lat,lon,btime,etime, &
     mbtime,julianb,weight1,weight2,czbegdata, &
     czenddata,czmodel,lis)
```

USES:

```
use lis_module      ! LDAS non-model-specific 1-D variables
implicit none
!INPUT PARAMETERS
type (lisdec) :: lis
integer        :: iflag    !Flag specifying if input data is
                           !time averaged or not
real           :: lat,lon  !Latitude and longitudes of current
                           !data point
real           :: mbtime   !Time of current time step Expects
                           !GMT time in hour fractions
integer        :: julianb  !Julian day upon which BTIME falls
real           :: interval
real           ::  btime   !beginning time of orig. avg. data (IFLAG=0) or
                           !time of original instantaneous data point which
                           !is located at or just prior to the current model
                           !time step (IFLAG=1).  Expects GMT time in hour
                           !fractions.  i.e., 6:30 Z would be 6.5.
real           :: etime    !Ending time of orig. avg. data (IFLAG=0) or
                           !time of original instantaneous data point which
                           !is located at or just after the current model
                           !time step (IFLAG=1).    Expects GMT time in hour
                           !fractions.  i.e., 6:30 Z would be 6.5.
```

```
   real           :: weight1  !weight applied to original time averaged
                              !data (IFLAG=0) or
                              !weight applied to orig instantaneous data point
                              !located just prior to the current model time step
   real           :: weight2  !weight applied to orig instantaneous
                              !data point located just after the current model
                              !time step (IFLAG=1)
                              !If IFLAG=0, then this weight is meaningless and
                              !should not be used
```

CONTENTS:

```
!-----------------------------------------------------------------------
! This section contains hardwired data that will be supplied by main program.
! These values were chosen arbitrarily and exist simply to check the
! functioning of the program.
!
! Initialize variables
!-----------------------------------------------------------------------
  i=1
  totangle=0
  weighte=lis%d%udef
  weightb=lis%d%udef
  weight1=lis%d%udef
  weight2=lis%d%udef
  czbegdata=lis%d%udef
  czenddata=lis%d%udef
  czmodel=lis%d%udef
  gmt=btime
  juliane=julianb
  julianmb=julianb
  juliantemp=julianb

  if (mbtime.lt.btime) julianmb=julianmb+1
  if (etime.le.btime) juliane=juliane+1
!-----------------------------------------------------------------------
!  First case, IFLAG=0 (Time average input, instantaneous output)
!  Compute time interval, here arbitrarily divided into 36 parts
!-----------------------------------------------------------------------
  if (iflag.eq.0) then
     call localtime (btime,lon,lbtime,zone)
     if (lbtime.gt.btime) julianb=julianb-1
     call coszenith(lon,lat,lbtime,zone,julianb,czbegdata)
     call localtime (etime,lon,letime,zone)
     if (letime.gt.etime) juliane=juliane-1
     call coszenith(lon,lat,letime,zone,juliane,czenddata)
     call localtime (mbtime,lon,lhour,zone)
     if (lhour.gt.mbtime) julianmb=julianmb-1
     call coszenith(lon,lat,lhour,zone,julianmb,czmodel)
```

```
     if (czmodel.eq.0) then
        weight1=0
        goto 818
     endif

     if (etime.gt.btime) then
        interval = ((etime-btime)/36.0)
     elseif (etime.lt.btime) then
        interval = (((24-btime)+(etime))/36.0)
     else
        interval=24.0/36.0
     endif
!-----------------------------------------------------------------
!       Compute cosine of zenith angle for each time interval
!-----------------------------------------------------------------
     czavgdata = 0.
     do while (i.le.37)
        if ((gmt+interval).lt.24) then
           gmt=gmt+interval
        else
           gmt=(interval-(24-gmt))
           juliantemp=juliantemp+1
        endif
        call localtime (gmt,lon,lhour,zone)
        call coszenith(lon,lat,lhour,zone, &
             juliantemp,czavgdata)
        totangle=totangle+czavgdata
        i=i+1
     enddo
!-----------------------------------------------------------------
!       Compute average cosine of zenith angle and also
!       weight which will be applied to original data (WEIGHT1
!-----------------------------------------------------------------
     avgangle=(totangle/37.0)
     if (avgangle.eq.0) then
        weight1=0
     else
        weight1=(czmodel/avgangle)
     endif
  endif
!-----------------------------------------------------------------
!       Second case:  IFLAG=1 (instantaneous input and output)
!-----------------------------------------------------------------
  if (iflag.eq.1) then
!-----------------------------------------------------------------
!       Compute local times and cosine (zenith angle)
!-----------------------------------------------------------------
     call localtime (btime,lon,lbtime,zonebtime)
```

```
     if (lbtime.gt.btime) julianb=julianb-1
     call localtime (etime,lon,letime,zoneetime)
     if (letime.gt.etime) juliane=juliane-1
     call localtime (mbtime,lon,lmbtime,zonembtime)
     if (lmbtime.gt.mbtime) julianmb=julianmb-1
     call coszenith (lon,lat,lbtime,zonebtime, &
          julianb,czbegdata)
     call coszenith (lon,lat,letime,zoneetime, &
          juliane,czenddata)
     call coszenith (lon,lat,lmbtime,zonembtime, &
          julianmb,czmodel)
!-------------------------------------------------------------------
!       Decision tree to deal with contingencies
!       If COS(zenith angle at current model time =0, weight =0
!       If COS(zenith angle =0 at beg. and end times, PROBLEM, STOP
!       Otherwise use beginning and ending data to calculate weight
!-------------------------------------------------------------------
     if (czmodel.le.0.01) then
        weight1=0
        weight2=0
     else
        if ((czbegdata.gt.0.01).or.(czenddata.gt.0.01)) then
           if (czbegdata.le.0.01) then
              weight1=0
              weight2=(czmodel/czenddata)
           endif
           if (czenddata.le.0.01) then
              weight1=(czmodel/czbegdata)
              weight2=0
           endif
           if((czenddata.gt.0.01).and. &
                (czbegdata.gt.0.01))then

              if (btime.le.mbtime) then
                 diffbm=mbtime-btime
              else
                 diffbm=24-btime+mbtime
              endif

              if (etime.ge.mbtime) then
                 diffem=etime-mbtime
              else
                 diffem=24-mbtime+etime
              endif

              if (etime.gt.btime) then
                 diffeb=etime-btime
              elseif (etime.eq.btime) then
```

```
                diffeb=24
             else
                diffeb=24-btime+etime
             endif
             weighte=(diffbm/diffeb)
             weightb=(diffem/diffeb)

             weight1=((czmodel/czbegdata)*weightb)
             weight2=((czmodel/czenddata)*weighte)
          endif
       else
          call lis_log_msg('MSG: zterp -- no data to interpolate to/from')
          call lis_log_msg('MSG: zterp -- beginning and ending data both = 0')
          call lis_log_msg('MSG: zterp -- setting the weights to 0')
          !print*, 'stopping!!!'
          !call endrun
          weight1 = 0
          weight2 = 0
       endif
    endif
  endif
818 continue
  return
```

1.23.17 coszenith (Source File: zterp.F90)

1) Day angle (GAMMA)
2) Solar DEClination
3) Equation of time
4) Local apparent time
5) Hour angle
6) Cosine of zenith angle
All equations come from "An Introduction to Solar Radition" By Muhammad Iqbal, 1983.

INTERFACE:

```
 subroutine coszenith (lon,latd,lhour,zone,julian,czenith)

   implicit none
```

ARGUMENTS:

```
   integer :: zone                ! time zone (1-24) gmt=12
   integer :: julian              ! julian day
   real ::  czenith             ! cosine of zenith angle (radians)
   real ::  dec                 ! solar declination (radians)
```

```
   real ::  et                  ! equation of time (minutes)
   real ::  gamma               ! day angle (radians)
   real ::  latime              ! local apparent time
   real ::  lcorr               ! longitudical correction
   real ::  lhour               ! local standard time
   real ::  lon                 ! local longitude (deg)
   real ::  llat                ! local latitude in radians
   real ::  latd                ! local latitude in degrees
   real ::  ls                  ! standard longitude (deg)
   real ::  omegad              ! omega in degrees
   real ::  omega               ! omega in radians
   real ::  pi                  ! universal constant pi [-]
   real ::  zenith              ! zenith angle(radians)
```

CONTENTS:

```
!-----------------------------------------------------------------------
!     Neither ZENITH nor ZEND are necessary for this program.
!     I originally used them as checks, and left them here in
!     case anyone else had a use for them.
!
!     1)  Day angle GAMMA (radians) page 3
!-----------------------------------------------------------------------
  pi= 3.141592              ! universal constant pi
  gamma=2*pi*(julian-1)/365.
!---------------------------------------------------------------------
!  2) Solar declination (assumed constant for a 24 hour period)  page 7
!  in radians
!---------------------------------------------------------------------
  dec=(0.006918-0.399912*cos(gamma)+0.070257*sin(gamma) &
       -0.006758*cos(2*gamma)+0.000907*sin(2*gamma) &
       -0.002697*cos(3*gamma)+0.00148*sin(3*gamma))
!---------------------------------------------------------------------
!  maximum error 0.0006 rad (<3'), leads to error of less than 1/2 degree
!  in ZENITH angle
!   3)  Equation of time  page 11
!---------------------------------------------------------------------

  et=(0.000075+0.001868*cos(gamma)-0.032077*sin(gamma) &
       -0.014615*cos(2*gamma)-0.04089*sin(2*gamma))*229.18
!---------------------------------------------------------------------
!     4) Local apparent time  page 13
!
!     LS     standard longitude (nearest 15 degree meridian)
!     LON     local longitude
!     LHOUR  local standard time
!     LATIME local apparent time
!     LCORR  longitudunal correction (minutes)
!---------------------------------------------------------------------
```

```
  ls=((zone-1)*15)-180.
  lcorr=4.*(ls-lon)*(-1)
  latime=lhour+lcorr/60.+et/60.
  if (latime.lt.0.) latime=latime+24
  if (latime.gt.24.) latime=latime-24
!-----------------------------------------------------------------------
!     5) Hour angle OMEGA  page 15
!
!     hour angle is zero at noon, postive in the morning
!     It ranges from 180 to -180
!
!     OMEGAD is OMEGA in degrees, OMEGA is in radians
!-----------------------------------------------------------------------
  OMEGAD=(LATime-12.)*(-15.)
  OMEGA=OMEGAD*PI/180.
!-----------------------------------------------------------------------
!     6)  Zenith angle page 15
!
!     CZENITH cosine of zenith angle (radians)
!     LATD=local latitude in degrees
!     LLAT=local latitude in radians
!-----------------------------------------------------------------------
  llat=latd*pi/180.
  czenith=sin(dec)*sin(llat)+cos(dec)*cos(llat)*cos(omega)
  czenith=amax1(0.,czenith)
  zenith=asin(czenith)
  return
```

1.23.18 localtime (Source File: zterp.F90)

Calculates the local time based on GMT

INTERFACE:

```
 subroutine localtime (gmt,lon,lhour,zone)
```

ARGUMENTS:

```
   real:: gmt                 ! GMT time (0-23)
   real:: lon                 ! longitude in degrees
   real:: change              ! the change in number of hours between
   real:: lhour               ! local hour (0-23) 0= midnight, 23= 11:00 p.m.

   integer::  i              ! working integer
   integer:: zone            ! time zone (1-24)
```

CONTENTS:

```
!-----------------------------------------------------------------------
!  Determine into which time ZONE (15 degree interval) the
!  longitude falls.
!-----------------------------------------------------------------------
  do i=1,25
     if (lon.lt.(-187.5+(15*i))) then
        zone=i
        if (zone.eq.25) zone=1
        go to 60
     end if
  end do
!-----------------------------------------------------------------------
!     Calculate change (in number of hours) from GMT time to
!     local hour.  Change will be negative for zones < 13 and
!     positive for zones > 13.
!     There is also a correction for LHOUR < 0 and LHOUR > 23
!     to LHOUR between 0 and 23.
!-----------------------------------------------------------------------
60 if (zone.lt.13) then
     change=zone-13
     lhour=gmt+change
  elseif (zone.eq.13) then
     lhour=gmt
  else
     change=zone-13
     lhour=gmt+change
  end if
  if (lhour.lt.0) lhour=lhour+24
  if (lhour.gt.23) lhour=lhour-24
  return
```

1.24 Fortran: Module Interface lsm_pluginMod.F90 (Source File: lsm_pluginMod.F90)

This module contains the definition of the functions used for land surface model initialization, execution, reading and writing of restart files and other relevant land surface model computations, corresponding to each of the LSMs used in LIS.

REVISION HISTORY:

```
    09 Oct 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
 module lsm_pluginMod
```

1.24.1 lsm_plugin (Source File: lsm_pluginMod.F90)

This is a custom-defined plugin point for introducing a new LSM. The interface mandates that the following routines be implemented and registered for each of the LSM that is included in LIS.

Initialization Definition of LSM variables (to be registered using registerlsmini)

Setup Initialization of parameters (to be registered using registerlsmsetup)

DynamicSetup Routines to setup time dependent parameters (to be registered using registerlsmdynsetup)

Run Routines to execute LSM on a single gridcell for single timestep (to be registered using registerlsmrun)

Read restart Routines to read a restart file for an LSM run (to be registered using registerlsmrestart)

Output Routines to write output (to be registered using registerlsmoutput)

Forcing transfer to model tiles Routines to transfer an array of given forcing to model tiles (to be registered using registerlsmf2t)

Write restart Routines to write a restart file (to be registered using registerlsmwrst)

Multiple LSMs can be included as well, each distinguished in the function table registry by the associated LSM index assigned in the card file.

INTERFACE:

```
subroutine lsm_plugin
```

USES:

```
use template_varder, only : template_varder_ini
use noah_varder, only : noah_varder_ini
use clm_varder, only : clm_varder_ini
use vic_varder, only : vic_varder_ini
use atmdrvMod, only : atmdrv
use mos_varder, only : mos_varder_ini
use hyssib_varder, only : hyssib_varder_ini
use ssib_varder, only : ssib_varder_ini
```

CONTENTS:

```
call registerlsmini(0,template_varder_ini)
call registerlsmini(1,noah_varder_ini)
call registerlsmini(2,clm_varder_ini)
call registerlsmini(3,vic_varder_ini)
call registerlsmini(4,mos_varder_ini)
call registerlsmini(5,hyssib_varder_ini)
call registerlsmini(6,ssib_varder_ini)
```

```
call registerlsmsetup(0,template_setup)
call registerlsmsetup(1,noah_setup)
call registerlsmsetup(2,clm2_setup)
call registerlsmsetup(3,vic_setup)
call registerlsmsetup(4, mos_setup)
call registerlsmsetup(5, hyssib_setup)
call registerlsmsetup(6, ssib_setup)

call registerlsmdynsetup(0,template_dynsetup)
call registerlsmdynsetup(1,noah_dynsetup)
call registerlsmdynsetup(2,clm2_dynsetup)
call registerlsmdynsetup(3,vic_dynsetup)
call registerlsmdynsetup(4, mos_dynsetup)
call registerlsmdynsetup(5,hyssib_dynsetup)
call registerlsmdynsetup(6,ssib_dynsetup)

call registerlsmrun(0,template_main)
call registerlsmrun(1,noah_main)
call registerlsmrun(2,driver)
call registerlsmrun(3,vic_main)
call registerlsmrun(4, mos_main)
call registerlsmrun(5, hyssib_main)
call registerlsmrun(6, ssib_main)

call registerlsmrestart(0,templaterst)
call registerlsmrestart(1,noahrst)
call registerlsmrestart(2,clm2_restart)
call registerlsmrestart(3,vic_readrestart)
call registerlsmrestart(4, mosrst)
call registerlsmrestart(5,hyssibrst)
call registerlsmrestart(6,ssibrst)

call registerlsmoutput(0,template_output)
call registerlsmoutput(1,noah_output)
call registerlsmoutput(2,clm2_output)
call registerlsmoutput(3,vic_output)
call registerlsmoutput(4, mos_output)
call registerlsmoutput(5,hyssib_output)
call registerlsmoutput(6,ssib_output)

call registerlsmf2t(0,template_f2t)
call registerlsmf2t(1,noah_f2t)
call registerlsmf2t(2,atmdrv)
call registerlsmf2t(3,vic_atmdrv)
call registerlsmf2t(4, mos_f2t)
call registerlsmf2t(5,hyssib_f2t)
call registerlsmf2t(6,ssib_f2t)
```

```
call registerlsmwrst(0,template_writerst)
call registerlsmwrst(1,noah_writerst)
call registerlsmwrst(2,clm2wrst)
call registerlsmwrst(3,vic_writerestart)
call registerlsmwrst(4, mos_writerst)
call registerlsmwrst(5,hyssib_writerst)
call registerlsmwrst(6,ssib_writerst)
```

1.25 Fortran: Module Interface baseforcing_pluginMod.F90 (Source File: baseforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module baseforcing_pluginMod
```

1.25.1 baseforcing_plugin (Source File: baseforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine baseforcing_plugin
```

USES:

```
     use geosdomain_module
     use gdasdomain_module
     use ecmwfdomain_module
     use nldasdomain_module
      use gswpdomain_module
     use bergdomain_module
```

CONTENTS:

```
    call registerget(1,getgdas)
    call registerget(2,getgeos)
    call registerget(3,getecmwf)
    call registerget(4,getnldas)
!    call registerget(5,getgswp)
    call registerget(6,getberg)

    call registerdefnat(1,defnatgdas)
    call registerdefnat(2,defnatgeos)
    call registerdefnat(3,defnatecmwf)
    call registerdefnat(4,defnatnldas)
!    call registerdefnat(5,defnatgswp)
    call registerdefnat(6,defnatberg)

    call registertimeinterp(1,time_interp_gdas)
    call registertimeinterp(2,time_interp_geos)
    call registertimeinterp(3,time_interp_ecmwf)
    call registertimeinterp(4,time_interp_nldas)
!    call registertimeinterp(5,time_interp_gswp)
    call registertimeinterp(6,time_interp_berg)
```

1.26 Fortran: Module Interface precipforcing_pluginMod.F90 (Source File: precipforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new observed precipitation forcing scheme.

REVISION HISTORY:

```
    12 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
 module precipforcing_pluginMod
```

1.26.1 precipforcing_plugin (Source File: precipforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new observed precipitation forcing scheme. The interface mandates that the following routines be implemented and registered for each of the forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnatpcp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
   subroutine precipforcing_plugin
```

USES:

```
     use huffdomain_module
     use persdomain_module
     use cmapdomain_module
```

CONTENTS:

```
    call registerpget(2,gethuff)
    call registerpget(3,getpers)
    call registerpget(4,getcmap)

    call registerdefnatpcp(2,defnathuff)
    call registerdefnatpcp(3,defnatpers)
    call registerdefnatpcp(4,defnatcmap)

    call registerpti(2,time_interp_huff)
    call registerpti(3,time_interp_pers)
    call registerpti(4,time_interp_cmap)

  end subroutine precipforcing_plugin
end module precipforcing_pluginMod

\markboth{Left}{Source File: radforcing\_pluginMod.F90,  Date: Fri Feb 11 15:41:03 EST 2005
}

!-------------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
```

```
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-------------------------------------------------------------------------
```

1.27 Fortran: Module Interface radforcing_pluginMod.F90 (Source File: radforcing_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new observed radiation forcing scheme.

REVISION HISTORY:

```
12 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module radforcing_pluginMod
```

1.27.1 radforcing_plugin (Source File: radforcing_pluginMod.F90)

This is a custom-defined plugin point for introducing a new observed radiation forcing scheme. The interface mandates that the following routines be implemented and registered for each of the forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnatrad)

Temporal interpolation Routines to temporally interpolate data (to be registered using registerrti)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine radforcing_plugin
```

USES:

```
use agrmetdomain_module
```

CONTENTS:

```
call registerrget(1,getgrad)
call registerdefnatrad(1,defnatagrmet)
call registerrti(1,time_interp_agrmet)
```

1.28 Fortran: Module Interface domain_pluginMod.F90 (Source File: domain_pluginMod.F90)

This module contains the definition of the functions used for defining routines that initialize various domains.

REVISION HISTORY:

```
17Feb2004;   Sujay Kumar  Initial Specification
```

INTERFACE:

```
module domain_pluginMod
```

1.28.1 domain_plugin (Source File: domain_pluginMod.F90)

This is a custom-defined plugin point for introducing a new domain. The specific computations that needs to be performed to initilaize a new domain/projection need to be defined in this method.

INTERFACE:

```
subroutine domain_plugin
```

USES:

CONTENTS:

```
#if ( defined OPENDAP )
   call registerdomain(1,createtiles_latlon)
   call registerinput(1,readdomain_default)
#else
   call registerdomain(1,createtiles_latlon)
   call registerinput(1,readdomain_default) !gaussian T62
#endif
```

1.28.2 calculate_domveg (Source File: calculate_domveg.F90)

This primary goal of this routine is to determine the percentages of dominant vegetation to create tiles

REVISION HISTORY:

```
09 Sept 2004: Sujay Kumar ; Initial version
```

INTERFACE:

```
subroutine calculate_domveg(fgrd, tsum)
  use lisdrv_module, only : lis
  use spmdMod, only : iam
  implicit none

  real :: fgrd(lis%d%lnc, lis%d%lnr, lis%p%nt)
  real :: tsum(lis%d%lnc, lis%d%lnr)

  integer, allocatable :: pveg(:,:,:)
  integer :: c, r, t, ierr, i, j
  real    :: rsum
  real    :: fvt(lis%p%nt)
  real    :: max
 ------------------------------------------------------------------------
  Exclude tiles with MINA (minimum tile grid area),
  normalize remaining tiles to 100%
 ------------------------------------------------------------------------
  do r=1,lis%d%lnr
     do c=1,lis%d%lnc
        rsum=0.0
        do t=1,lis%p%nt
           if(fgrd(c,r,t).lt.lis%d%mina)then
              fgrd(c,r,t)=0.0
           endif
           rsum=rsum+fgrd(c,r,t)
        enddo
 ------------------------------------------------------------------------
  renormalize veg fractions within a grid to 1
 ------------------------------------------------------------------------
        if(rsum.gt.0.0) then
           do t=1,lis%p%nt
              if(rsum.gt.0.0)fgrd(c,r,t)=fgrd(c,r,t)/rsum
           enddo

           rsum=0.0
           do t=1,lis%p%nt
              rsum=rsum+fgrd(c,r,t)
           enddo
```

```
            if(rsum.lt.0.9999.or.rsum.gt.1.0001)then
               write(*,*) 'Error1 in vegetation tiles',rsum,c,r
            endif
         endif
      enddo
   enddo

   allocate(pveg(lis%d%lnc,lis%d%lnr,lis%p%nt), stat=ierr)
   call check_error(ierr,'Error allocating pveg.',iam)
  ------------------------------------------------------------------------
   Exclude tiles with MAXT (Maximum Tiles per grid),
     normalize remaining tiles to 100%
   Determine the grid predominance order of the tiles
    PVEG(NT) will contain the predominance order of tiles
  ------------------------------------------------------------------------
   do r=1,lis%d%lnr
      do c=1,lis%d%lnc
         do t=1,lis%p%nt
            fvt(t)=fgrd(c,r,t)
            pveg(c,r,t)=0
         enddo
         do i=1,lis%p%nt
            max=0.0
            t=0
            do j=1,lis%p%nt
               if(fvt(j).gt.max)then
                  if(fgrd(c,r,j).gt.0) then
                     max=fvt(j)
                     t=j
                  endif
               endif
            enddo
            if(t.gt.0) then
               pveg(c,r,t)=i
               fvt(t)=-999.0
            endif
         enddo
      enddo
   enddo
  ------------------------------------------------------------------------
   Impose MAXT Cutoff
  ------------------------------------------------------------------------
   do r=1,lis%d%lnr
      do c=1,lis%d%lnc
         rsum=0.0
         do t=1,lis%p%nt
            if(pveg(c,r,t).lt.1) then
               fgrd(c,r,t)=0.0
```

```
               pveg(c,r,t)=0
            endif
            if(pveg(c,r,t).gt.lis%d%maxt) then
               fgrd(c,r,t)=0.0
               pveg(c,r,t)=0
            endif
            rsum=rsum+fgrd(c,r,t)
         enddo
  ----------------------------------------------------------------------
   renormalize veg fractions within a grid to 1
  ----------------------------------------------------------------------
         if(rsum.gt.0.0) then
            do t=1,lis%p%nt
               if(rsum.gt.0.0)fgrd(c,r,t)= fgrd(c,r,t)/rsum
            enddo

            rsum=0.0
            do t=1,lis%p%nt
               rsum=rsum+ fgrd(c,r,t)  !recalculate rsum to check
            enddo
            tsum(c,r)=rsum

            if(rsum.lt.0.9999.or.rsum.gt.1.0001)then  !check renormalization
               write(*,*) 'Error2 in vegetation tiles',rsum,c,r
            endif
         endif
      enddo
   enddo
   deallocate(pveg)
 end subroutine calculate_domveg
\markboth{Left}{Source File: createtiles\_latlon.F90,  Date: Fri Feb 11 15:41:03 EST 2005
}


  ------------------------------------------------------------------------
   NASA Goddard Space Flight Center Land Information System (LIS) V3.0
   Released May 2004
   See SOFTWARE DISTRIBUTION POLICY for software distribution policies
   The LIS source code and documentation are in the public domain,
   available without fee for educational, research, non-commercial and
   commercial purposes.  Users may distribute the binary or source
   code to third parties provided this statement appears on all copies and
   that no charge is made for such copies.
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See COPYRIGHT.TXT for copyright details.
```

```
  ------------------------------------------------------------------------
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection{createtiles\_latlon (Source File: createtiles\_latlon.F90)}



    This primary goal of this routine is to determine tile space for
    a lat/lon domain

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}    1  Oct 1999: Jared Entin; Initial code
    15 Oct 1999: Paul Houser; Major F90 and major structure revision
    3  Jan 2000: Minor T=0 bug fix, should have no effect on output
    8  Mar 2000: Brian Cosgrove; Initialized FGRD to 0 For Dec Alpha Runs
   22  Aug 2000: Brian Cosgrove; Altered code for US/Mexico/Canada Mask
   04  Feb 2001: Jon Gottschalck; Added option to read and use Koster tile space
   17  Oct 2003: Sujay Kumar ; Initial version of subsetting code
```

INTERFACE:

```
 subroutine createtiles_latlon()
```

USES:

```
   use lisdrv_module, only: lis
   use grid_module
   use spmdMod
```

CONTENTS:

```
  if ( masterproc ) then
     if(lis%d%gridDesc(42) > lis%d%lnc .or. &
          lis%d%gridDesc(43) > lis%d%lnr)  then !using a subdomain
        lis%d%gnc = lis%d%gridDesc(42)
        lis%d%gnr = lis%d%gridDesc(43)
     else
        lis%d%gnc = lis%d%lnc
        lis%d%gnr = lis%d%lnr
     endif

     allocate(localmask(lis%d%lnc,lis%d%lnr))
     call readlandmask(lis%d%landcover, localmask)


     allocate(elevdiff(lis%d%lnc, lis%d%lnr), stat=ierr)
```

```
    call check_error(ierr,'Error allocating elev diff.',iam)

    call readelevdiff(lis%d%elev, elevdiff)

    allocate(fgrd(lis%d%lnc,lis%d%lnr,lis%p%nt), stat=ierr)
    call check_error(ierr,'Error allocating fgrd.',iam)

    call readlandcover(lis%d%landcover, fgrd)

    allocate(tsum(lis%d%lnc, lis%d%lnr), stat=ierr)
    call check_error(ierr,'Error allocating tsum.',iam)
    tsum = 0.0

    call calculate_domveg(fgrd, tsum)

    call create_vegtilespace(fgrd, tsum, localmask, elevdiff)

    deallocate(elevdiff)

    deallocate(localmask, stat=ierr)
    call check_error(ierr,'Error allocating glbmask',iam)
    deallocate(fgrd, stat=ierr)
    call check_error(ierr,'Error allocating glbfgrd',iam)
    deallocate(tsum, stat=ierr)
    call check_error(ierr,'Error allocating glbtsum.',iam)

    write(*,*) 'msg: maketiles -- actual number of tiles:', &
         lis%d%glbnch,' (',iam,')'
    write(*,*)

    write(*,*) 'msg: maketiles -- size of grid dimension:', &
         lis%d%glbngrid,' (',iam,')'

  endif
  print*,'MSG: maketiles -- done',' (',iam,')'
  return
```

1.28.3 create_vegtilespace (Source File: create_vegtilespace.F90)

This primary goal of this routine is to determine the tiles based on dominant vegetaiton

REVISION HISTORY:

```
09 Sept 2004: Sujay Kumar ; Initial version
```

INTERFACE:

```
subroutine create_vegtilespace(fgrd, tsum, localmask, elevdiff)
  use lisdrv_module, only : lis, grid, tile, glbgindex
  use spmdMod, only : iam
  implicit none

  real :: fgrd(lis%d%lnc, lis%d%lnr, lis%p%nt)
  real :: tsum(lis%d%lnc, lis%d%lnr)
  real :: localmask(lis%d%lnc, lis%d%lnr)
  real :: elevdiff(lis%d%lnc, lis%d%lnr)
  real,allocatable :: lat(:,:)
  real,allocatable :: lon(:,:)
  real, allocatable :: domveg(:,:)
  real :: locallat, locallon
  integer :: landnveg
  integer :: c, r, t, cindex, rindex
  integer :: gnc, gnr
  integer :: count, ierr

  if(lis%d%gridDesc(9) .ne.0.01) then
     if(lis%d%gridDesc(42) > lis%d%lnc .or. &
          lis%d%gridDesc(43) > lis%d%lnr)  then !using a subdomain
        gnc = lis%d%gridDesc(42)
        gnr = lis%d%gridDesc(43)
     else
        gnc = lis%d%lnc
        gnr = lis%d%lnr
     endif
     lis%d%gnc = gnc
     lis%d%gnr = gnr
  else
     gnc = lis%d%lnc
     gnr = lis%d%lnr
  endif
  allocate(lat(gnc,gnr), stat=ierr)
  call check_error(ierr,'Error allocating lat.',iam)

  allocate(lon(gnc,gnr), stat=ierr)
  call check_error(ierr,'Error allocating lon.',iam)

  do r=1,gnr
     do c=1,gnc
        lat(c,r) = lis%d%gridDesc(44)+(r-1)*lis%d%gridDesc(49)
        lon(c,r) = lis%d%gridDesc(45)+(c-1)*lis%d%gridDesc(50)
     enddo
  enddo

  landnveg = 5
  lis%d%glbnch=0
```

```
   do t=1,lis%p%nt
      do r=1,lis%d%lnr
         do c=1,lis%d%lnc
            if(localmask(c,r).gt.0.99.and. &
                 localmask(c,r).lt.3.01)then !we have land
               if(fgrd(c,r,t).gt.0.0)then
                  lis%d%glbnch=lis%d%glbnch+1
               endif
               if(tsum(c,r).eq.0.0.and.t.eq.landnveg)then
                  lis%d%glbnch=lis%d%glbnch+1
               endif
            endif
         enddo
      enddo
   enddo

print*, 'DBG: maketiles -- glbnch',lis%d%glbnch,' (',iam,')'
allocate(tile(lis%d%glbnch))

lis%d%glbngrid=0
do r=1,lis%d%lnr
   do c=1,lis%d%lnc
      if(localmask(c,r).gt.0.99 .and. &
           localmask(c,r).lt.3.01) then
         lis%d%glbngrid=lis%d%glbngrid+1
      endif
   enddo
enddo
count = 1
print*, 'DBG: maketiles1 -- glbnch',lis%d%glbnch,' (',iam,')'
allocate(grid(lis%d%glbngrid))
allocate(glbgindex(lis%d%lnc, lis%d%lnr))
print*, 'DBG: maketiles2 -- glbnch',lis%d%glbnch,' (',iam,')'
do r=1,lis%d%lnr
   do c=1,lis%d%lnc
      glbgindex(c,r) = -1
      if(localmask(c,r).gt.0.99 .and. &
           localmask(c,r).lt.3.01) then
         locallat = lis%d%gridDesc(4)+(r-1)*lis%d%gridDesc(9)
         locallon = lis%d%gridDesc(5)+(c-1)*lis%d%gridDesc(10)
         grid(count)%lat = locallat
         grid(count)%lon = locallon
         grid(count)%fgrd = fgrd(c,r,:)
         glbgindex(c,r) = count
         count = count+1
      endif
   enddo
```

```
 enddo
 print*, 'DBG: maketiles3 -- glbnch',lis%d%glbnch,' (',iam,')'
-----------------------------------------------------------------
   For writing dominant Vegetation types
-----------------------------------------------------------------
 if(lis%o%wparam .eq.1) then
    allocate(domveg(lis%d%lnc,lis%d%lnr))
    domveg = 0
 endif
 count = 0
 if(lis%d%gridDesc(9) .ne. 0.01 ) then
    do r=1,gnr
       do c=1,gnc
          if(lat(c,r).ge.lis%d%gridDesc(4).and. &
               lat(c,r).le.lis%d%gridDesc(7).and. &
               lon(c,r).ge.lis%d%gridDesc(5).and. &
               lon(c,r).le.lis%d%gridDesc(8)) then
             rindex = r - nint((lis%d%gridDesc(4)-lis%d%lc_gridDesc(1)) &
                  /lis%d%gridDesc(9))
             cindex = c - nint((lis%d%gridDesc(5)-lis%d%lc_gridDesc(2)) &
                  /lis%d%gridDesc(10))
             do t=1,lis%p%nt
                if(localmask(cindex,rindex).gt.0.99.and. &
                     localmask(cindex,rindex).lt.3.01)then
                   if(fgrd(cindex,rindex,t).gt.0.0)then
                      count = count+1

                      tile(count)%row=rindex
                      tile(count)%col=cindex
                      tile(count)%index = glbgindex(cindex,rindex)
                      tile(count)%vegt=t
                      if(lis%o%wparam.eq.1) then
                         domveg(cindex,rindex) = t*1.0
                      endif
                      tile(count)%fgrd=fgrd(cindex,rindex,t)
                      if(lis%f%ecor.eq.1) then
                         if(elevdiff(cindex,rindex).eq.-9999.0) then
                            elevdiff(cindex,rindex) = 0.0
                         endif
                          tile(count)%elev=elevdiff(cindex,rindex)
                         grid(tile(count)%index)%elev=elevdiff(cindex,rindex)
                      endif
                   endif
-------------------------------------------------------------------
 What if we we have land without vegetation assigned
-------------------------------------------------------------------
                   if(tsum(cindex,rindex).eq.0.0.and.t.eq.landnveg)then
                      count=count+1
```

```
                      tile(count)%row=rindex
                      tile(count)%col=cindex
                      tile(count)%index = glbgindex(cindex,rindex)
                      tile(count)%vegt=t
                      if(lis%o%wparam.eq.1) then
                         domveg(cindex,rindex) = t*1.0
                      endif
                      tile(count)%fgrd=1.0
                      if(lis%f%ecor.eq.1) then
                         if(elevdiff(c,r).eq.-9999.0) &
                              elevdiff(c,r) = 0.0
                          tile(count)%elev=elevdiff(cindex,rindex)
                         grid(tile(count)%index)%elev=elevdiff(cindex,rindex)
                      endif
                   endif
                endif
             enddo
          endif
       enddo
    enddo
 else
    do r=1,lis%d%lnr
        do c=1,lis%d%lnc
           do t=1,lis%p%nt
              if(localmask(c,r).gt.0.0)then
                 if(fgrd(c,r,t).gt.0.0)then
                    count = count+1
                    tile(count)%row=r
                    tile(count)%col=c
                    tile(count)%index = glbgindex(c,r)
                    tile(count)%vegt=t
                    if(lis%o%wparam.eq.1) then
                       domveg(c,r) = t*1.0
                    endif
                    tile(count)%fgrd=fgrd(c,r,t)
                    if(lis%f%ecor.eq.1) then
                       if(elevdiff(c,r).eq.-9999.0) &
                            elevdiff(c,r) = 0.0
                        tile(count)%elev = elevdiff(c,r)
                       grid(tile(count)%index)%elev=elevdiff(c,r)
                      endif
                 endif
 ----------------------------------------------------------------------
  What if we we have land without vegetation assigned
 ----------------------------------------------------------------------
                 if(tsum(c,r).eq.0.0.and.t.eq.landnveg)then
                    count=count+1
                    tile(count)%row=r
```

```
                      tile(count)%col=c
                      tile(count)%index = glbgindex(c,r)
                      tile(count)%vegt=t
                      if(lis%o%wparam.eq.1) then
                         domveg(c,r) = t*1.0
                      endif
                      tile(count)%fgrd=1.0
                      if(lis%f%ecor.eq.1) then
                         if(elevdiff(c,r).eq.-9999.0) &
                              elevdiff(c,r) = 0.0
                            tile(count)%elev = elevdiff(c,r)
                         grid(tile(count)%index)%elev=elevdiff(c,r)
                        endif
                   endif
                endif
             enddo
          enddo
       enddo
    endif
   if(lis%o%wparam.eq.1) then
      open(32,file="domvegtype.bin",form='unformatted')
      write(32) domveg
      close(32)
      deallocate(domveg)
   endif
   deallocate(lat)
   deallocate(lon)
 end subroutine create_vegtilespace
```

\markboth{Left}{Source File: readdomain_default.F90, Date: Fri Feb 11 15:41:03 EST 2005
}

```
  ---------------------------------------------------------------------------
   NASA Goddard Space Flight Center Land Information System (LIS) V3.0
   Released May 2004
   See SOFTWARE DISTRIBUTION POLICY for software distribution policies
   The LIS source code and documentation are in the public domain,
   available without fee for educational, research, non-commercial and
   commercial purposes.  Users may distribute the binary or source
   code to third parties provided this statement appears on all copies and
   that no charge is made for such copies.
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
```

```
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See COPYRIGHT.TXT for copyright details.
  -------------------------------------------------------------------------
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection{readdomain\_default (Source File: readdomain\_default.F90)}




    Reads in LIS run specifics from lis.crd

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}
   REVISION HISTORY:
    15 Oct 1999: Paul Houser; Initial code
    4  Apr 2000: Jeffrey Walker; Added catchment model output interval
    11 Apr 2000: Brian Cosgrove; Added Elevation correction and Forcing
                 Mask read statements
    6  Jun 2000: Jon Radakovich; Updated for new version of CLM
    23 Feb 2001: Urszula Jambor; Added GEOS or GDAS forcing option
    27 Mar 2001: Jon Gottschalck; Revision of subroutine by implementing namelists
    05 Sep 2001: Brian Cosgrove; Altered forcing logfile output to include
                 more precip types
    04 Feb 2002: Jon Gottschalck; Added section to set to Koster tilespace files if necessary
    15 Apr 2002: Urszula Jambor; Added ECMWF forcing options, also
                 adding 1 & 1/2 degree GLDAS domain options.
    28 Apr 2002: Kristi Arsenault; Added NOAH LSM code
    14 Nov 2003: Sujay Kumar; Modified card file that includes regional
                 modeling options
```

INTERFACE:

```
 subroutine readdomain_default
```

USES:

```
   use lisdrv_module, only : lis
   implicit none
```

CONTENTS:

```
  open(10,file='lis.crd',form='formatted',status='old')

  read(unit=10,NML=run_domain)
  read(unit=10,NML=param_domain)
  print*,'DOMAIN details:'
```

```
!-----------------------------------------------------------------------
! Read namelist of parameters depending on the domain
!-----------------------------------------------------------------------
  lis%d%gridDesc(1) = run_dd(1)
  lis%d%gridDesc(4) = run_dd(2)
  lis%d%gridDesc(5) = run_dd(3)
  lis%d%gridDesc(7) = run_dd(4)
  lis%d%gridDesc(8) = run_dd(5)
  lis%d%gridDesc(9) = run_dd(6)

  lis%d%gridDesc(44) = param_dd(1)
  lis%d%gridDesc(45) = param_dd(2)
  lis%d%gridDesc(47) = param_dd(3)
  if(lis%d%gridDesc(1).eq.0) then
     lis%d%gridDesc(10) = run_dd(7)
     lis%d%gridDesc(50) = param_dd(5)
  elseif(lis%d%gridDesc(1) .eq. 4 ) then
!     lis%d%gridDesc(10) = dd(7)
!     lis%d%gridDesc(50) = dd(13)
  endif
  lis%d%gridDesc(48) = param_dd(4)
  lis%d%gridDesc(49) = param_dd(6)
  if(lis%d%gridDesc(1).eq.0) then
     lis%d%gridDesc(43) = nint((lis%d%gridDesc(47)-lis%d%gridDesc(44))/lis%d%gridDesc(49)) + 1
     lis%d%gridDesc(42) = nint((lis%d%gridDesc(48)-lis%d%gridDesc(45))/lis%d%gridDesc(50)) + 1
  elseif(lis%d%gridDesc(1).eq.4) then
!     lis%d%gridDesc(43) = 2*lis%d%gridDesc(50)
!     lis%d%gridDesc(42) = nint(360/dd(12))
  endif


  if(lis%d%gridDesc(1).eq.0) then
     lis%d%gridDesc(6) = 128
     lis%d%gridDesc(11) = 64
     lis%d%gridDesc(20) = 255

  if(lis%d%gridDesc(7).le.lis%d%gridDesc(4)) then
     print*, 'lat2 must be greater than lat1'
     print*, 'Stopping run...'
     call endrun
  endif
  if(lis%d%gridDesc(8).le.lis%d%gridDesc(5)) then
     print*, 'lon2 must be greater than lon1'
     print*, 'Stopping run...'
     call endrun
  endif
#if 0
  if(mod(lis%d%gridDesc(4)-lis%d%gridDesc(44),lis%d%gridDesc(9)).ne.0) then
```

```
      tl = (nint((lis%d%gridDesc(4)-lis%d%gridDesc(44))/lis%d%gridDesc(9)))
      lis%d%gridDesc(4) = lis%d%gridDesc(44)+tl*lis%d%gridDesc(9)
      print*, 'modified gridDesc(4) ',lis%d%gridDesc(4)
   endif
   if(mod(lis%d%gridDesc(5)-lis%d%gridDesc(45),lis%d%gridDesc(10)).ne.0) then
      tl = (nint((lis%d%gridDesc(5)-lis%d%gridDesc(45))/lis%d%gridDesc(10)))
      lis%d%gridDesc(5) = lis%d%gridDesc(45)+tl*lis%d%gridDesc(10)
      print*, 'modified gridDesc(5) ',lis%d%gridDesc(5)
   endif

   if(mod(lis%d%gridDesc(47)-lis%d%gridDesc(7),lis%d%gridDesc(9)).ne.0) then
      tl = (int((lis%d%gridDesc(47)-lis%d%gridDesc(7))/lis%d%gridDesc(9)))
      lis%d%gridDesc(7) = lis%d%gridDesc(47)-tl*lis%d%gridDesc(9)
      print*, 'modified gridDesc(7) ',lis%d%gridDesc(7)
   endif
   if(mod(lis%d%gridDesc(48)-lis%d%gridDesc(8),lis%d%gridDesc(10)).ne.0) then
      tl = (int((lis%d%gridDesc(48)-lis%d%gridDesc(8))/lis%d%gridDesc(10)))
      lis%d%gridDesc(8) = lis%d%gridDesc(48)-tl*lis%d%gridDesc(10)
      print*, 'modified gridDesc(8) ',lis%d%gridDesc(8)
   endif
#endif
   lis%d%gridDesc(2) = nint((lis%d%gridDesc(8)-lis%d%gridDesc(5))/lis%d%gridDesc(10))+ 1
   lis%d%gridDesc(3) = nint((lis%d%gridDesc(7)-lis%d%gridDesc(4))/lis%d%gridDesc(9)) + 1
   elseif(lis%d%gridDesc(1).eq.4) then
!     lis%d%gridDesc(6) = 128
!     lis%d%gridDesc(11) = 64
!     lis%d%gridDesc(20) = 255
!     lis%d%gridDesc(3) = 2*lis%d%gridDesc(10)
!     lis%d%gridDesc(2) = nint(360/dd(6))
   endif
   do k=1,13
      print*, '(',k,',',lis%d%gridDesc(k),')'
   enddo
   do k=40,50
      print*, '(',k,',',lis%d%gridDesc(k),')'
   enddo

   lis%d%lnc = lis%d%gridDesc(2)
   lis%d%lnr = lis%d%gridDesc(3)
   print*,'running domain','(',lis%d%lnc,lis%d%lnr,')'
   print*,'parameter domain','(',lis%d%gridDesc(42),lis%d%gridDesc(43),')'
   close(10)
   return
```

1.29 Fortran: Module Interface elevdiff_pluginMod.F90 (Source File: elevdiff_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module elevdiff_pluginMod
```

1.29.1 elevdiff_plugin (Source File: elevdiff_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine elevdiff_plugin

  external read_elevdiff_gtopo30
```

USES:

```
  call registerreadelevdiff(1,read_elevdiff_gtopo30)
```

1.30 Fortran: Module Interface lai_pluginMod.F90 (Source File: lai_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module lai_pluginMod
```

1.30.1 lai_plugin (Source File: lai_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine lai_plugin
  external read_avhrrlai, read_avhrrsai
```

USES:

```
  call registerreadlai(2,read_avhrrlai)
  call registerreadsai(2,read_avhrrsai)
```

1.31 Fortran: Module Interface landcover_pluginMod.F90 (Source File: landcover_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module landcover_pluginMod
```

1.31.1 landcover_plugin (Source File: landcover_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine landcover_plugin

  external read_umdavhrr_mask
  external read_umdavhrr_lc
```

USES:

```
  call registerreadlc(1,read_umdavhrr_lc)
  call registerreadmask(1,read_umdavhrr_mask)
```

1.32 Fortran: Module Interface soils_pluginMod.F90 (Source File: soils_pluginMod.F90)

This module contains the definition of the functions used for incorporating a new model forcing scheme.

REVISION HISTORY:

```
11 Dec 03    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module soils_pluginMod
```

1.32.1 soils_plugin (Source File: soils_pluginMod.F90)

This is a custom-defined plugin point for introducing a new forcing scheme. The interface mandates that the following routines be implemented and registered for each model forcing scheme.

retrieval of forcing data Routines to retrieve forcing data and to interpolate them. (to be registered using registerget)

definition of native domain Routines to define the native domain as a kgds array (to be registered using registerdefnat)

temporal interpolation Interpolate forcing data temporally. (to be registered using registertimeinterp)

Multiple forcing schemes can be included as well, each distinguished in the function table registry by the associated forcing index assigned in the card file.

INTERFACE:

```
subroutine soils_plugin
  external read_faosand, read_faoclay, read_faosilt
  external read_statsgosand, read_statsgoclay, read_statsgosilt
```

USES:

```
  call registerreadsand(2,read_faosand)
  call registerreadclay(2,read_faoclay)
  call registerreadsilt(2,read_faosilt)

  call registerreadsand(3,read_statsgosand)
  call registerreadclay(3,read_statsgoclay)
  call registerreadsilt(3,read_statsgosilt)
```

1.32.2 read_color (Source File: read_color.F90)

This subroutine retrieves soil color data

REVISION HISTORY:

```
03 Jan 2005: James Geiger; Initial Specification
```

INTERFACE:

```
subroutine read_color(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.3 read_faosoils (Source File: read_faoclay.F90)

This subroutine retrieves FAO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_faoclay(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.4 read_faosoils (Source File: read_faosand.F90)

This subroutine retrieves FAO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_faosand(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.5 read_faosoils (Source File: read_faosilt.F90)

This subroutine retrieves FAO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_faosilt(array)
```

USES:

```
use lisdrv_module, only : lis
use lis_openfileMod
use lis_indices_module
```

1.32.6 read_statsgosoils (Source File: read_statsgoclay.F90)

This subroutine retrieves STATSGO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_statsgoclay(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.7 read_statsgosoils (Source File: read_statsgosand.F90)

This subroutine retrieves STATSGO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_statsgosand(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.8 read_statsgosoils (Source File: read_statsgosilt.F90)

This subroutine retrieves STATSGO soils data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_statsgosilt(array)
```

USES:

```
use lisdrv_module, only : lis, tile
use lis_openfileMod
use lis_indices_module
```

1.32.9 read_umdavhrr_lc (Source File: read_umdavhrr_lc.F90)

This subroutine retrieves UMD-AVHRR landcover data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_umdavhrr_lc(fgrd)
```

USES:

```
use lisdrv_module, only : lis
use spmdMod
```

1.32.10 read_umdavhrr_mask (Source File: read_umdavhrr_mask.F90)

This subroutine retrieves UMD-AVHRR landcover data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_umdavhrr_mask(localmask)
```

USES:

```
use lisdrv_module, only : lis
use spmdMod
```

1.32.11 read_elevdiff_gtopo30 (Source File: read_elevdiff_gtopo30.F90)

This subroutine retrieves UMD-AVHRR landcover data

REVISION HISTORY:

```
03 Sept 2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine read_elevdiff_gtopo30(elevdiff)
```

USES:

```
use lisdrv_module, only : lis
use lis_openfileMod
use spmdMod

implicit none

integer :: line1, line2
integer :: nc_dom, ierr
real    :: elevdiff(lis%d%lnc, lis%d%lnr)
integer :: c, r, glnc, glnr, line
```

1.32.12 climatologylairead.F90 (Source File: climatologylairead.F90)

This program reads in AVHRR LAI data for CLM

REVISION HISTORY:

```
27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data
```

INTERFACE:

```
subroutine climatologylairead(name11, name12, lai_t1_f, lai_t2_f)
```

USES:

```
use time_manager
use lisdrv_module, only : grid,lis,tile
use lis_indices_module, only : lis_grid_offset
use spmdMod, only : iam
use precision
```

1.32.13 climatologysairead.F90 (Source File: climatologysairead.F90)

This program reads in AVHRR LAI data for CLM

REVISION HISTORY:

```
27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data
```

INTERFACE:

```
subroutine climatologysairead(name15,name16,sai_t1_f,sai_t2_f)
```

USES:

```
use time_manager
use lisdrv_module, only : grid,lis,tile
use lis_indices_module, only : lis_grid_offset
use spmdMod, only : iam
use precision
```

1.32.14 read_avhrrlai (Source File: read_avhrrlai.F90)

This program reads in AVHRR LAI data

REVISION HISTORY:

```
27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
10 Sept 2004: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine read_avhrrlai(lai1, lai2, wt1, wt2)
```

USES:

```
use time_manager
use lisdrv_module, only : grid,lis
use filename_mod
use precision
```

CONTENTS:

```
!-------------------------------------------------------------------------
! Determine current time to find correct LAI files
!-------------------------------------------------------------------------
  if (lis%t%tscount .eq. 0) then
   lis%t%yr = lis%t%syr
   lis%t%mo = lis%t%smo
   lis%t%da = lis%t%sda
   lis%t%mn = lis%t%smn
   lis%t%ss = lis%t%sss
  else
   lis%t%yr = lis%t%yr
   lis%t%mo = lis%t%mo
   lis%t%da = lis%t%da
   lis%t%mn = lis%t%mn
   lis%t%ss = lis%t%ss
  endif

  call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr, &
```

```
       lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
!-------------------------------------------------------------------------
! Initialize LAI flag varaible
!-------------------------------------------------------------------------
   lis%p%laiflag = 0

   zeroi=0
   numi=16
!-------------------------------------------------------------------------
! Determine Monthly data Times (Assume Monthly
! value valid at DA=16 HR=00Z)
!-------------------------------------------------------------------------
   if (lis%t%da .lt. 16) then
      mo1 = lis%t%mo-1
      yr1 = lis%t%yr
      if (mo1 .eq. 0) then
         mo1 = 12
         yr1 = lis%t%yr - 1
      endif
      mo2 = lis%t%mo
      yr2 = lis%t%yr
   else
      mo1 = lis%t%mo
      yr1 = lis%t%yr
      mo2 = lis%t%mo+1
      yr2 = lis%t%yr
      if (mo2 .eq. 13) then
         mo2 = 1
         yr2 = lis%t%yr + 1
      endif
   endif

   call date2time(time1,doy1,gmt1,yr1,mo1,numi,zeroi,zeroi,zeroi)
   call date2time(time2,doy2,gmt2,yr2,mo2,numi,zeroi,zeroi,zeroi)
!-------------------------------------------------------------------------
! Check to see if need new LAI data
!-------------------------------------------------------------------------
   if (time2 .gt. lis%p%laitime) then
      lis%p%laiflag = 1
   else
      lis%p%laiflag = 0
   endif

   avhrrdir = lis%p%avhrrdir
!-------------------------------------------------------------------------
! Determine weights between months
!-------------------------------------------------------------------------
   wt1 = (time2-lis%t%time)/(time2-time1)
```

```
   wt2 = (lis%t%time-time1)/(time2-time1)

!-----------------------------------------------------------------------
! Get new LAI data if required
!-----------------------------------------------------------------------
  if (lis%p%laiflag .eq. 1) then
     print*, 'in avhrr lai read.'
     write(unit=temp,fmt='(i4,i2.2)') yr1, mo1
     read (unit=temp,fmt='(a4,a2)') cyr1, cmo1
     write(unit=temp,fmt='(i4,i2.2)') yr2,  mo2
     read (unit=temp,fmt='(a4,a2)') cyr2, cmo2

   lis%p%laitime = time2
   if(lis%d%gridDesc(9).eq.0.01) then
      domain = 8
   else if(lis%d%gridDesc(9).eq.0.05) then
      domain = 7
   endif
   if(domain ==8)  then
      call avhrr_laifile_1km(name9,name10,name11,name12,&
           lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
   else if(domain ==7)  then
      call avhrr_laifile_5km( &
           name9,name10,name11,name12,&
           lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
   else
      call avhrr_laifilename(name9,name10,name11,name12, &
           lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
   endif
!-----------------------------------------------------------------------
! Open AVHRR LAI files (assumes realtime monthly files are present first
! then uses climatology files)
! Assume realtime monthly files are present as default
!-----------------------------------------------------------------------
   print*, 'name9  ',name9
   print*, 'name10 ',name10
   inquire(file=name9,exist=laifile1)
   inquire(file=name10,exist=laifile2)
   call climatologylairead(name11, name12, lai1, lai2)
end if
end subroutine read_avhrrlai

\markboth{Left}{Source File: read\_avhrrsai.F90,  Date: Fri Feb 11 15:41:03 EST 2005
}

!-----------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
```

```
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------
```

1.32.15 clmlairead.F90: (Source File: read_avhrrsai.F90)

This program reads in AVHRR LAI data for CLM

REVISION HISTORY:

```
27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data
```

INTERFACE:

```
subroutine read_avhrrsai(sai1, sai2, wt1, wt2)
```

USES:

```
use time_manager
use lisdrv_module, only : grid,lis
use filename_mod
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Determine current time to find correct LAI files
!-----------------------------------------------------------------------
  if (lis%t%tscount .eq. 0) then
   lis%t%yr = lis%t%syr
   lis%t%mo = lis%t%smo
   lis%t%da = lis%t%sda
```

```
   lis%t%mn = lis%t%smn
   lis%t%ss = lis%t%sss
  else
   lis%t%yr = lis%t%yr
   lis%t%mo = lis%t%mo
   lis%t%da = lis%t%da
   lis%t%mn = lis%t%mn
   lis%t%ss = lis%t%ss
  endif

  call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr, &
       lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
!-----------------------------------------------------------------------
! Initialize LAI flag varaible
!-----------------------------------------------------------------------
   lis%p%saiflag = 0

   zeroi=0
   numi=16
!-----------------------------------------------------------------------
! Determine Monthly data Times (Assume Monthly
! value valid at DA=16 HR=00Z)
!-----------------------------------------------------------------------
   if (lis%t%da .lt. 16) then
      mo1 = lis%t%mo-1
      yr1 = lis%t%yr
      if (mo1 .eq. 0) then
         mo1 = 12
         yr1 = lis%t%yr - 1
      endif
      mo2 = lis%t%mo
      yr2 = lis%t%yr
   else
      mo1 = lis%t%mo
      yr1 = lis%t%yr
      mo2 = lis%t%mo+1
      yr2 = lis%t%yr
      if (mo2 .eq. 13) then
         mo2 = 1
         yr2 = lis%t%yr + 1
      endif
   endif

   call date2time(time1,doy1,gmt1,yr1,mo1,numi,zeroi,zeroi,zeroi)
   call date2time(time2,doy2,gmt2,yr2,mo2,numi,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
! Check to see if need new LAI data
!-----------------------------------------------------------------------
```

```
!<kluge sai read>
!   if (time2 .gt. lis%p%laitime) then
   if (time2 .gt. lis%p%saitime) then
!</kluge sai read>
      lis%p%saiflag = 1
   else
      lis%p%saiflag = 0
   endif

   avhrrdir = lis%p%avhrrdir
!-----------------------------------------------------------------------
! Determine weights between months
!-----------------------------------------------------------------------
   wt1 = (time2-lis%t%time)/(time2-time1)
   wt2 = (lis%t%time-time1)/(time2-time1)

!-----------------------------------------------------------------------
! Get new LAI data if required
!-----------------------------------------------------------------------
  if (lis%p%saiflag .eq. 1) then
     print*, 'in avhrr lai read.'
     write(unit=temp,fmt='(i4,i2.2)') yr1, mo1
     read (unit=temp,fmt='(a4,a2)') cyr1, cmo1
     write(unit=temp,fmt='(i4,i2.2)') yr2,  mo2
     read (unit=temp,fmt='(a4,a2)') cyr2, cmo2

!<kluge sai read>
!   lis%p%laitime = time2
   lis%p%saitime = time2
!</kluge sai read>
   if(lis%d%gridDesc(9).eq.0.01) then
      domain = 8
   else if(lis%d%gridDesc(9).eq.0.05) then
      domain = 7
   endif
   if(domain ==8)  then
         call avhrr_saifile_1km(name13,&
              name14,name15,name16, &
              lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
   else if(domain ==7)  then
         call avhrr_saifile_5km( &
                   name13,&
                   name14,name15,name16, &
                   lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
   else
         call avhrr_saifilename(name13,&
              name14,name15,name16, &
              lis%p%avhrrdir,cyr1,cyr2,cmo1,cmo2)
```

```
      endif
!-----------------------------------------------------------------------
! Open AVHRR LAI files (assumes realtime monthly files are present first
! then uses climatology files)
! Assume realtime monthly files are present as default
!-----------------------------------------------------------------------

   call climatologysairead(name15,name16,sai1,sai2)

end if
end subroutine read_avhrrsai

\markboth{Left}{Source File: bilinear\_interp.F90,  Date: Fri Feb 11 15:41:03 EST 2005
}

!-----------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------
```

1.32.16 bilinear_interp.F90 (Source File: bilinear_interp.F90)

This subprogram performs bilinear interpolation from any grid to any grid for scalar fields. The routine is based on the spatial interpolation package ipolates from NCEP.
The algorithm simply computes (weighted) averages of bilinearly interpolated points arranged in a square box centered around each output grid point and stretching nearly halfway to each of the neighboring grid points. options allow choices of number of points in each radius from the center point (ipopt(1)) which defaults to 2 (if ipopt(1)=-1) meaning that 25 points will be averaged; further options are the respective weights for the radius points

starting at the center point (ipopt(2:2+ipopt(1)) which defaults to all 1 (if ipopt(2)=-1.). only horizontal interpolation is performed. the grids are defined by their grid description sections
The grid description arrays are based on the decoding schemes used by NCEP. However, in order to remove the integer arithmetic employed in the original ipolates, the routines are rewritten using real number manipulations. The general structure remains the same.
The current code recognizes the following projections: (gridDesc(1)=0) equidistant cylindrical (gridDesc(1)=1) mercator cylindrical (gridDesc(1)=3) lambert conformal conical (gridDesc(1)=4) gaussian cylindrical (spectral native) (gridDesc(1)=5) polar stereographic azimuthal (gridDesc(1)=202) rotated equidistant cylindrical (eta native) where gridDesc could be either input gridDesci or output gridDesco. as an added bonus the number of output grid points and their latitudes and longitudes are also returned. input bitmaps will be interpolated to output bitmaps. output bitmaps will also be created when the output grid extends outside of the domain of the input grid. the output field is set to 0 where the output bitmap is off. INPUT ARGUMENT LIST: ipopt - integer (20) interpolation options ipopt(1) is number of radius points (defaults to 2 if ipopt(1)=-1); ipopt(2:2+ipopt(1)) are respective weights (defaults to all 1 if ipopt(2)=-1). gridDesci - real(200) input domain description parameters gridDesco - integer (200) output domain description parameters mi - integer dimension of input grid fields mo - integer dimension of output grid fields ibi - integer input bitmap flags li - logical*1 (mi) input bitmaps (if some ibi(k)=1) gi - real (mi) input fields to interpolate
OUTPUT ARGUMENT LIST: no - integer number of output points rlat - real (mo) output latitudes in degrees rlon - real (mo) output longitudes in degrees ibo - integer (km) output bitmap flags lo - logical*1 (mo) output bitmaps (always output) go - real (mo) output fields interpolated iret - integer return code 0 successful interpolation 2 unrecognized input grid or no grid overlap 3 unrecognized output grid 31 invalid undefined output grid 32 invalid budget method parameters

REVISION HISTORY:

```
04-10-96  Mark Iredell; Initial Specification
05-27-04  Sujay Kumar : Modified verision with floating point arithmetic,
```

INTERFACE:

```
subroutine bilinear_interp(gridDesco,ibi,li,gi,ibo,lo,go,mi,mo, &
     rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,IRET)
```

USES:

1.33 Fortran: Module Interface bilinear_interpMod.F90 (Source File: bilinear_interpMod.F90)

This module contains routines that precomputes weights and other parameters required for spatial interpolation of model forcing

REVISION HISTORY:

```
    14Nov02    Sujay Kumar  Initial Specification
```

INTERFACE:

```
 module bilinear_interpMod

   implicit none
```

ARGUMENTS:

```
   real, allocatable      :: rlat0(:)
   real, allocatable      :: rlon0(:)
   integer, allocatable   :: n110(:)
   integer, allocatable   :: n120(:)
   integer, allocatable   :: n210(:)
   integer, allocatable   :: n220(:)
   real, allocatable      :: w110(:),w120(:)
   real, allocatable      :: w210(:),w220(:)
```

1.33.1 allocate_bilinear_interp (Source File: bilinear_interpMod.F90)

! ! ! Allocates memory for interpolation of model forcing data (GEOS and GDAS) !
INTERFACE:

```
   subroutine allocate_bilinear_interp(n)
```

ARGUMENTS:

```
     integer, intent(in) :: n
```

CONTENTS:

```
    allocate(rlat0(n))
    allocate(rlon0(n))
    allocate(n110(n))
    allocate(n120(n))
    allocate(n210(n))
    allocate(n220(n))
    allocate(w110(n))
    allocate(w120(n))
    allocate(w210(n))
    allocate(w220(n))
!    mo = n
!    nn = n

    w110 = 0.0
    w120 = 0.0
    w210 = 0.0
```

```
    w220 = 0.0
  end subroutine allocate_bilinear_interp
```

1.33.2 bilinear_interp_input (Source File: bilinear_interpMod.F90)

! ! ! Calculates spatial variables required for interpolation of GEOS/GDAS ! model forcing
! INTERFACE:

```
   subroutine bilinear_interp_input (gridDesci,gridDesco,npts)
     ! !INPUT ARGUMENTS:
     real, intent(in) :: gridDesci(50)
     integer          :: npts
```

CONTENTS:

```
    mo = npts
    !-----------------------------------------------------------------------
    !  Calls the routines to decode the grid description and
    !  calculates the weights and neighbor information to perform
    !  spatial interpolation. This routine eliminates the need to
    !  compute these weights repeatedly during interpolation.
    !-----------------------------------------------------------------------
    if(gridDesco(1).ge.0) then
       call compute_coord(gridDesco, 0,mo,fill,xpts,ypts,rlon0,rlat0,nv,0)
    endif
    call compute_coord(gridDesci,-1,mo,fill,xpts,ypts,rlon0,rlat0,nv,0)
    do n=1,mo
       xi=xpts(n)
       yi=ypts(n)
       if(xi.ne.fill.and.yi.ne.fill) then
          i1=xi
          i2=i1+1
          j1=yi
          j2=j1+1
          xf=xi-i1
          yf=yi-j1
          n110(n)=get_fieldpos(i1,j1,gridDesci)
          n210(n)=get_fieldpos(i2,j1,gridDesci)
          n120(n)=get_fieldpos(i1,j2,gridDesci)
          n220(n)=get_fieldpos(i2,j2,gridDesci)
          if(min(n110(n),n210(n),n120(n),n220(n)).gt.0) then
             w110(n)=(1-xf)*(1-yf)
             w210(n)=xf*(1-yf)
             w120(n)=(1-xf)*yf
             w220(n)=xf*yf
```

```
         else
            n110(n)=0
            n210(n)=0
            n120(n)=0
            n220(n)=0
         endif
      else
         n110(n)=0
         n210(n)=0
         n120(n)=0
         n220(n)=0
      endif
   enddo
```

1.33.3 compute_coord.F90 (Source File: compute_coord.F90)

This subroutine computes the grid and earth coordinates of the specified domain. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include : (iopt= 0) grid and earth coordinates of all grid points (iopt=+1) earth coordinates of selected grid coordinates (iopt=-1) grid coordinates of selected earth coordinates The current code recognizes the following projections: (gridDesc(1)=000) equidistant cylindrical (gridDesc(1)=001) mercator cylindrical (gridDesc(1)=003) lambert conformal conical (gridDesc(1)=004) gaussian cylindrical (gridDesc(1)=005) polar stereographic azimuthal (gridDesc(1)=201) staggered rotated equidistant cylindrical (gridDesc(1)=202) rotated equidistant cylindrical

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
05-27-04 Sujay Kumar; Modified verision with floating point arithmetic.
input argument list:
  gridDesc     - integer (200) domain description parameters
  iopt     - integer option flag
             ( 0 to compute earth coords of all the grid points)
             (+1 to compute earth coords of selected grid coords)
             (-1 to compute grid coords of selected earth coords)
  npts     - integer maximum number of coordinates
  fill     - real fill value to set invalid output data
             (must be impossible value; suggested value: -9999.)
  xpts     - real (npts) grid x point coordinates if iopt>0
  ypts     - real (npts) grid y point coordinates if iopt>0
  rlon     - real (npts) earth longitudes in degrees e if iopt<0
             (acceptable range: -360. to 360.)
  rlat     - real (npts) earth latitudes in degrees n if iopt<0
             (acceptable range: -90. to 90.)
```

```
  lrot     - integer flag to return vector rotations if 1
output argument list:
  xpts     - real (npts) grid x point coordinates if iopt<=0
  ypts     - real (npts) grid y point coordinates if iopt<=0
  rlon     - real (npts) earth longitudes in degrees e if iopt>=0
  rlat     - real (npts) earth latitudes in degrees n if iopt>=0
  nret     - integer number of valid points computed
             (-1 if projection unrecognized)
```

INTERFACE:

```
subroutine compute_coord(gridDesc,iopt,npts,fill,xpts,ypts,rlon,rlat,nret, &
     lrot)
```

1.33.4 compute_coord_gauss (Source File: compute_coord_gauss.F90)

This subroutine computes the grid and earth coordinates of the specified domain for an gaussian cylindrical projection. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include : (iopt= 0) grid and earth coordinates of all grid points (iopt=+1) earth coordinates of selected grid coordinates (iopt=-1) grid coordinates of selected earth coordinates The current code recognizes the following projections: (gridDesc(1)=000) equidistant cylindrical (gridDesc(1)=001) mercator cylindrical (gridDesc(1)=003) lambert conformal conical (gridDesc(1)=004) gaussian cylindrical (gridDesc(1)=005) polar stereographic azimuthal (gridDesc(1)=201) staggered rotated equidistant cylindrical (gridDesc(1)=202) rotated equidistant cylindrical

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
05-27-04 Sujay Kumar; Modified verision with floating point arithmetic.
input argument list:
  gridDesc     - real (200) domain description parameters
  iopt     - integer option flag
             ( 0 to compute earth coords of all the grid points)
             (+1 to compute earth coords of selected grid coords)
             (-1 to compute grid coords of selected earth coords)
  npts     - integer maximum number of coordinates
  fill     - real fill value to set invalid output data
             (must be impossible value; suggested value: -9999.)
  xpts     - real (npts) grid x point coordinates if iopt>0
  ypts     - real (npts) grid y point coordinates if iopt>0
  rlon     - real (npts) earth longitudes in degrees e if iopt<0
             (acceptable range: -360. to 360.)
  rlat     - real (npts) earth latitudes in degrees n if iopt<0
             (acceptable range: -90. to 90.)
  lrot     - integer flag to return vector rotations if 1
output argument list:
```

```
  xpts     - real (npts) grid x point coordinates if iopt<=0
  ypts     - real (npts) grid y point coordinates if iopt<=0
  rlon     - real (npts) earth longitudes in degrees e if iopt>=0
  rlat     - real (npts) earth latitudes in degrees n if iopt>=0
  nret     - integer number of valid points computed
             (-1 if projection unrecognized)
```

INTERFACE:

```
subroutine compute_coord_gauss(gridDesc,iopt,npts,fill,xpts,ypts,&
     rlon,rlat,nret,lrot)
```

!ROUTINE : compute_coord_latlon.F90
This subroutine computes the grid and earth coordinates of the specified domain for an equidistant cylindrical projection. This routine is based on the grid decoding routines in the ipolates interoplation package.
The input options include : (iopt= 0) grid and earth coordinates of all grid points (iopt=+1) earth coordinates of selected grid coordinates (iopt=-1) grid coordinates of selected earth coordinates The current code recognizes the following projections: (gridDesc(1)=000) equidistant cylindrical (gridDesc(1)=001) mercator cylindrical (gridDesc(1)=003) lambert conformal conical (gridDesc(1)=004) gaussian cylindrical (gridDesc(1)=005) polar stereographic azimuthal (gridDesc(1)=201) staggered rotated equidistant cylindrical (gridDesc(1)=202) rotated equidistant cylindrical

REVISION HISTORY:

```
04-10-96 Mark Iredell;  Initial Specification
05-27-04 Sujay Kumar; Modified verision with floating point arithmetic.
input argument list:
  gridDesc     - integer (200) domain description parameters
  iopt     - integer option flag
             ( 0 to compute earth coords of all the grid points)
             (+1 to compute earth coords of selected grid coords)
             (-1 to compute grid coords of selected earth coords)
  npts     - integer maximum number of coordinates
  fill     - real fill value to set invalid output data
             (must be impossible value; suggested value: -9999.)
  xpts     - real (npts) grid x point coordinates if iopt>0
  ypts     - real (npts) grid y point coordinates if iopt>0
  rlon     - real (npts) earth longitudes in degrees e if iopt<0
             (acceptable range: -360. to 360.)
  rlat     - real (npts) earth latitudes in degrees n if iopt<0
             (acceptable range: -90. to 90.)
  lrot     - integer flag to return vector rotations if 1
output argument list:
  xpts     - real (npts) grid x point coordinates if iopt<=0
  ypts     - real (npts) grid y point coordinates if iopt<=0
  rlon     - real (npts) earth longitudes in degrees e if iopt>=0
  rlat     - real (npts) earth latitudes in degrees n if iopt>=0
```

```
      nret     - integer number of valid points computed
                 (-1 if projection unrecognized)
```

INTERFACE:

```
subroutine compute_coord_latlon(gridDesc,iopt,npts,fill,xpts,ypts,&
     rlon,rlat,nret,lrot)
```

1.33.5 conserv_interp.F90 (Source File: conserv_interp.F90)

This subprogram performs budget interpolation from any grid to any grid for scalar fields. The routine is based on the spatial interpolation package ipolates from NCEP.
The algorithm simply computes (weighted) averages of bilinearly interpolated points arranged in a square box centered around each output grid point and stretching nearly halfway to each of the neighboring grid points. options allow choices of number of points in each radius from the center point (ipopt(1)) which defaults to 2 (if ipopt(1)=-1) meaning that 25 points will be averaged; further options are the respective weights for the radius points starting at the center point (ipopt(2:2+ipopt(1)) which defaults to all 1 (if ipopt(2)=-1.). only horizontal interpolation is performed. the grids are defined by their grid description sections
The grid description arrays are based on the decoding schemes used by NCEP. However, in order to remove the integer arithmetic employed in the original ipolates, the routines are rewritten using real number manipulations. The general structure remains the same.
The current code recognizes the following projections: (gridDesc(1)=0) equidistant cylindrical (gridDesc(1)=1) mercator cylindrical (gridDesc(1)=3) lambert conformal conical (gridDesc(1)=4) gaussian cylindrical (spectral native) (gridDesc(1)=5) polar stereographic azimuthal (gridDesc(1)=202) rotated equidistant cylindrical (eta native) where gridDesc could be either input gridDesci or output gridDesco. as an added bonus the number of output grid points and their latitudes and longitudes are also returned. input bitmaps will be interpolated to output bitmaps. output bitmaps will also be created when the output grid extends outside of the domain of the input grid. the output field is set to 0 where the output bitmap is off. INPUT ARGUMENT LIST: ipopt - integer (20) interpolation options ipopt(1) is number of radius points (defaults to 2 if ipopt(1)=-1); ipopt(2:2+ipopt(1)) are respective weights (defaults to all 1 if ipopt(2)=-1). gridDesci - real(200) input domain description parameters gridDesco - integer (200) output domain description parameters mi - integer dimension of input grid fields mo - integer dimension of output grid fields ibi - integer input bitmap flags li - logical*1 (mi) input bitmaps (if some ibi(k)=1) gi - real (mi) input fields to interpolate
OUTPUT ARGUMENT LIST: no - integer number of output points rlat - real (mo) output latitudes in degrees rlon - real (mo) output longitudes in degrees ibo - integer (km) output bitmap flags lo - logical*1 (mo) output bitmaps (always output) go - real (mo) output fields interpolated iret - integer return code 0 successful interpolation 2 unrecognized input grid or no grid overlap 3 unrecognized output grid 31 invalid undefined output grid 32 invalid budget method parameters

REVISION HISTORY:

```
04-10-96  Mark Iredell; Initial Specification
05-27-04  Sujay Kumar : Modified verision with floating point arithmetic,
```

INTERFACE:

```
subroutine conserv_interp(gridDesco,ibi,li,gi,ibo,lo,go,mi,mo,&
     rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)
```

USES:

```
use conserv_interpMod, only :nb3,nb4
```

1.34 Fortran: Module Interface conserv_interpMod.F90 (Source File: conserv_interpMod.F90)

This module contains routines that precomputes weights and other parameters required for spatial interpolation of model forcing

REVISION HISTORY:

```
14Nov02    Sujay Kumar  Initial Specification
```

INTERFACE:

```
module conserv_interpMod

  implicit none
```

ARGUMENTS:

```
  integer                :: nb3,nb4
  real, allocatable      :: rlat3(:)
  real, allocatable      :: rlon3(:)
  integer, allocatable   :: n113(:,:)
  integer, allocatable   :: n123(:,:)
  integer, allocatable   :: n213(:,:)
  integer, allocatable   :: n223(:,:)
  real, allocatable      ::  w113(:,:),w123(:,:)
  real, allocatable      ::  w213(:,:),w223(:,:)
```

1.34.1 allocate_interp (Source File: conserv_interpMod.F90)

! ! ! Allocates memory for interpolation of model forcing data (GEOS and GDAS) !
INTERFACE:

```
  subroutine allocate_conserv_interp(n)
```

ARGUMENTS:

```
    integer, intent(in) :: n
```

!ROUTINE : gausslat
This subroutine computes gaussian latitudes Computes cosines of colatitude and gaussian weights on the gaussian latitudes. the gaussian latitudes are at the zeroes of the legendre polynomial of the given order.

REVISION HISTORY:

```
04-16-92 Mark Iredell; Initial Specification
10-20-97 Mark Iredell; Increased precision
05-14-02 Urzula Jambor; Reduced limit of eps from e-12 to e-7
INPUT ARGUMENT LIST:
  jmax     - input number of latitudes.
OUTPUT ARGUMENT LIST:
  slat     - real (k) cosines of colatitude.
  wlat     - real (k) gaussian weights.
```

INTERFACE:

```
subroutine gausslat(jmax,slat,wlat)
```

1.34.2 get_field_pos.F90 (Source File: get_fieldpos.F90)

This subprogram returns the field position for a given grid point based on the input grid definition.

REVISION HISTORY:

```
04-10-96  Mark Iredell; Initial Specification
03-11-96  Mark Iredell; Allowed hemispheric grids to wrap over one pole
05-27-04  Sujay Kumar; Modified code with floating point arithmetic
INPUT ARGUMENT LIST:
  i        - integer x grid point
  j        - integer y grid point
  gridDesc     - real (200)  domain description parameters
OUTPUT ARGUMENT LIST:
  gridDesc   - integer position in grib field to locate grid point
               (0 if out of bounds)
```

INTERFACE:

```
function get_fieldpos(i,j,gridDesc) result(field_pos)
```

1.34.3 polfixs.F90 (Source File: polfixs.F90)

This subroutine averages multiple pole scalar values on a latitude/longitude grid. bitmaps may be averaged too.

REVISION HISTORY:

```
04-10-96  Mark Iredell; Initial Specification
INPUT ARGUMENT LIST:
  no       - integer number of grid points
  nx       - integer leading dimension of fields
  km       - integer number of fields
  rlat     - real (no) latitudes in degrees
  rlon     - real (no) longitudes in degrees
  ib       - integer (km) bitmap flags
  lo       - logical*1 (nx,km) bitmaps (if some ib(k)=1)
  go       - real (nx,km) fields
OUTPUT ARGUMENT LIST:
  lo       - logical*1 (nx,km) bitmaps (if some ib(k)=1)
  go       - real (nx,km) fields
```

INTERFACE:

```
subroutine polfixs(nm,nx,km,rlat,rlon,ib,lo,go)
```

1.35 Fortran: Module Interface geosdomain_module.F90 (Source File: geosdomain_module.F90)

Contains routines and variables that define the native domain for GEOS model forcing.

INTERFACE:

```
module geosdomain_module
```

USES:

```
use geosdrv_module
```

ARGUMENTS:

```
type(geosdrvdec) :: geosdrv
integer :: mi
```

1.35.1 defnatgeos.F90 (Source File: geosdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for GEOS data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatgeos(gridDesci)
```

USES:

```
use lisdrv_module, only: lis
use time_manager, only : date2time
implicit none
```

CONTENTS:

```
call readgeoscrd(geosdrv,gridDesci)
mi = geosdrv%ncold*geosdrv%nrold
yr1 = 2002  !grid update time
mo1 = 10
da1 = 01
hr1 = 0; mn1 = 0; ss1 = 0
call date2time(geosdrv%griduptime,updoy,upgmt,yr1,mo1,da1,hr1,mn1,ss1)
geosdrv%gridchange = .true.
```

1.36 Fortran: Module Interface geosdrv_module.F90 (Source File: geosdrv_module.F90)

Module containing runtime specific GEOS variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module geosdrv_module
```

ARGUMENTS:

```
type geosdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   integer :: nmif
   character*40 :: geosdir   !GEOS Forcing Directory
   real*8 :: geostime1,geostime2
   real*8 :: griduptime
   logical :: gridchange
end type geosdrvdec
```

1.37 Fortran: Module Interface geosopendap_module.F90 (Source File: geosopendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O specfific to GEOS forcing routines

REVISION HISTORY:

```
10 Jul 2003; James Geiger  Initial Specification
22 Dec 2003; Sujay Kumar Separted geos specific routines from the main
             module
```

INTERFACE:

```
module geosopendap_module
#if ( defined OPENDAP )
  TODO: Stubs to allow the code to compile
  TODO: GEOS does not yet work in GDS-mode
   integer :: geos_slat = -90
   integer :: geos_nlat = 90
   character*4 :: cgeos_slat = "-90"
   character*4 :: cgeos_nlat = "90"
   integer :: geos_nc = 360
   integer :: geos_nr = 181
   contains
   function get_geos_index(offset)
   integer offset
   integer get_geos_index
   get_geos_index = -9999
   end function get_geos_index
   subroutine opendap_geos_init()
   end subroutine opendap_geos_init
   subroutine def_gridDesc(gridDesci)
      real gridDesci(50)
   end subroutine def_gridDesc
#endif
#if ( defined OPENDAP_KLUGE )
  TODO: Correct this section
```

USES:

```
  use geosdomain_module, only : geosdrv
  use ESMF_TimeMgmtMod
  use lisdrv_module,     only : lis, grid, gindex
  use grid_spmdMod,      only : gdi, gdisp
  use tile_spmdMod,      only : di_array
  use spmdMod
  use opendap_module,    only : opendap_data_prefix, ciam

  implicit none
   type(esmf_date), save :: geos_ref_date !Reference date for GEOS forcing data
   integer               :: geos_slat     !Southern latitude boundary for each subdomain (for
   integer               :: geos_nlat     !Northern latitude boundary for each subdomain (for
   integer               :: geos_nc       !Number of columns for each subdomain (for GEOS)
   integer               :: geos_nr       !Number of rows for each subdomain (for GEOS)
   real                  :: input_slat    !Southern latitude boundary for the native domain
   real                  :: input_nlat    !Northern latitude boundary for the native domain
```

```
    character*4           :: cgeos_slat    !character representation of geos$_-$slat
    character*4           :: cgeos_nlat    !character representation of geos$_-$nlat
    integer               :: grid_offset   !Global to local grid mapping offset
    integer               :: fnroffset

    contains
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{opendap\_geos\_init (Source File: geosopendap\_module.F90)}



    Initializes the GEOS-GDS variables

\bigskip{\sf INTERFACE:}
\begin{verbatim}    subroutine opendap_geos_init()
```

CONTENTS:

```
     call init_geos_vars()
     call reset_geos_filepaths()
```

1.37.1 reset_geos_filepaths (Source File: geosopendap_module.F90)

Resets input data filenames for GEOS forcing for execution through GDS

INTERFACE:

```
      subroutine reset_geos_filepaths()
```

CONTENTS:

```
       geosdrv%geosdir      = trim(opendap_data_prefix)//'/'// &
                              trim(adjustl(ciam))//'/'//geosdrv%geosdir
```

1.37.2 init_geos_vars (Source File: geosopendap_module.F90)

Computes domain decomposition for native as well as interpolated domains for input GEOS forcing

INTERFACE:

```
     subroutine init_geos_vars()
```

CONTENTS:

```
if(lis%d%gridDesc(9) .eq. 0.01) then
   domain = 8
elseif(lis%d%gridDesc(9).eq.0.05) then
   domain = 7
elseif(lis%d%gridDesc(9) .eq. 0.125) then
   domain = 6
elseif(lis%d%gridDesc(9) .eq. 0.25) then
   domain = 5
elseif(lis%d%gridDesc(9) .eq. 0.50) then
   domain = 4
elseif(lis%d%gridDesc(9) .eq. 1.0) then
   domain = 3
elseif((lis%d%gridDesc(9) .eq. 2) .and.  &
     (lis%d%gridDesc(10) .eq. 2.5)) then
   domain = 2
endif
select case (domain)
case(1)
   print*,'Error!  Cannot handle nldas.'
   stop 999
case(2)
   print*,'Error!  Cannot handle 2x2.5.'
   stop 999
case(3) ! 1 deg
   res = 1000
   center = 500
   bottom = -59500
case(4) ! 1/2 deg
   res = 500
   center = 750
   bottom = -59750
case(5) ! 1/4 deg
   res = 250
   center = 875
   bottom = -59875
case(6) ! 5km
   res = 50
   center = 975
   bottom = -59975
case(7) ! 5km
   res = 50
   center = 975
   bottom = -59975
case(8) ! 1km
   res = 10
   center = 995
   bottom = -59995
case DEFAULT
```

```
      print*, "Select domain size (1,2,3,4,5,6,7,8)"
      stop 999
   end select

   call set_geos_lat(geos_slat,geos_nlat,input_slat,input_nlat)

   lis%d%ngrid = gdi(iam)
   lis%d%nch   = di_array(iam)

   geos_nc    = 360
   geos_nr    = ( geos_nlat - geos_slat + 1 )

   fnroffset = geos_slat - 1
   grid_offset = tile(1)%index-1

   write(cgeos_slat, '(i4)') geos_slat
   write(cgeos_nlat, '(i4)') geos_nlat

   print*,'DBG: geos_init -- geos_slat', geos_slat, ' (', iam, ')'
   print*,'DBG: geos_init -- geos_nlat', geos_nlat, ' (', iam, ')'
   print*,'DBG: geos_init -- ngrid', lis%d%ngrid, ' (', iam, ')'
   print*,'DBG: geos_init -- glbngrid', lis%d%glbngrid, ' (', iam, ')'
   print*,'DBG: geos_init -- ncold', geosdrv%ncold, ' (', iam, ')'
   print*,'DBG: geos_init -- nrold', geosdrv%nrold, ' (', iam, ')'
   print*,'DBG: geos_init -- lnc', lis%d%lnc, ' (', iam, ')'
   print*,'DBG: geos_init -- lnr', lis%d%lnr, ' (', iam, ')'
   print*,'DBG: geos_init -- fnroffset', fnroffset, ' (', iam, ')'
   print*,'DBG: geos_init -- grid_offset', grid_offset, ' (', iam, ')'
   print*,'DBG: geos_init -- cgeos_slat ', cgeos_slat, ' (', iam, ')'
   print*,'DBG: geos_init -- cgeos_nlat ', cgeos_nlat, ' (', iam, ')'
   print*,'DBG: geos_init -- geos_nc', geos_nc, ' (', iam, ')'
   print*,'DBG: geos_init -- geos_nr', geos_nr, ' (', iam, ')'
```

1.37.3 def_kgds (Source File: geosopendap_module.F90)

Initializes the kgds array for GEOS-GDS runs.

INTERFACE:

```
subroutine def_kgds(kgdsi)
```

CONTENTS:

```
kgdsi(1) = 0
kgdsi(2) = geos_nc
kgdsi(3) = geos_nr
kgdsi(4) = input_slat
```

```
kgdsi(7) = input_nlat
kgdsi(5) = -180000
kgdsi(6) = 128
kgdsi(8) = 179000
kgdsi(9) = 1000
kgdsi(10) = 1000
kgdsi(20) = 255
```

1.37.4 def_gridDesc (Source File: geosopendap_module.F90)

Initializes the grid description array for GEOS-GDS runs.

INTERFACE:

```
subroutine def_gridDesc(gridDesci)
```

CONTENTS:

```
gridDesci(1) = 0
gridDesci(2) = geos_nc    ! geosdrv%ncold
gridDesci(3) = geos_nr    ! geosdrv%nrold
gridDesci(4) = input_slat ! -90.000
gridDesci(5) = -180.000
gridDesci(7) = input_nlat ! 90.000
gridDesci(8) = 179.000
gridDesci(6) = 128
gridDesci(9) = 1.000
gridDesci(10) = 1.000
gridDesci(20) = 255
```

1.37.5 init_geos_ref_date (Source File: geosopendap_module.F90)

Initializes the reference date for GEOS forcing data

INTERFACE:

```
subroutine init_geos_ref_date()
```

CONTENTS:

```
geos_ref_date = esmf_dateinit(esmf_no_leap, ref_ymd, ref_tod, rc)
```

1.37.6 get_geos_index (Source File: geosopendap_module.F90)

Computes the time-based index for GEOS forcing data

INTERFACE:

```
function get_geos_index(offset)
  implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: offset   ! offset from current date in hours
```

CONTENTS:

```
if ( ref_data_uninit ) then
   print*,'DBG: get_geos_index -- initializing ref date',' (', iam, ')'
   call init_geos_ref_date()
   ref_data_uninit = .false.
endif
ymd = ( lis%t%yr * 10000 ) + ( lis%t%mo * 100 ) + lis%t%da
tod = ( lis%t%hr *  3600 ) + ( lis%t%mn *  60 ) + lis%t%ss
current_date = esmf_dateinit(esmf_no_leap, ymd, tod, rc)

diff = esmf_timeinit()
call esmf_datediff(current_date, geos_ref_date, diff, islater, rc)
call esmf_timeget(diff, ndays, nsecs, rc)

get_geos_index = ( (ndays * 24) + (nsecs / 3600) + offset ) / 3 + 1
print*,'DBG: get_geos_index -- get_geos_index = ', get_geos_index, &
     ' (', iam, ')'
```

1.37.7 set_geos_lat (Source File: geosopendap_module.F90)

Computes the latitudes of the decomposed domain for GEOS forcing data

INTERFACE:

```
subroutine set_geos_lat(slat, nlat, islat, inlat)
   implicit none
```

OUTPUT PARAMETERS:

```
integer, intent(out) :: slat, nlat
real, intent(out)    :: islat, inlat
```

CONTENTS:

```
!-----------------------------------------------------------------
! Set southern latitude index
!-----------------------------------------------------------------
      print*, 'DBG: set_geos_lat -- grid(1)%lat', grid(1)%lat,' (', iam, ')'
      lat = grid(1)%lat
      if ( lat < 0.0 ) then
         slat = int(lat) - 1 ! E.g. map -54.5 \to -55
      else
         slat = int(lat)     ! E.g. map  40.5 \to  40
      endif
      if ( slat < -90 ) then
         print*, 'ERR: set_geos_lat -- Setting slat = -90', &
                 ' (', iam, ')'
         slat = -90
      endif
!-----------------------------------------------------------------
! Set southern latitude boundary
!-----------------------------------------------------------------
      islat = 1000 * slat
!-----------------------------------------------------------------
! Set northern latitude index
!-----------------------------------------------------------------
      print*, 'DBG: set_geos_lat -- grid(gdi(iam))', &
              gdi(iam), grid(gdi(iam))%lat,' (', iam, ')'
      lat = grid(gdi(iam))%lat
      if ( lat < 0.0 ) then
         nlat = int(lat)     ! E.g. map -10.5 \to -10
      else
         nlat = int(lat) + 1 ! E.g. map  10.5 \to  11
      endif
      if ( nlat > 90 ) then
         print*, 'ERR: set_geos_lat -- Setting nlat = 90', &
                 ' (', iam, ')'
         nlat = 90
      endif
!-----------------------------------------------------------------
! Set northern latitude boundary
!-----------------------------------------------------------------
      inlat = 1000 * nlat

      slat = slat + 90 + 1 ! map [-90,90] \to [1,181]
      nlat = nlat + 90 + 1 ! map [-90,90] \to [1,181]
      slat = 1
      nlat = 181
      !islat = -90000
      !inlat = 90000
      islat = -90.000
      inlat = 90.000
```

1.37.8 getgeos.F90 (Source File: getgeos.F90)

Opens, reads, and interpolates GEOS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.38 Core Functions of getgeos

tick Determines GEOS data times

geosfile Puts together appropriate file name for 3 hour intervals

readgeos Interpolates GEOS data to LDAS grid

REVISION HISTORY:

```
 1  Oct 1999: Jared Entin; Initial code
 25 Oct 1999: Jared Entin; Significant F90 Revision
 11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HRFILE
 27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave & longwave data
              Altered times which are passed into ZTERP--used to be GMT1
              and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
 30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
 17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
 13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
  5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
subroutine getgeos()
```

USES:

```
use lisdrv_module, only : lis
use time_manager
use spmdMod
use tile_spmdMod
use baseforcing_module, only: glbdata1,glbdata2
use geosdomain_module, only : geosdrv
use bilinear_interpMod, only : bilinear_interp_input
use conserv_interpMod, only : conserv_interp_input
use lis_indices_module, only : lis_nc_working, lis_nr_working
```

CONTENTS:

```
  if ( masterproc ) then
     nstep = get_nstep(lis%t)
  endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
  call MPI_BCAST(nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,ierr)
#endif
!-----------------------------------------------------------------
! Determine the correct number of forcing variables
!-----------------------------------------------------------------
  if ( nstep .eq. 0) then
     nforce = geosdrv%nmif
  else
     nforce = lis%f%nf
  endif
  lis%f%findtime1=0
  lis%f%findtime2=0
  movetime=0
!-----------------------------------------------------------------
! Determine Required GEOS Data Times
! (The previous hour & the future hour)
!-----------------------------------------------------------------
  yr1=lis%t%yr    !Time now
  mo1=lis%t%mo
  da1=lis%t%da
  hr1=lis%t%hr
  mn1=lis%t%mn
  ss1=0
  ts1=0

  call tick(timenow,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

  yr1=lis%t%yr    !Previous Hour
  mo1=lis%t%mo
  da1=lis%t%da
  hr1=3*((lis%t%hr)/3)
  mn1=0
  ss1=0
  ts1=0
  call tick(time1,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

  yr2=lis%t%yr    !Next Hour
  mo2=lis%t%mo
  da2=lis%t%da
  hr2=3*((lis%t%hr)/3)
  mn2=0
  ss2=0
```

```
  ts2=3*60*60

  call tick(time2,doy2,gmt2,yr2,mo2,da2,hr2,mn2,ss2,ts2)
  if(timenow.gt.geosdrv%geostime2) then
     movetime=1
     lis%f%findtime2=1
  endif

  if ( nstep.eq.0 .or. nstep.eq.1 .or.lis%f%rstflag.eq.1 ) then
     lis%f%findtime1=1
     lis%f%findtime2=1
     glbdata1 = 0
     glbdata2 = 0
     movetime=0
     lis%f%rstflag = 0
  endif
  lis%f%shortflag=2            !Time averaged SW
  lis%f%longflag=2             !Time averaged LW

  if(time1>geosdrv%griduptime.and.geosdrv%gridchange) then
     print*, 'Time change..., Switching to GEOS4'
     geosdrv%ncold = 288
!-----------------------------------------------------------------
! Reinitialize the weights and neighbors
!-----------------------------------------------------------------
     gridDesci = 0
     gridDesci(1) = 0
     gridDesci(2) = geosdrv%ncold
     gridDesci(3) = geosdrv%nrold
     gridDesci(4) = -90.000
     gridDesci(7) = 90.000
     gridDesci(5) = -180.000
     gridDesci(6) = 128
     gridDesci(8) = 179.000
     gridDesci(9) = 1.000
     gridDesci(10) = 1.000
     gridDesci(20) = 255
     if(lis%f%interp .eq.1) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc,lis_nc_working*lis_nr_working)
     elseif(lis%f%interp.eq.2) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc,lis_nc_working*lis_nr_working)
        call conserv_interp_input(gridDesci,lis%d%gridDesc,lis_nc_working*lis_nr_working)
     endif
     geosdrv%gridchange = .false.

     if ( lis%f%ecor == 1 ) then
        lis%f%gridchange = 1
        elevfile = lis%p%elevfile
```

```
        c = index(elevfile,"geos3")
        fpart1 = elevfile(1:c+3)
        fpart2 = elevfile(c+5:40)
        lis%p%elevfile = trim(fpart1) // "4" // trim(fpart2)
        print*, 'Use newer elevation difference file: ', lis%p%elevfile
        print*, 'Transitioned from GEOS3 to GEOS4 grid dimensions.'
        call get_geos4_diff()
     endif
  endif
!-----------------------------------------------------------------
! Establish geostime1
!-----------------------------------------------------------------
  if (lis%f%findtime1==1) then
     order=1
     ferror = 0
     try = 0
     ts1 = -24*60*60
     do
        if ( ferror /= 0 ) then
           exit
        end if
        try = try+1
        call geosfile(name,geosdrv%geosdir,yr1,mo1,da1,hr1,geosdrv%ncold)
        call readgeos(order,name,lis%t%tscount,ferror)
        if ( ferror == 1 ) then
!-----------------------------------------------------------------
! successfully retrieved forcing data
!-----------------------------------------------------------------
           geosdrv%geostime1=time1
        else
!-----------------------------------------------------------------
! ferror still=0, so roll back one day
!-----------------------------------------------------------------
           call tick(dumbtime1,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)
        end if
        if ( try > ndays ) then
           print *, 'ERROR: GEOS data gap exceeds 10 days on file 1'
           call endrun
        end if
     end do
  endif
  if(movetime.eq.1) then
     geosdrv%geostime1=geosdrv%geostime2
     lis%f%findtime2=1
     do f=1,nforce
        do c=1,lis%d%ngrid
           glbdata1(f,c)=glbdata2(f,c)
        enddo
```

```
     enddo
  endif
  if(lis%f%findtime2.eq.1) then
     order=2
     ferror = 0
     try = 0
     ts2 = -24*60*60
     do
        if ( ferror /= 0 ) exit
        try = try+1
        call geosfile(name,geosdrv%geosdir,yr2,mo2,da2,hr2,geosdrv%ncold)
        call readgeos(order,name,lis%t%tscount,ferror)
        if ( ferror == 1 ) then
!-----------------------------------------------------------------
! successfully retrieved forcing data
!-----------------------------------------------------------------
           geosdrv%geostime2=time2
        else
!-----------------------------------------------------------------
! ferror still=0, so roll back one day
!-----------------------------------------------------------------
           call tick(dumbtime2,doy2,gmt2,yr2,mo2,da2,hr2,mn2,ss2,ts2)
        end if
        if ( try > ndays ) then
           print *, 'ERROR: GEOS data gap exceeds 10 days on file 2'
           call endrun
        end if
     end do
  endif

84 format('now',i4,4i3,2x,'pvt ',a22,' nxt ',a22)
!  if ((lis%f%gridchange==1).and.(geosdrv%ncold==288)) then
!     lis%f%gridchange=0
!  endif
  return
```

1.38.1 geosfile (Source File: getgeos.F90)

This subroutine puts together GEOS file name

INTERFACE:

```
 subroutine geosfile(name,geosdir,yr,mo,da,hr,ncold)

   implicit none
```

INPUT PARAMETERS:

```
character*40, intent(in) :: geosdir
integer, intent(in)      :: yr,mo,da,hr,ncold
```

OUTPUT PARAMETERS:

```
character*80, intent(out) :: name
```

CONTENTS:

```
!-----------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----------------------------------------------------------------
  uyr=yr
  umo=mo
  uda=da
  uhr = 3*(hr/3)  !hour needs to be a multiple of 3 hours
!-----------------------------------------------------------------
!  Determine initcode for the hour of the forecast file
!  If the time is 12 or later the file is time stamped
!  with the next day.  So check for that first
!-----------------------------------------------------------------

  if(uhr<3)then
     initcode = '00'
  elseif(uhr<6)then
     initcode = '03'
  elseif(uhr<9)then
     initcode = '06'
  elseif(uhr<12)then
     initcode = '09'
  elseif(uhr<15)then
     initcode = '12'
  elseif(uhr<18)then
     initcode = '15'
  elseif(uhr<21)then
     initcode = '18'
  elseif(uhr<24)then
     initcode = '21'
  endif

  write(UNIT=temp,FMT='(A40)') geosdir
  read(UNIT=temp,FMT='(80A1)') (fbase(i),i=1,80)

  write(UNIT=temp,FMT='(a1,i4,i2,a1)') '/',uyr,umo,'/'
  read(UNIT=temp,FMT='(8A1)') fdir
  do i=1,8
     if(fdir(i).eq.(' ')) fdir(i)='0'
```

```
  enddo

  write(UNIT=temp,FMT='(i4,i2,i2,a2)') uyr,umo,uda,initcode
  read(UNIT=temp,FMT='(10A1)') ftime
  do i=1,10
     if(ftime(i).eq.(' ')) ftime(i)='0'
  enddo

  if(ncold==360) then
     write(UNIT=temp,FMT='(A8)') '.GEOS323'
     read(UNIT=temp,FMT='(80A1)') (fsubs(i),i=1,8)
  else
     write(UNIT=temp,FMT='(A6)') '.GEOS4'
     read(UNIT=temp,FMT='(80A1)') (fsubs(i),i=1,6)
  endif
  c=0
  do i=1,80
     if(fbase(i).eq.(' ').and.c.eq.0) c=i-1
  enddo

  if (ncold==360) then
     write(UNIT=temp,FMT='(80a1)') (fbase(i),i=1,c),(fdir(i),i=1,8), &
          (ftime(i),i=1,10),(fsubs(i),i=1,8)
  else
     write(UNIT=temp,FMT='(80a1)') (fbase(i),i=1,c),(fdir(i),i=1,8), &
          (ftime(i),i=1,10),(fsubs(i),i=1,6)
  endif

  read(UNIT=temp, FMT='(a80)') name
  return
```

1.38.2 readgeoscrd.F90 (Source File: readgeoscrd.F90)

Routine to read GEOS specific parameters from the card file.

REVISION HISTORY:

```
   11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine readgeoscrd(geosdrv,gridDesci)
```

USES:

```
   use geosdrv_module
 #if ( defined OPENDAP )
     use geosopendap_module, only : opendap_geos_init, &
```

```
                                    def_gridDesc
 #endif
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=geos)
  print*,'Using GEOS forcing'
  print*, 'GEOS forcing directory :',geosdrv%GEOSDIR
  geosdrv%geostime1 = 3000.0
  geosdrv%geostime2 = 0.0

#if ( defined OPENDAP )
    call opendap_geos_init()
!    call def_gridDesc(gridDesci)
#endif
#if (! defined OPENDAP) || ( defined OPENDAP )
    gridDesci(1) = 0
    gridDesci(2) = geosdrv%ncold
    gridDesci(3) = geosdrv%nrold
    gridDesci(4) = -90.000
    gridDesci(5) = -180.000
    gridDesci(7) = 90.000
    gridDesci(8) = 179.000
    gridDesci(6) = 128
    gridDesci(9) = 1.000
    gridDesci(10) = 1.000
    gridDesci(20) = 255
#endif
  close(11)
```

1.38.3 readgeos.F90 (Source File: readgeos.F90)

Reads in GEOS data and performs interpolation to the LDAS domain.

1.39 Core Functions of readgeos

bilinear_interp Interpolates GEOS data to LDAS grid using bilinear interpolation

GEOS FORCING VARIABLES (unless noted, fields are 3-hr upstream averaged):
1. T 2m Temperature interpolated to 2 metres $[K]$
2. q 2m Instantaneous specific humidity interpolated to 2 metres$[kg/kg]$
3. radswg Downward shortwave flux at the ground $[W/m^2]$
4. lwgdwn Downward longwave radiation at the ground $[W/m^2]$
5. u 10m Instantaneous zonal wind interpolated to 10 metres $[m/s]$
6. v 10m Instantaneous meridional wind interpolated to 10 metres$[m/s]$
7. ps Instantaneous Surface Pressure $[Pa]$

8. preacc Total precipitation $[mm/s]$
9. precon Convective precipatation $[mm/s]$
10. albedo Surface albedo (0-1)

REVISION HISTORY:

```
 1  Oct 1999: Jared Entin; Initial code
 15 Oct 1999: Paul Houser; Significant F90 Revision
 11 Apr 2000: Brian Cosgrove; Added read statements for forcing interpolation
 17 Apr 2001: Jon Gottschalck; Added code to perform initialization of
                               Mosaic with GEOS forcing and new intp. scheme
 14 Aug 2001: Urszula Jambor; Added ferror flag as a routine argument
 07 Dec 2001: Urszula Jambor; Began used of LDAS$%$LDAS_GRIDDESC array
```

INTERFACE:

```
subroutine readgeos(order,name,tscount,ferror)
```

USES:

```
  use lisdrv_module, only : lis
  use spmdMod
  use baseforcing_module, only : glbdata1, glbdata2
  use bilinear_interpMod, only : w110,w120,w210,w220, &
                                 n110,n120,n210,n220, &
                                 rlat0,rlon0
  use conserv_interpMod, only : w113,w123,w213,w223, &
                                n113,n123,n213,n223, &
                                rlat3,rlon3
  use geosdomain_module, only : geosdrv,mi

#if ( defined OPENDAP )
  use geosopendap_module, only : geos_slat, geos_nlat,  &
                                 cgeos_slat, cgeos_nlat,&
                                 geos_nr,geos_nc,       &
                                 get_geos_index
  use opendap_module, only : ciam
#endif
  use lisdrv_module, only : gindex ! LDAS non-model-specific 1-D variables
  use lis_indices_module, only : lis_nc_working, lis_nr_working
```

CONTENTS:

```
#if ( defined OPENDAP )
  integer :: nr_index, nc_index
  integer :: geos_index
  character*4 :: cgeos_index
  real, allocatable :: debuggeos0(:,:)
  real, allocatable :: debuggeos1(:,:)
```

```
  integer :: ngrid1
  integer :: nmif1
  nr_index = geos_nr
  nc_index = geos_nc
#else
  integer :: nr_index, nc_index
  nr_index = geosdrv%nrold
  nc_index = geosdrv%ncold
#endif

  allocate(tempvar(nc_index,nr_index,geosdrv%nmif), stat=ios)
  call check_error(ios,'Error allocating tempvar.',iam)

  allocate(tempgeos(lis%d%ngrid,geosdrv%nmif), stat=ios)
  call check_error(ios,'Error allocating tempgeos.',iam)

  allocate(f(nc_index*nr_index), stat=ios)
  call check_error(ios,'Error allocating f.',iam)

  allocate(go(lis_nc_working*lis_nr_working), stat=ios)
  call check_error(ios,'Error allocating go.',iam)

  allocate(gmask(nc_index*nr_index), stat=ios)
  call check_error(ios,'Error allocating gmask.',iam)

  allocate(lb(nc_index*nr_index), stat=ios)
  call check_error(ios,'Error allocating lb.',iam)

  allocate(lo(lis_nc_working*lis_nr_working), stat=ios)
  call check_error(ios,'Error allocating lo.',iam)

  gmask = 0.0
  ngeos = nc_index*nr_index
  glis = lis_nc_working*lis_nr_working
  ferror = 1
!-----------------------------------------------------------------------
! Open GEOS forcing file
!-----------------------------------------------------------------------
#if ( defined OPENDAP_KLUGE )
  if ( order == 1 ) then
     geos_index = get_geos_index(0)
  else ! order == 2
     geos_index = get_geos_index(3)
  endif
  write(cgeos_index, '(i4)') geos_index

  print*, 'MSG: readgeos -- Retrieving GEOS forcing file ', &
       trim(name), ' (',iam,')'
```

```
  call system("opendap_scripts/getgeos.pl "//ciam//" "// &
       trim(name)//" "// &
       cgeos_index//" "//cgeos_slat//" "//cgeos_nlat)
#endif
  print*, 'MSG: readgeos -- Reading GEOS forcing file -', &
       trim(name), ' (',iam,')'

#if ( defined OPENDAP )
         open(40,file=name,form='unformatted',iostat=ios)
#else
!<kluge>
         open(40,file=name,form='unformatted',iostat=ios)
!         open(40,file=name,form='unformatted', &
!              access="direct",recl=geosdrv%ncold*geosdrv%nrold*4,iostat=ios)
!</kluge>
#endif
  if ( ios /= 0 ) then
     print*,'ERR: readgeos -- Error opening ', &
          trim(name),'. Stopping.', ' (', iam,')'
     call endrun
  endif

#if ( defined OPENDAP_KLUGE )
  do i = 1,geosdrv%nmif
     read(40,iostat=ioerror)tempvar(:,:,i)
  enddo
#else
!<kluge>
  read(40,iostat=ioerror)tempvar
!  do i = 1,geosdrv%nmif
!     read(40,rec=i,iostat=ioerror)tempvar(:,:,i)
!  enddo
!</kluge>
#endif
  if ( ioerror /= 0 ) then
     print*,'ERR: readgeos -- Error reading ', &
          trim(name),'. ioerror = ',ioerror,'. Stopping.', ' (', iam,')'
     call endrun
  else
     print*, 'MSG: readgeos -- Read GEOS forcing file -', &
          trim(name), ' (',iam,')'
  endif
!-----------------------------------------------------------------------
! Finding number of forcing variables
! (13 if time step is 0, otherwise the normal 10)
!-----------------------------------------------------------------------
  if (tscount .eq. 0) then
     nforce = geosdrv%nmif
```

```
  else
     nforce = 10
  endif
  do v=1,nforce
!-----------------------------------------------------------------------
! Transferring current data to 1-D array for interpolation
!-----------------------------------------------------------------------
     c=0
     do i=1,nr_index    !i=1,lf%nrold
        do j=1,nc_index !j=1,lf%ncold
           c = c + 1
           f(c) = tempvar(j,i,v)
           if (tscount .eq. 0 .and. order .eq. 1 &
                .and. v .eq. 11) then
              gmask(c) = f(c) ! Storing geos land mask for later use
           endif
        enddo
     enddo
!-----------------------------------------------------------------------
! Initializing input and output grid arrays
!-----------------------------------------------------------------------
     gridDesco = 0
     gridDesco = lis%d%gridDesc
     mo = lis_nc_working * lis_nr_working
     if (v .eq. 8 .or. v .eq. 9) then
        ibi = 1
     else
        ibi = 1
     endif
!-----------------------------------------------------------------------
! Defining input data bitmap
!-----------------------------------------------------------------------
     do i=1,ngeos
        lb(i)=.true.
     enddo
!-----------------------------------------------------------------------
! Alter default bitmap prescribed above for
! surface parameters (soil wetness, snow)
!-----------------------------------------------------------------------
     if (v .eq. 12 .or. v .eq. 13) then
        do i=1,ngeos
           if (gmask(i)==100.0 .or. gmask(i)==101.0) then
              lb(i)=.false.
           else
              lb(i)=.true.
           endif
        enddo
     endif
```

```
!-----------------------------------------------------------------------
! Defining output data bitmap
!-----------------------------------------------------------------------
     do i=1,glis
        lo(i)=.true.
     enddo
!-----------------------------------------------------------------------
! Interpolate data from GEOS grid to GLDAS grid
!-----------------------------------------------------------------------
     if(lis%f%interp.eq.1) then
        call bilinear_interp(gridDesco,ibi,lb,f,ibo,lo,go,mi,mo, &
             rlat0,rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
     elseif(lis%f%interp.eq.2) then
        if(v.eq.8 .or. v.eq. 9) then
           call conserv_interp(gridDesco,ibi,lb,f,ibo,lo,go,mi,mo,&
                rlat3,rlon3,w113,w123,w213,w223,n113,n123,n213,n223,iret)
        else
           call bilinear_interp(gridDesco,ibi,lb,f,ibo,lo,go,mi,mo, &
                rlat0,rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
        endif
     endif
!-----------------------------------------------------------------------
! Convert data to original 3D array & a 2D array to
! fill in of missing points due to geography difference
!-----------------------------------------------------------------------
     count = 0
     do j = 1, lis_nr_working
        do i = 1, lis_nc_working
           if(gindex(i,j) .ne. -1) then
              tempgeos(gindex(i,j),v) = go(i+count)
           endif
        enddo
        count = count + lis%d%lnc
     enddo
!-----------------------------------------------------------------------
! Fill in undefined and ocean points
!-----------------------------------------------------------------------
     do i = 1, lis%d%ngrid
        if (tempgeos(i,v) >= 9.9e+14) then
           tempgeos(i,v) = lis%d%udef
        endif
        if(order.eq.1)then
           glbdata1(v,i)=tempgeos(i,v)
        else
           glbdata2(v,i)=tempgeos(i,v)
        endif
     enddo                !i
  enddo                   !v
```

```
print*,'DBG: readgeos -- Deallocating arrays'

deallocate(tempvar, stat=ios)
call check_error(ios,'Error deallocating tempvar.',iam)

deallocate(tempgeos, stat=ios)
call check_error(ios,'Error deallocating tempgeos.',iam)

deallocate(f, stat=ios)
call check_error(ios,'Error deallocating f.',iam)

deallocate(go, stat=ios)
call check_error(ios,'Error deallocating go.',iam)

deallocate(gmask, stat=ios)
call check_error(ios,'Error deallocating gmask.',iam)

deallocate(lb, stat=ios)
call check_error(ios,'Error deallocating lb.',iam)

deallocate(lo, stat=ios)
call check_error(ios,'Error deallocating lo.',iam)

print*, 'MSG: readgeos -- Closing GEOS forcing file -', &
     trim(name), ' (',iam,')'
close(40, iostat=ios)
if ( ios /= 0 ) then
   print*,'ERR: readgeos -- Error closing ', trim(name), &
        '. Stopping.', ' (', iam, ')'
   call endrun
endif

print*,'DBG: readgeos -- leaving', ' (', iam, ')'

return
```

1.39.1 time_interp_geos.F90 (Source File: time_interp_geos.F90)

Opens, reads, and interpolates GEOS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.40 Core Functions of time_interp_geos

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
 1  Oct 1999: Jared Entin; Initial code
 25 Oct 1999: Jared Entin; Significant F90 Revision
 11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HRFILE
 27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave & longwave data
              Altered times which are passed into ZTERP--used to be GMT1
              and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
 30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
 17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
 13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
  5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
subroutine time_interp_geos()
```

USES:

```
  use lisdrv_module, only : lis, grid
  use baseforcing_module, only : glbdata1, glbdata2
  use time_manager
  use grid_spmdMod
  use spmdMod
  use geosdomain_module, only :geosdrv
```

CONTENTS:

```
  if(masterproc) then
     if (get_nstep(lis%t) .eq. 0) then
        lis%f%nforce = lis%f%nmif
     else
        lis%f%nforce = lis%f%nf
     endif
  endif

#if(defined SPMD)
  call MPI_BCAST(geosdrv%geostime1,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(geosdrv%geostime2,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%nforce,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
```

```
  call MPI_BCAST(lis%t%time,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%shortflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%longflag,1,MPI_INTEGER,0, &
         MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%rstflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
#endif
  btime=geosdrv%geostime1
  call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
  btime=geosdrv%geostime2
  call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
!-----------------------------------------------------------------------
!  Interpolate Data in Time
!-----------------------------------------------------------------------
  wt1=(geosdrv%geostime2-lis%t%time)/ &
       (geosdrv%geostime2-geosdrv%geostime1)
  wt2=1.0-wt1
  do f=1,lis%f%nforce
     if(f.eq.3) then
        if (lis%f%shortflag.eq.2) then
!-----------------------------------------------------------------------
! Got Time Averaged SW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              zdoy=lis%t%doy
!              print*, c, zdoy, grid(c)%lat, grid(c)%lon, &
!                   gmt1, gmt2, lis%t%gmt
              call zterp(0,grid(c)%lat,grid(c)%lon, &
                   gmt1,gmt2,lis%t%gmt,zdoy, &
                   zw1,zw2,czb,cze,czm,lis)
              grid(c)%forcing(f)=glbdata2(f,c)*zw1
              if ((grid(c)%forcing(f).ne.lis%d%udef).and. &
                   (grid(c)%forcing(f).lt.0) ) then
                 if (grid(c)%forcing(f) > -0.00001) then
                    grid(c)%forcing(f) = 0.0
                 else
                    print*,'ERR: time_interp_geos -- Stopping because ', &
                         'forcing not udef but lt0,'
                    print*,'ERR: time_interp_geos -- ', &
                         'f,c,grid(c)%forcing(f),glbdata2(f,c)',  &
                         f,c,grid(c)%forcing(f),glbdata2(f,c), &
                         ' (',iam,')'
                    call endrun
                 end if
```

```
              endif

              if (grid(c)%forcing(f).gt.1367) then
                 grid(c)%forcing(f)=glbdata2(f,c)
              endif
           enddo
        endif

     else if(f.eq.8.or.f.eq.9) then
!-----------------------------------------------------------------------
! precip variable Block Interpolation
!-----------------------------------------------------------------------
        do c=1,gdi(iam)
           grid(c)%forcing(f)=glbdata2(f,c)
        enddo

     else if (f.eq.4) then
        if (lis%f%longflag.eq.1) then
!-----------------------------------------------------------------------
!    Got Instantaneous LW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
                   glbdata2(f,c)*wt2
           enddo
        endif

        if (lis%f%longflag.eq.2) then
!-----------------------------------------------------------------------
!    Got Time Averaged LW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              grid(c)%forcing(f)=glbdata2(f,c)

           enddo
          endif

       else
!-----------------------------------------------------------------------
!     Linearly interpolate everything else
!-----------------------------------------------------------------------
          do c=1,gdi(iam)
             grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
                  glbdata2(f,c)*wt2
          enddo
       endif
    enddo
84  format('now',i4,4i3,2x,'pvt ',a22,' nxt ',a22)
```

```
return
```

1.41 Fortran: Module Interface gdasdomain_module.F90 (Source File: gdasdomain_module.F90)

Contains routines and variables that define the native domain for GDAS model forcing

INTERFACE:

```
module gdasdomain_module
```

USES:

```
use gdasdrv_module
```

1.41.1 defnatgdas.F90 (Source File: gdasdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for GDAS data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatgdas(gridDesci)
```

USES:

```
use lisdrv_module, only: lis
use time_manager, only : date2time
implicit none
```

ARGUMENTS:

```
real, intent(inout) :: gridDesci(50)
integer :: updoy, yr1,mo1,da1,hr1,mn1,ss1
real :: upgmt
```

CONTENTS:

```
call readgdascrd(gdasdrv)
gridDesci(1) = 4
gridDesci(2) = 384
gridDesci(3) = 190
gridDesci(4) = 89.277
gridDesci(5) = 0
gridDesci(6) = 128
```

```
gridDesci(7) = -89.277
gridDesci(8) = -0.938
gridDesci(9) = 0.938
gridDesci(10) = 95
gridDesci(20) = 255
mi = gdasdrv%ncold*gdasdrv%nrold

yr1 = 2000
mo1 = 01
da1 = 24
hr1 = 12
mn1 = 0; ss1 = 0
call date2time( gdasdrv%griduptime1,updoy,upgmt,yr1,mo1,da1,hr1,mn1,ss1 )

yr1 = 2002     !grid update time
mo1 = 10
da1 = 29
hr1 = 12
mn1 = 0; ss1 = 0
call date2time(gdasdrv%griduptime2,updoy,upgmt,yr1,mo1,da1,hr1,mn1,ss1 )
gdasdrv%gridchange1 = .true.
gdasdrv%gridchange2 = .true.
```

1.42 Fortran: Module Interface gdasdrv_module.F90 (Source File: gdasdrv_module.F90)

Module containing runtime specific GDAS variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module gdasdrv_module
```

ARGUMENTS:

```
type gdasdrvdec
   integer      :: ncold, nrold   !AWIPS 212 dimensions
   integer      :: nmif
   character*40 :: gdasdir   !GDAS Forcing Directory
   real*8       :: gdastime1,gdastime2
   real*8       :: griduptime1,griduptime2
   logical      :: gridchange1, gridchange2
end type gdasdrvdec
```

1.43 Fortran: Module Interface gdasopendap_module.F90 (Source File: gdasopendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O specfific to GDAS forcing routines

REVISION HISTORY:

```
05 Feb 2004; James Geiger  Initial Specification
```

INTERFACE:

```
module gdasopendap_module
#if ( defined OPENDAP )
```

USES:

```
   use lisdrv_module, only : lis, grid
   use grid_spmdMod,   only : gdi, gdisp
   use tile_spmdMod,   only : di_array
   use spmdMod

   implicit none
    integer               :: gdas_slat     !Southern latitude boundary for each subdomain (for
    integer               :: gdas_nlat     !Northern latitude boundary for each subdomain (for
    integer               :: gdas_wlon     !Western longitude boundary for each subdomain (for
    integer               :: gdas_elon     !Eastern longitude boundary for each subdomain (for
    integer               :: gdas_nc       !Number of columns for each subdomain (for GDAS)
    integer               :: gdas_nr       !Number of rows for each subdomain (for GDAS)
    integer               :: input_slat    !Southern latitude boundary for the native domain
    integer               :: input_nlat    !Northern latitude boundary for the native domain
    character*4           :: cgdas_slat    !character representation of gdas$_-$slat
    character*4           :: cgdas_nlat    !character representation of gdas$_-$nlat
    character*4           :: cgdas_wlon    !character representation of gdas$_-$wlon
    character*4           :: cgdas_elon    !character representation of gdas$_-$elon
    integer               :: grid_offset   !Global to local grid mapping offset
    integer               :: fnroffset

    contains
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{opendap\_gdas\_init (Source File: gdasopendap\_module.F90)}



    Initializes the GDAS-GDS variables

\bigskip{\sf INTERFACE:}
\begin{verbatim}    subroutine opendap_gdas_init(gdasdrv)
```

CONTENTS:

```
call init_gdas_vars()
call reset_gdas_filepaths(gdasdrv)
```

1.43.1 reset_gdas_filepaths (Source File: gdasopendap_module.F90)

Resets input data filenames for GDAS forcing for execution through GDS

INTERFACE:

```
subroutine reset_gdas_filepaths(gdasdrv)
```

CONTENTS:

```
gdasdrv%gdasdir      = trim(opendap_data_prefix)//'/'// &
                       trim(adjustl(ciam))//'/'//gdasdrv%gdasdir
```

1.43.2 init_gdas_vars (Source File: gdasopendap_module.F90)

Computes domain decomposition for native as well as interpolated domains for input GDAS forcing

INTERFACE:

```
subroutine init_gdas_vars()
```

CONTENTS:

```
call set_gdas_lat(gdas_slat,gdas_nlat,gdas_wlon,gdas_elon)

lis%d%ngrid = gdi(iam)
lis%d%nch   = di_array(iam)

gdas_nc    = ( gdas_elon - gdas_wlon + 1 )
gdas_nr    = ( gdas_nlat - gdas_slat + 1 )

fnroffset = gdas_slat - 1
grid_offset = tile(1)%index-1

write(cgdas_slat, '(i4)') gdas_slat
write(cgdas_nlat, '(i4)') gdas_nlat
write(cgdas_wlon, '(i4)') gdas_wlon
write(cgdas_elon, '(i4)') gdas_elon

print*,'DBG: gdas_init -- gdas_slat', gdas_slat, ' (', iam, ')'
```

```
print*,'DBG: gdas_init -- gdas_nlat', gdas_nlat, ' (', iam, ')'
print*,'DBG: gdas_init -- gdas_wlon', gdas_wlon, ' (', iam, ')'
print*,'DBG: gdas_init -- gdas_elon', gdas_elon, ' (', iam, ')'
print*,'DBG: gdas_init -- ngrid', lis%d%ngrid, ' (', iam, ')'
print*,'DBG: gdas_init -- glbngrid', lis%d%glbngrid, ' (', iam, ')'
!print*,'DBG: gdas_init -- ncold', gdasdrv%ncold, ' (', iam, ')'
!print*,'DBG: gdas_init -- nrold', gdasdrv%nrold, ' (', iam, ')'
print*,'DBG: gdas_init -- lnc', lis%d%lnc, ' (', iam, ')'
print*,'DBG: gdas_init -- lnr', lis%d%lnr, ' (', iam, ')'
print*,'DBG: gdas_init -- fnroffset', fnroffset, ' (', iam, ')'
print*,'DBG: gdas_init -- grid_offset', grid_offset, ' (', iam, ')'
print*,'DBG: gdas_init -- cgdas_slat ', cgdas_slat, ' (', iam, ')'
print*,'DBG: gdas_init -- cgdas_nlat ', cgdas_nlat, ' (', iam, ')'
print*,'DBG: gdas_init -- cgdas_wlon ', cgdas_wlon, ' (', iam, ')'
print*,'DBG: gdas_init -- cgdas_elon ', cgdas_elon, ' (', iam, ')'
print*,'DBG: gdas_init -- gdas_nc', gdas_nc, ' (', iam, ')'
print*,'DBG: gdas_init -- gdas_nr', gdas_nr, ' (', iam, ')'
```

1.43.3 def_kgds (Source File: gdasopendap_module.F90)

Initializes the kgds array for GDAS-GDS runs.

INTERFACE:

```
subroutine def_kgds(kgdsi)
```

CONTENTS:

```
kgdsi(1) = 4
kgdsi(2) = 512
kgdsi(3) = 256
kgdsi(4) = 89463
kgdsi(5) = 0
kgdsi(6) = 128
kgdsi(7) = -89463
kgdsi(8) = -703
kgdsi(9) = 703
kgdsi(10) = 128
kgdsi(20) = 255
```

1.43.4 set_gdas_lat (Source File: gdasopendap_module.F90)

Computes the latitudes of the decomposed domain for GDAS forcing data

INTERFACE:

```
    subroutine set_gdas_lat(slat, nlat, wlon, elon)
       implicit none
```

OUTPUT PARAMETERS:

```
       integer, intent(out) :: slat, nlat, wlon, elon
```

CONTENTS:

```
   ! Return global domain
   gdas_slat = 1
   gdas_nlat = 256
   gdas_wlon = 1
   gdas_elon = 512
```

1.43.5 twotoone (Source File: gdasopendap_module.F90)

Remaps a 2-dimensional array of GDAS forcing values into a 1-dimensional array.

INTERFACE:

```
 subroutine twotoone(f2d,f,nc,nr)
    implicit none
```

INPUT PARAMETERS:

```
    integer, intent(in) :: nc, nr
    real, dimension(nc,nr), intent(in) :: f2d
```

OUTPUT PARAMETERS:

```
    real, dimension(nc*nr), intent(out) :: f
```

CONTENTS:

```
   integer :: i, j, k

   k = 1
   do j = nr, 1, -1
      do i = 1, nc
         f(k) = f2d(i,j)
         k = k + 1
      enddo
   enddo
end subroutine twotoone
```

1.43.6 get_kpds (Source File: gdasopendap_module.F90)

Sets the local kpds array for use in interp_gdas.

INTERFACE:

```
subroutine get_kpds(kpds,iv)
   implicit none
```

INPUT PARAMETERS:

```
   integer, intent(in) :: iv
```

OUTPUT PARAMETERS:

```
   integer, intent(out) :: kpds(200)
```

CONTENTS:

```
   kpds = 0

   select case ( iv )
   case ( 1 )
      kpds(1:25) = (/7,82,255,128,11,105,2,1,6,10,18,0,1,3,0,&
                     10,0,1,2,0,21,1,0,0,32/)
   case ( 2 )
      kpds(1:25) = (/7,82,255,128,51,105,2,1,6,10,18,0,1,3,0,&
                     10,0,1,2,0,21,4,0,0,32/)
   case ( 3 )
      kpds(1:25) = (/7,82,255,128,204,1,0,1,6,10,18,0,1,0,3,3,&
                     0,1,2,0,21,0,0,0,32/)
   case ( 4 )
      kpds(1:25) = (/7,82,255,128,205,1,0,1,6,10,18,0,1,0,3,3,0,&
                     1,2,0,21,0,0,0,32/)
   case ( 5 )
      kpds(1:25) = (/7,82,255,128,33,105,10,1,6,10,18,0,1,3,0,10,&
                     0,1,2,0,21,1,0,0,32/)
   case ( 6 )
      kpds(1:25) = (/7,82,255,128,34,105,10,1,6,10,18,0,1,3,0,10,&
                     0,1,2,0,21,1,0,0,32/)
   case ( 7 )
      kpds(1:25) = (/7,82,255,128,1,1,0,1,6,10,18,0,1,3,0,10,0,1,&
                     2,0,21,-1,0,0,32/)
   case ( 8 )
      kpds(1:25) = (/7,82,255,128,59,1,0,1,6,10,18,0,1,0,3,3,0,1,&
                     2,0,21,6,0,0,32/)
   case ( 9 )
      kpds(1:25) = (/7,82,255,128,214,1,0,1,6,10,18,0,1,0,3,3,0,&
                     1,2,0,21,6,0,0,32/)
   case ( 10 )
      kpds(1:25) = (/7,82,255,128,84,1,0,1,6,10,18,0,1,0,3,3,0,1,&
```

```
              2,0,21,1,0,0,32/)
   case default
      print*,'ERR: get_kpds -- forcing index ',iv,'is out of range (10)',&
             ' (',iam,')'
      stop 344
   end select
end subroutine get_kpds
```

1.43.7 get_kgds (Source File: gdasopendap_module.F90)

Sets the local kgds array for use in interp_gdas.

INTERFACE:

```
 subroutine get_kgds(kgds)
    implicit none
```

OUTPUT PARAMETERS:

```
    integer, intent(out) :: kgds(200)
```

CONTENTS:

```
   kgds = 0
   kgds(1:20) = (/4,512,256,89463,0,128,-89463,-703,703,128,&
                  0,0,0,0,0,0,0,0,0,255/)

end subroutine get_kgds
```

1.43.8 get_gridDesc (Source File: gdasopendap_module.F90)

Sets the local grid description array for use in interp_gdas.

INTERFACE:

```
 subroutine get_gridDesc(gridDesc)
    implicit none
```

OUTPUT PARAMETERS:

```
    integer, intent(out) :: gridDesc(50)
```

CONTENTS:

```
   gridDesc = 0
   gridDesc(1:20) = (/4,512,256,89463,0,128,-89463,-703,703,128,&
                      0,0,0,0,0,0,0,0,0,255/)
```

```
   call lis_log_msg('DBG: get_gridDesc -- FIX THIS - DOES NOT HANDLE GRID CHANGE')

end subroutine get_gridDesc
```

1.43.9 set_lb (Source File: gdasopendap_module.F90)

Sets the local lb mask array for use in interp_gdas.

INTERFACE:

```
 subroutine set_lb(f,lb,nc,nr)

    implicit none
```

INPUT PARAMETERS:

```
    integer, intent(in) :: nc,nr
    real, dimension(nc*nr), intent(in) :: f
```

OUTPUT PARAMETERS:

```
    logical*1, intent(out) :: lb(nc*nr)
```

CONTENTS:

```
   integer :: i

   lb = .false.

   do i = 1, nc*nr
      if ( f(i) > 0.0 ) then
         lb(i) = .true.
      endif
   enddo

end subroutine set_lb
```

1.43.10 getgdas.F90 (Source File: getgdas.F90)

Opens, reads, and interpolates NCEP-GDAS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The idea is to open either the 00 or 03 forecast file associated with the most recent GDAS assimilation (available every 6 hours). Precipitation rates and radiation fluxes will be taken from the F03 and F06 files, since averages are provided.
- if that fails, the strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.44 Core Functions of getgdas

tick Determines GDAS data times

gdasfile Puts together appropriate file name for 3 hour intervals

gdasfilef06 Puts together appropriate file name for 6 hour intervals

retgdas Interpolates GDAS data to LDAS grid

REVISION HISTORY:

```
1  Oct 1999: Jared Entin; Initial code
25 Oct 1999: Jared Entin; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Fixed name construction error
             in Subroutine ETA6HRFILE
27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
             data with opposite sign convention from recent shortwave data.
             Added capability to use time averaged shortwave and longwave data.
             Altered times which are passed into ZTERP--used to be GMT1 and GMT2,
             now they are LDAS%ETATIME1 and LDAS%ETATIME2
11 May 2000: Brian Cosgrove; Added checks for SW values that are too high
             due to zenith angle weighting...if too high, use linear weighting.
             Also, if cos(zen) less than .1, then use linear weighting to
             avoid computed values of SW that are too high.
18 May 2000: Brian Cosgrove; Corrected line of code in ETAEDASNAME which
             assigned wrong year directory variable when constructing
             EDAS filename
26 May 2000: Jared Entin; Changed numerical bound of the TRY variable
             to fix a rollback problem.
5 June 2000: Brian Cosgrove; Fixed a problem with the correction of the negative
             radiation sign convention. Prior to fix, was not correcting negative
             values of -999.9...now it changes all negative values to positive ones.
18 Aug 2000: Brian Cosgrove; Fixed undefined value problem in check for negative
             radiation values over land points.
08 Dec 2000: Urszula Jambor; Rewrote geteta.f in fortran90 to use GDAS in GLDAS
15 Mar 2001: Jon Gottschalck; Slight change to handle more forcing parameters at
             time step 0.
09 Apr 2001: Urszula Jambor; Added capability of using DAAC forcing data every
             6 hours, rather than every 3 hours.
30 May 2001: Urszula Jambor; Changed forcing used: T,q,u fields taken
             from F00 & F03 files, radiation and precip. fields taken
             from F06 & F03 (F03 fields are subtracted out from F06)
```

INTERFACE:

```
#include "misc.h"
subroutine getgdas()
```

USES:

```
   use lisdrv_module, only : lis, gindex
   use baseforcing_module, only: glbdata1, glbdata2
   use time_manager
   use gdasdomain_module, only : gdasdrv
   use bilinear_interpMod, only : bilinear_interp_input
   use conserv_interpMod, only : conserv_interp_input
   use spmdMod
   use lis_indices_module, only: lis_nc_working, lis_nr_working
```

CONTENTS:

```
  lis%f%F00_flag = 0
  lis%f%F06_flag = 0

  lis%f%findtime1=0
  lis%f%findtime2=0
  movetime=0

!-----------------------------------------------------------------
! Determine the correct number of forcing variables
!-----------------------------------------------------------------
  if ( masterproc ) then
     nstep = get_nstep(lis%t)
  endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
  call MPI_BCAST(nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,ferror)
#endif
  if ( nstep == 0 ) then
     nforce = gdasdrv%nmif
  else
     nforce = lis%f%nf
  endif
  if ( nstep == 1 .or. lis%f%rstflag == 1) then
    lis%f%findtime1=1
    lis%f%findtime2=1
    glbdata1 = 0
    glbdata2 = 0
    movetime=0        ! movetime is not properly set at time-step = 1
    lis%f%rstflag = 0
  endif


!-----------------------------------------------------------------
```

```
! Determine required GDAS data times
! (previous assimilation, current & future assimilation hours)
! The adjustment of the hour and the direction will be done
! in the subroutines that generate the names
!-----------------------------------------------------------------
  yr1 = lis%t%yr  !current time
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = lis%t%mn
  ss1 = 0
  ts1 = 0
  call tick( timenow, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

  yr1 = lis%t%yr  !previous assimilation/forecast hour
  mo1 = lis%t%mo
  da1 = lis%t%da
  if ( lis%f%F06_flag == 0 ) then
     hr1 = 3*(int(real(lis%t%hr)/3.0))
  else
     hr1 = 6*(int(real(lis%t%hr)/6.))
  end if
  mn1 = 0
  ss1 = 0
  ts1 = 0
  call tick( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

  yr2 = lis%t%yr  !next assimilation/forecast hour
  mo2 = lis%t%mo
  da2 = lis%t%da
  if ( lis%f%F06_flag == 0 ) then
     hr2 = 3*(int(real(lis%t%hr)/3.0))
  else
     hr2 = 6*(int(real(lis%t%hr)/6.0))
  end if
  mn2 = 0
  ss2 = 0
  if ( lis%f%F06_flag == 0 ) then
     ts2 = 3*60*60
  else
     ts2 = 6*60*60
  end if
  call tick( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
!-----------------------------------------------------------------
! Use these if need to roll back time.
!-----------------------------------------------------------------
  dumbtime1 = time1
  dumbtime2 = time2
```

```
!-----------------------------------------------------------------
! Check to see if current time (timenow) has crossed past gdastime2,
! requiring that both gdastime2 be assigned to gdastime1 and a new
! gdastime2 be set 3 or 6 hours ahead of the current gdastime2.
!-----------------------------------------------------------------
  if ( timenow > gdasdrv%gdastime2 ) then
     movetime  = 1
     lis%f%findtime2 = 1
  end if

  if(time1>gdasdrv%griduptime1 .and. &
       time1 <gdasdrv%griduptime2.and. &
       gdasdrv%gridchange1) then
     print*, 'changing grid....to 2000-2002'

     gdasdrv%ncold = 512
     gdasdrv%nrold = 256
!-------------------------------------------------------------------
! Reinitialize the weights and neighbors
!-------------------------------------------------------------------
     gridDesci = 0
     gridDesci(1) = 4
     gridDesci(2) = 512
     gridDesci(3) = 256
     gridDesci(4) = 89.463
     gridDesci(5) = 0
     gridDesci(6) = 128
     gridDesci(7) = -89.463
     gridDesci(8) = -0.703
     gridDesci(9) = 0.703
     gridDesci(10) = 128
     gridDesci(20) = 255
     if(lis%f%interp.eq.1) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc,&
             lis_nc_working*lis_nr_working)
     elseif(lis%f%interp.eq.2) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc,&
             lis_nc_working*lis_nr_working)
        call conserv_interp_input(gridDesci,lis%d%gridDesc,&
             lis_nc_working*lis_nr_working)
     endif
     gdasdrv%gridchange1 = .false.
  else if(time1 > gdasdrv%griduptime2.and. &
       gdasdrv%gridchange2) then
     print*, 'changing grid to 2002-'
     gdasdrv%ncold = 768
     gdasdrv%nrold = 384
!-------------------------------------------------------------------
```

```
! Reinitialize the weights and neighbors
!-----------------------------------------------------------------
     gridDesci = 0
     gridDesci(1) = 4
     gridDesci(2) = 768
     gridDesci(3) = 384
     gridDesci(4) = 89.642
     gridDesci(5) = 0
     gridDesci(6) = 128
     gridDesci(7) = -89.642
     gridDesci(8) = -0.469
     gridDesci(9) = 0.469
     gridDesci(10) = 192
     gridDesci(20) = 255
     if(lis%f%interp.eq.1) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc, &
                                   lis_nc_working*lis_nr_working)
     elseif(lis%f%interp.eq.2) then
        call bilinear_interp_input(gridDesci,lis%d%gridDesc, &
                                   lis_nc_working*lis_nr_working)
        call conserv_interp_input(gridDesci,lis%d%gridDesc, &
                                  lis_nc_working*lis_nr_working)
     endif
     gdasdrv%gridchange2 = .false.
  endif
!----------------------------------------------------------------
! Establish gdastime1
!----------------------------------------------------------------
  if ( lis%f%findtime1 == 1 ) then  !get new time1 from the past
     print *, 'Getting new time1 data'
     ferror = 0
     order = 1
     try = 0
     ts1 = -24*60*60

     do
        if ( ferror /= 0 ) exit
        try = try+1
        if ( lis%f%F06_flag == 0 ) then
           call gdasfile   ( name, gdasdrv%gdasdir, yr1, mo1, da1, hr1 )
        else
           call gdasfileF06( name, gdasdrv%gdasdir, yr1, mo1, da1, hr1 )
        end if
        print*, 'Reading GDAS file1 ',name
        call retgdas( order, name, nameF06, 0,ferror, try)

        if ( ferror == 1 ) then
!----------------------------------------------------------------
```

```
! successfully retrieved forcing data
!-----------------------------------------------------------------
           gdasdrv%gdastime1 = time1
        else
!-----------------------------------------------------------------
! ferror still=0, so roll back one day & start again
!-----------------------------------------------------------------
           call tick( dumbtime1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
        end if
        if ( try > ndays ) then
!-----------------------------------------------------------------
! data gap exceeds 10 days so stop
!-----------------------------------------------------------------
           print *, 'ERROR: GDAS data gap exceeds 10 days on file 1'
           call endrun
        end if
     end do
  end if
!-----------------------------------------------------------------
! Establish gdastime2
!-----------------------------------------------------------------
  if ( movetime == 1 ) then
     gdasdrv%gdastime1 = gdasdrv%gdastime2
     lis%f%findtime2 = 1
     do f = 1, nforce
        do c = 1, lis%d%ngrid
           glbdata1(f,c) = glbdata2(f,c)
        end do
     end do
  end if

  if ( lis%f%findtime2 == 1 ) then
     print *, 'Getting new time2 data'
     lis%f%F00_flag = 0

     ferror = 0
     order = 2
     try = 0
     ts2 = -24*60*60
!-----------------------------------------------------------------
! determine if both F00 & F06 files needed
!-----------------------------------------------------------------
     if ( lis%f%F06_flag == 0 ) then
        if ( modulo(hr2,2) > 0 ) then
           lis%f%F00_flag = 0 !odd hr, use F03 file
        else
           lis%f%F00_flag = 1 !even hr, use F00 & F06
        end if
```

```
     end if
     do
        if ( ferror /= 0 ) exit
        try = try+1
        if ( lis%f%F06_flag == 0 ) then
           if ( lis%f%F00_flag == 0 ) then
              call gdasfile( name,gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
           else
              call gdasfile( name,gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
              call gdasfileF06( nameF06, gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
           end if
        else
           call gdasfileF06( name, gdasdrv%gdasdir, yr2, mo2, da2, hr2 )
        end if
        print*, 'Reading GDAS file2 ',name
        call retgdas( order, name, nameF06, &
             lis%f%F00_flag, ferror,try )
        if ( ferror == 1 ) then
!-----------------------------------------------------------------
! successfully retrieved forcing data
!-----------------------------------------------------------------
           gdasdrv%gdastime2 = time2
        else
!-----------------------------------------------------------------
! ferror still=0, so roll back one day & start again
!-----------------------------------------------------------------
           call tick( dumbtime2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
        end if
        if ( try > ndays ) then
!-----------------------------------------------------------------
! data gap exceeds 10 days so stop
!-----------------------------------------------------------------
           print *, 'ERROR: GDAS data gap exceeds 10 days on file 2'
           call endrun
        end if
     end do
  end if
!-----------------------------------------------------------------
! loop through all forcing parameters & interpolate
!-----------------------------------------------------------------
  do f = 1, lis%f%nforce
     if ( (f == 3) .or. (f == 4) ) then
        do c = 1, lis_nc_working
           do r = 1, lis_nr_working
              index = gindex(c,r)
              if(index .ne.-1) then
                 if ( (glbdata2(f,index) /= -9999.9) .and.  &
                      (glbdata2(f,index) < 0)) then
```

```
                   glbdata2(f,index) = (-1) * glbdata2(f,index)
                end if
                if ( (glbdata1(f,index) /= -9999.9) .and.  &
                     (glbdata1(f,index) < 0)) then
                   glbdata1(f,index) = (-1) * glbdata1(f,index)
                end if
             end if
          end do
       end do
    end if
 enddo
```

1.44.1 gdasfile (Source File: getgdas.F90)

This subroutine puts together GDAS file name for 3 hour file intervals

INTERFACE:

```
 subroutine gdasfile( name, gdasdir, yr, mo, da, hr )

   implicit none
```

INPUT PARAMETERS:

```
   character(len=80), intent(in)    :: gdasdir
   integer, intent(in)              :: yr, mo, da, hr
```

OUTPUT PARAMETERS:

```
   character(len=80), intent (out)  :: name
```

CONTENTS:

```
!-----------------------------------------------------------------
!  Make variables for the time used to create the file
!  We don't want these variables being passed out
!-----------------------------------------------------------------
  uyr = yr
  umo = mo
  uda = da
  uhr = 3*(hr/3)  !hour needs to be a multiple of 3 hours
  umn = 0
  uss = 0
  ts1 = -24*60*60 !one day interval to roll back date.
  remainder = modulo(uhr,2)  !if even, then remainder equals zero
                             !if odd, then remainder equals one
!-----------------------------------------------------------------
```

```
! if hour is   00 or 03, look for 00ZF00 or 00ZF03
! if hour is   06 or 09, look for 06ZF00 or 06ZF03
! if hour is   12 or 15, look for 12ZF00 or 12ZF03
! if hour is   18 or 21, look for 18ZF00 or 18ZF03
!-----------------------------------------------------------------
  if ( uhr <= 3 ) then
     initcode = '00'
  else if (uhr <= 9 ) then
     initcode = '06'
  else if ( uhr <= 15 ) then
     initcode = '12'
  else if ( uhr <= 21 ) then
     initcode = '18'
  end if
  if ( remainder > 0 ) then
     fcstcode = '03'
  else
     fcstcode = '00'
  end if

  write(UNIT=temp, fmt='(a40)') gdasdir
  read(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,80)

  write(UNIT=temp, fmt='(a1, i4, i2, a1)') '/', uyr, umo, '/'
  read(UNIT=temp, fmt='(8a1)') fdir
  do i = 1, 8
     if ( fdir(i) == ' ' ) fdir(i) = '0'
  end do

  write(UNIT=temp, fmt='(i4, i2, i2, a2)') uyr, umo, uda, initcode
  read(UNIT=temp, fmt='(10a1)') ftime
  do i = 1, 10
     if ( ftime(i) == ' ' ) ftime(i) = '0'
  end do

  write(UNIT=temp, fmt='(a16, a2, a3)') '.gdas1.sfluxgrbf', fcstcode, '.sg'
  read (UNIT=temp, fmt='(80a1)') (fsubs(i), i=1,21)

  c = 0
  do i = 1, 80
     if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  end do

  write(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,8),  &
                       (ftime(i), i=1,10), (fsubs(i), i=1,21)

  read(UNIT=temp, fmt='(a80)') name
```

```
  return
```

1.44.2 gdasfilef06 (Source File: getgdas.F90)

This subroutine puts together GDAS file name for 6 hour forecast files (DAAC archive friendly)

INTERFACE:

```
 subroutine gdasfileF06( name, gdasdir, yr, mo, da, hr )
```

USES:

```
   use time_manager

   implicit none
```

INPUT PARAMETERS:

```
   character(len=80) :: gdasdir
   integer :: yr, mo, da, hr
```

OUTPUT PARAMETERS:

```
   character(len=80) :: name
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, a2)
95 format (10a1)
96 format (a40)
97 format (a16, a2, a3)
98 format (a1, i4, i2, a1)
99 format (8a1)
!-----------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----------------------------------------------------------------
  uyr = yr
  umo = mo
  uda = da
  uhr = 6*(hr/6)  !hour needs to be a multiple of 6 hours
  umn = 0
  uss = 0
  ts1 = -24*60*60 !one day interval to roll back date.
!-----------------------------------------------------------------
```

```
! if hour is 00Z look for 18ZF06 after rolling back one day
! if hour is 06Z look for 00ZF06
! if hour is 12Z look for 06ZF06
! if hour is 18Z look for 12ZF06
!-----------------------------------------------------------------
  fcstcode = '06'
  if ( uhr == 0 ) then
     call tick( dumbtime, doy, gmt, uyr, umo, uda, uhr, umn, uss, ts1 )
     initcode = '18'
  else if ( uhr == 6 ) then
     initcode = '00'
  else if ( uhr == 12 ) then
     initcode = '06'
  else if ( uhr == 18 ) then
     initcode = '12'
  end if

  write(UNIT=temp, fmt='(a40)') gdasdir
  read(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,80)

  write(UNIT=temp, fmt='(a1, i4, i2, a1)') '/', uyr, umo, '/'
  read(UNIT=temp, fmt='(8a1)') fdir
  do i = 1, 8
     if ( fdir(i) == ' ' ) fdir(i) = '0'
  end do

  write(UNIT=temp, fmt='(i4, i2, i2, a2)') uyr, umo, uda, initcode
  read(UNIT=temp, fmt='(10a1)') ftime
  do i = 1, 10
     if ( ftime(i) == ' ' ) ftime(i) = '0'
  end do
  write(UNIT=temp, fmt='(a16, a2, a3)') '.gdas1.sfluxgrbf', '06', '.sg'
  read (UNIT=temp, fmt='(80a1)') (fsubs(i), i=1,21)
  c = 0
  do i = 1, 80
     if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  end do

  write(UNIT=temp, fmt='(80a1)') (fbase(i), i=1,c), (fdir(i), i=1,8),  &
                       (ftime(i), i=1,10), (fsubs(i), i=1,21)

  read(UNIT=temp, fmt='(a80)') name
  return
```

1.44.3 readgdascrd.F90 (Source File: readgdascrd.F90)

Routine to read GDAS specific parameters from the card file.

REVISION HISTORY:

```
   11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine readgdascrd(gdasdrv)
```

USES:

```
   use gdasdrv_module
 #if ( defined OPENDAP )
   use gdasopendap_module
 #endif
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=gdas)
  print*,'Using GDAS forcing'
  print*, 'GDAS forcing directory :',gdasdrv%GDASDIR
  gdasdrv%GDASTIME1  = 3000.0
  gdasdrv%GDASTIME2  = 0.0

  close(11)
#if ( defined OPENDAP )
  call opendap_gdas_init(gdasdrv)
#endif
```

1.44.4 retgdas.F90 (Source File: retgdas.F90)

Defines forcing parameters, retrieves the fields using calls to getgb, and interpolates the fields to LDAS specifications

REVISION HISTORY:

```
    14 Dec 2000: Urszula Jambor; Rewrote geteta.f to use GDAS forcing in GLDAS
    15 Mar 2001: Jon Gottschalck; Added additional parameters and octets in
                 which to search in GRIB files
    01 Jun 2001: Urszula Jambor; Added option to get forcing from different
                 files (F00 instantaneous and F06 six hour means)
    29 Jan 2003: Urszula Jambor; Rewrote code, uses GETGB call to replace
                 ungribgdas.  Interpolation now occurs in interp_gdas.
                 Using GETGB avoids problems with the Oct2002 GDAS
                 grid update.
```

```
    12 Nov 2003: Matt Rodell; Check to make sure input file exists before
   opening and thereby creating a new, empty file.
    14 Nov 2003: Matt Rodell; Ensure lugb varies in call to baopen
    05 Feb 2004: James Geiger; Added GrADS-DODS Server functionality
```

INTERFACE:

```
 #include "misc.h"
 subroutine retgdas( order, name, nameF06, F00flag,ferror,try )
```

USES:

```
   use lisdrv_module, only : lis, gindex
   use time_manager
   use baseforcing_module, only: glbdata1, glbdata2
   use gdasdomain_module, only : gdasdrv
   use spmdMod
 #if ( defined OPENDAP )
   use opendap_module, only : ciam
   use gdasopendap_module
 #endif
   use lis_indices_module, only : lis_nc_working, lis_nr_working
   implicit none
```

ARGUMENTS:

```
   integer, intent(in)           :: order
   character(len=80), intent(in) :: name, nameF06
   integer, intent(in)           :: F00flag
   integer, intent(out)          :: ferror
   integer, intent(inout)        :: try
```

CONTENTS:

```
  varfield = 0.0
  ngdas = (gdasdrv%ncold*gdasdrv%nrold)
!-----------------------------------------------------------------------
! Set the GRIB parameter specifiers
!-----------------------------------------------------------------------
  if ( masterproc ) then
     nstep = get_nstep(lis%t)
  endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
  call MPI_BCAST(nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,errorflag)
#endif
  if ( nstep == 0 ) then
     pds5 = (/011,051,204,205,033,034,001,059,214,084,144,144, 011,011, 065/) !parameter
     pds7 = (/002,002,000,000,010,010,000,000,000,000,010,2760,010,2760,000/) !htlev2
     nforce = gdasdrv%nmif
#if ( defined OPENDAP )
     forcing_index = (/15,13,4,3,18,21,9,8,2,1,11,12,16,17,23/)
```

```
#endif
  else
     pds5 = (/ 011,051,204,205,033,034,001,059,214,084 /) !parameter
     pds7 = (/ 002,002,000,000,010,010,000,000,000,000 /) !htlev2
     nforce = 10 ! for now
#if ( defined OPENDAP )
     forcing_index = (/15,13,4,3,18,21,9,8,2,1/)
#endif

  endif

  ferror = 1
!-----------------------------------------------------------------------
! if there's a problem then ferror is set to zero
!-----------------------------------------------------------------------
  iv = 0
  errorflag = 0
  endloop = 0
  allocate(lb(gdasdrv%ncold*gdasdrv%nrold))
  allocate(f(gdasdrv%ncold*gdasdrv%nrold))
  do
     if ( endloop == 1 ) exit
     iv = iv+1
     if ( (F00flag > 0)  .and. &
          (iv==3.or.iv==4.or.iv==8.or.iv==9.or.iv==10) ) then
        fname = nameF06
     else
        fname = name
     end if
#if ( defined OPENDAP )
!     print*,'DBG: retgdas -- fname = ',fname
!     inquire (file=fname, exist=file_exists)
!     if ( .not. file_exists ) then
        print*, 'MSG: retgdas -- Retrieving GDAS forcing file ', &
                trim(fname), ' (',iam,')'
        call system("opendap_scripts/getgdas.pl "//ciam//" "// &
                     trim(fname)//" "//                        &
                     cgdas_slat//" "//cgdas_nlat//" "//        &
                     cgdas_wlon//" "//cgdas_elon)
!        call perror("ERR: retgdas --")
!     endif
     open(unit=10,file=fname,access='sequential',form='unformatted')
     do j = 1, forcing_index(iv)-1
        read(10) dummy
     enddo
     read(10) f2d
     close(10)
     gbret=0
```

```
     ! Now make things look like grib data
     call twotoone(f2d,f,gdasdrv%ncold,gdasdrv%nrold)
     call get_kpds(kpds,iv)
     !call get_kgds(kgds)
     call get_gridDesc(gridDesc)
     if ( iv == 1 ) then
        call set_lb(f,lb,gdasdrv%ncold,gdasdrv%nrold)
     endif
#else
     inquire (file=fname, exist=file_exists)
     if (file_exists) then
!-----------------------------------------------------------------------------
! Set up to open file and retrieve specified field
!-----------------------------------------------------------------------------
        lugb = iv + try
        j = 0
        jpds = -1
        jpds(5) = pds5(iv)
        jpds(7) = pds7(iv)

        call baopen(lugb,fname,iret)
        if(iret==0) then
           call getgb(lugb,lubi,ngdas,j,jpds,jgds,kf,k,kpds,gridDesc,lb,f,gbret)
        else
           gbret = 99
        endif
        call baclose(lugb,jret)
     else
        ferror = 0
        deallocate(f)
        deallocate(lb)
        return
     endif
#endif
!-----------------------------------------------------------------------------
! If field successfully retrieved, interplate to LIS domain
!-----------------------------------------------------------------------------
     if (gbret==0) then
        call interp_gdas(kpds,ngdas,f,lb,lis%d%gridDesc, &
             lis_nc_working,lis_nr_working,varfield)
     else
        errorflag = 1
     endif

     if ( errorflag == 1 ) then
        endloop = 1
        ferror = 0
     else
```

```
        do c =1, lis_nc_working
           do r = 1, lis_nr_working
             if (gindex(c,r).ne. -1) then
              if ( order == 1 ) then
                 glbdata1(iv,gindex(c,r)) = varfield(c,r)
              else
                 glbdata2(iv,gindex(c,r)) = varfield(c,r)
              end if
             endif
           end do
        end do
     end if

     if ( errorflag == 1 ) then
        print *, 'Could not find correct forcing parameter in file',name
     end if
     if ( iv == nforce ) endloop = 1
  end do
  deallocate(lb)
  deallocate(f)
  return
```

1.44.5 interp_gdas (Source File: retgdas.F90)

This subroutine interpolates a given GDAS field to the LIS domain. Special treatment for some initialization fields. Code based on old ungribgdas.f

INTERFACE:

```
 subroutine interp_gdas(kpds,ngdas,f,lb,lis_gds,nc,nr, &
      varfield)
```

USES:

```
   use lisdrv_module, only : lis
   use bilinear_interpMod, only : w110,w120,w210,w220,n110,n120,n210,n220,&
        rlat0,rlon0
   use conserv_interpMod, only : w113,w123,w213,w223,n113,n123,n213,n223,rlat3,rlon3
   use gdasdomain_module, only : mi
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Setting interpolation options (ip=0,bilinear)
! (km=1, one parameter, ibi=1,use undefined bitmap
! (needed for soil moisture and temperature only)
! Use budget bilinear (ip=3) for precip forcing fields
!-----------------------------------------------------------------------
```

```
  mo = nc*nr
  if (kpds(5)==59 .or. kpds(5)==214) then
     ip=3
     ipopt(1)=-1
     ipopt(2)=-1
     km=1
     ibi=1
  else
     ip=0
     do i=1,20
        ipopt(i)=0
     enddo
     km=1
     ibi=1
  endif
!-----------------------------------------------------------------------
! Initialize output bitmap. Important for soil moisture and temp.
!-----------------------------------------------------------------------
  lo = .true.
!-----------------------------------------------------------------------
! Interpolate to LIS grid
!-----------------------------------------------------------------------
  if(lis%f%interp.eq.1) then
     call bilinear_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
          rlat0, rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
  elseif(lis%f%interp.eq.2) then
     if (kpds(5)==59 .or. kpds(5)==214)then
        call conserv_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo, &
             rlat3,rlon3,w113,w123,w213,w223,n113,n123,n213,n223,iret)
     else
        call bilinear_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
             rlat0, rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
     endif
  endif
!-----------------------------------------------------------------------
! Create 2D array for main program. Also define a "soil" mask
! due to different geography between GDAS & LDAS. For LDAS land
! points not included in GDAS geography dataset only.
!-----------------------------------------------------------------------
  count = 0
  do j = 1, nr
     do i = 1, nc
        varfield(i,j) = lis1d(i+count)
     enddo
     count = count + nc
  enddo
!  if(kpds(5).eq.1) then
!     open(232, file='forcing1.bin',form='unformatted')
```

```
!     write(232) varfield
!     close(232)
!  endif

!-----------------------------------------------------------------------
! Save air tempertaure interpolated field for later use in
! initialization of soil temp where geography differs
! between GDAS and LDAS
!-----------------------------------------------------------------------
!  if (kpds(5) .eq. 11 .and. kpds(6) .eq. 105) then
!     do i = 1, nc
!        do j = 1, nr
!           geogtemp(i,j) = varfield(i,j)
!        enddo
!     enddo
!  endif
```

1.44.6 time_interp_gdas (Source File: time_interp_gdas.F90)

Opens, reads, and interpolates GDAS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.45 Core Functions of time_interp_gdas

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
 1  Oct 1999: Jared Entin; Initial code
 25 Oct 1999: Jared Entin; Significant F90 Revision
 11 Apr 2000: Brian Cosgrove; Fixed name construction error
              in Subroutine ETA6HRFILE
 27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
              data with opposite sign convention from recent shortwave data.
              Added capability to use time averaged shortwave & longwave data
              Altered times which are passed into ZTERP--used to be GMT1
              and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
 30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
 17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
 13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
  5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
subroutine time_interp_gdas()
```

USES:

```
use lisdrv_module, only :lis, grid
use baseforcing_module, only: glbdata1, glbdata2
use time_manager
use grid_spmdMod
use spmdMod
use gdasdomain_module, only : gdasdrv
```

CONTENTS:

```
  if(masterproc) then
     if (get_nstep(lis%t) .eq. 0) then
        lis%f%nforce = gdasdrv%nmif
     else
        lis%f%nforce = lis%f%nf
     endif
  endif
#if(defined SPMD)
  call MPI_BCAST(gdasdrv%gdastime1,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(gdasdrv%gdastime2,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%time,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%nforce,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%F00_flag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%F06_flag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%rstflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
#endif
  btime=gdasdrv%gdastime1
  call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
  btime=gdasdrv%gdastime2
  call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
  if ( (lis%f%F06_flag==0) .and. (lis%f%F00_flag==1) ) then
     inittime = gdasdrv%gdastime2
     call time2date( inittime, idoy, igmt, iyr, imo, ida, ihr, imn )
     its = -6*60*60
     call tick( inittime, idoy, igmt,iyr,imo,ida,ihr,imn,iss,its)
  end if
```

```
  wt1 = (gdasdrv%gdastime2-lis%t%time) / &
       (gdasdrv%gdastime2-gdasdrv%gdastime1)
  wt2 = 1.0 - wt1
  do f=1,lis%f%nforce
     if ( f == 3 ) then  !shortwave
        do c = 1, gdi(iam)
           zdoy = lis%t%doy
!           print*, 'f0,f6',lis%f%F00_flag,lis%f%F06_flag
           if ( (lis%f%F06_flag==0) .and. (lis%f%F00_flag==1) ) then
              call zterp( 0, grid(c)%lat, grid(c)%lon, igmt, gmt2, &
                   lis%t%gmt,zdoy,zw1,zw2,czb,cze,czm,lis)
!              if(c==200) then
!                 print*, grid(c)%lat, grid(c)%lon
!                 print*, igmt
!                 print*, gmt2
!                 print*, lis%t%gmt
!                 print*, zdoy
!              endif
           else
              call zterp( 0, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
                   lis%t%gmt,zdoy,zw1,zw2,czb,cze,czm,lis)
!                 print*, 'blk2 ',c,zw1,zw2
           end if

           grid(c)%forcing(f) = zw1 * glbdata2(f,c)
!           if(c==200) print*, c,grid(c)%forcing(f),glbdata2(f,c),zw1
           if (grid(c)%forcing(f) < 0) then
              print *, '2 warning!!!  SW radiation is negative!!'
              print *, 'sw=', grid(c)%forcing(f), '... negative'
              print *, 'gdas2=', glbdata2(f,c)
              call endrun
           end if

           if (grid(c)%forcing(f).gt.1367) then
!              print*, 'in this high block..',c,f,grid(c)%forcing(f)
              grid(c)%forcing(f)=glbdata2(f,c)
           endif
        end do
     else if ( (f==4) .or. (f==10) ) then
        do c = 1, gdi(iam)
           if ( lis%f%F00_flag==1 ) then
              grid(c)%forcing(f)=2*glbdata2(f,c) -glbdata1(f,c)
           else
              grid(c)%forcing(f) = glbdata2(f,c)
           end if
        end do

     else if ( (f==8) .or. (f==9 ) ) then
```

```
      do c = 1, gdi(iam)
         if ( lis%f%F00_flag == 1) then
            if (2*glbdata2(f,c) >= glbdata1(f,c)) then
               grid(c)%forcing(f)=2*glbdata2(f,c)-glbdata1(f,c)
            else
               grid(c)%forcing(f) = 0.0
            end if
         else
            grid(c)%forcing(f) = glbdata2(f,c)
         end if
      end do
   else
      do c = 1, gdi(iam)
         grid(c)%forcing(f) = wt1 * glbdata1(f,c) +  &
              wt2 *glbdata2(f,c)
      end do
   end if
end do
return
```

1.46 Fortran: Module Interface ecmwfdomain_module.F90 (Source File: ecmwfdomain_module.F90)

Contains routines and variables that define the native domain for ECMWF model forcing.

INTERFACE:

```
module ecmwfdomain_module
```

USES:

```
use ecmwfdrv_module
```

ARGUMENTS:

```
type(ecmwfdrvdec) :: ecmwfdrv
integer :: mi
```

1.46.1 defnatecmwf.F90 (Source File: ecmwfdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for ECMWF data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatecmwf(gridDesci)
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

CONTENTS:

```
call readecmwfcrd(ecmwfdrv,gridDesci)
mi = ecmwfdrv%ncold*ecmwfdrv%nrold
```

1.47 Fortran: Module Interface ecmwfdrv_module.F90 (Source File: ecmwfdrv_module.F90)

Module containing runtime specific ECMWF variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module ecmwfdrv_module
```

ARGUMENTS:

```
type ecmwfdrvdec
   integer      :: ncold, nrold   !AWIPS 212 dimensions
   integer      :: nmif
   character*40 :: ecmwfdir   !ecmwf Forcing Directory
   real*8       :: ecmwftime1,ecmwftime2
end type ecmwfdrvdec
```

1.48 Fortran: Module Interface ecmwfopendap_module.F90 (Source File: ecmwfopendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O specfific to ECMWF forcing routines

REVISION HISTORY:

```
10 Jul 2003; James Geiger  Initial Specification
22 Dec 2003; Sujay Kumar Separted ECMWF specific routines from the main
             module
```

INTERFACE:

```
 module ecmwfopendap_module
 #if ( defined OPENDAP )
   TODO: Stubs to allow the code to compile
   TODO: ECMWF does not yet work in GDS-mode
    contains
    subroutine opendap_ecmwf_init()
    end subroutine opendap_ecmwf_init
    subroutine def_gridDesc(gridDesci)
       real :: gridDesci(50)
    end subroutine def_gridDesc
 #endif
 #if ( defined OPENDAP_KLUGE )
   TODO: Correct this section
```

USES:

```
   use ecmwfdomain_module, only : ecmwfdrv
   use ESMF_TimeMgmtMod
   use lisdrv_module, only : lis, grid, gindex
   use grid_spmdMod,   only : gdi, gdisp
   use tile_spmdMod,   only : di_array
   use spmdMod

   implicit none
    type(esmf_date), save :: ecmwf_ref_date !Reference date for ECMWF forcing data
    integer               :: ecmwf_slat     !Southern latitude boundary for each subdomain (fo
    integer               :: ecmwf_nlat     !Northern latitude boundary for each subdomain (fo
    integer               :: ecmwf_nc       !Number of columns for each subdomain (for ECMWF)
    integer               :: ecmwf_nr       !Number of rows for each subdomain (for ECMWF)
    integer               :: input_slat     !Southern latitude boundary for the native domain
    integer               :: input_nlat     !Northern latitude boundary for the native domain
    character*4           :: cecmwf_slat    !character representation of ecmwf$_-$slat
    character*4           :: cecmwf_nlat    !character representation of ecmwf$_-$nlat
    character*3           :: ciam           !character representation of processor id
    integer               :: grid_offset    !Global to local grid mapping offset
    integer               :: fnroffset

 #if ( defined ABSOFT )
    character*15 :: opendap_ecmwf_home = '/home/jim/'
 #else
    character*2  :: opendap_ecmwf_home = './'
 #endif
    contains
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\
```

```
\subsubsection{opendap\_ecmwf\_init (Source File: ecmwfopendap\_module.F90)}



    Initializes the ECMWF-GDS variables

\bigskip{\sf INTERFACE:}
\begin{verbatim}    subroutine opendap_ecmwf_init()
```

CONTENTS:

```
     call init_ecmwf_vars()
     call reset_ecmwf_filepaths()
```

1.48.1 reset_ecmwf_filepaths (Source File: ecmwfopendap_module.F90)

Resets input data filenames for ECMWF forcing for execution through GDS

INTERFACE:

```
      subroutine reset_ecmwf_filepaths()
```

CONTENTS:

```
       ecmwfdrv%ecmwfdir = trim(opendap_ecmwf_home)//trim(adjustl(ciam))//'/'//ecmwfdrv%ecmwfdi
```

1.48.2 init_ecmwf_vars (Source File: ecmwfopendap_module.F90)

Computes domain decomposition for native as well as interpolated domains for input ECMWF forcing

INTERFACE:

```
     subroutine init_ecmwf_vars()
```

CONTENTS:

```
      if(lis%d%gridDesc(9) .eq. 0.01) then
         domain = 8
      elseif(lis%d%gridDesc(9).eq.0.05) then
         domain = 7
      elseif(lis%d%gridDesc(9) .eq. 0.125) then
         domain = 6
      elseif(lis%d%gridDesc(9) .eq. 0.25) then
         domain = 5
      elseif(lis%d%gridDesc(9) .eq. 0.50) then
         domain = 4
```

```
elseif(lis%d%gridDesc(9) .eq. 1.0) then
   domain = 3
elseif((lis%d%gridDesc(9) .eq. 2) .and.  &
     (lis%d%gridDesc(10) .eq. 2.5)) then
   domain = 2
endif
select case (domain)
case(1)
   print*,'Error!  Cannot handle ndas.'
   stop 999
case(2)
   res = 250
   center = 875
   bottom = -59875
case(3)
   print*,'Error!  Cannot handle 2x2.5.'
   stop 999
case(4)
   res = 1000
   center = 500
   bottom = -59500
case(5)
   res = 500
   center = 750
   bottom = -59750
case(6)
   res = 50
   center = 975
   bottom = -59975
case DEFAULT
   print*, "Select domain size (1,2,3,4,5,6)"
   stop 999
end select

call set_ecmwf_lat(ecmwf_slat,ecmwf_nlat,input_slat,input_nlat)

lis%d%ngrid = gdi(iam)
lis%d%nch   = di_array(iam)

ecmwf_nc    = 1440
ecmwf_nr    = ( ecmwf_nlat - ecmwf_slat + 1 )

fnroffset = ecmwf_slat - 1
grid_offset = tile(1)%index-1

write(ciam, '(i3)') iam
write(cecmwf_slat, '(i4)') ecmwf_slat
write(cecmwf_nlat, '(i4)') ecmwf_nlat
```

```
print*,'DBG: ecmwf_init -- ecmwf_slat', ecmwf_slat, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ecmwf_nlat', ecmwf_nlat, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ngrid', lis%d%ngrid, ' (', iam, ')'
print*,'DBG: ecmwf_init -- glbngrid', lis%d%glbngrid, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ncold', ecmwfdrv%ncold, ' (', iam, ')'
print*,'DBG: ecmwf_init -- nrold', ecmwfdrv%nrold, ' (', iam, ')'
print*,'DBG: ecmwf_init -- lnc', lis%d%lnc, ' (', iam, ')'
print*,'DBG: ecmwf_init -- lnr', lis%d%lnr, ' (', iam, ')'
print*,'DBG: ecmwf_init -- fnroffset', fnroffset, ' (', iam, ')'
print*,'DBG: ecmwf_init -- grid_offset', grid_offset, ' (', iam, ')'
print*,'DBG: ecmwf_init -- cecmwf_slat ', cecmwf_slat, ' (', iam, ')'
print*,'DBG: ecmwf_init -- cecmwf_nlat ', cecmwf_nlat, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ciam ', ciam, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ecmwf_nc', ecmwf_nc, ' (', iam, ')'
print*,'DBG: ecmwf_init -- ecmwf_nr', ecmwf_nr, ' (', iam, ')'
line = '"'//ciam//'"'
print*,'DBG: ecmwf_init -- line', line, ' (', iam, ')'
```

1.48.3 def_gridDesc (Source File: ecmwfopendap_module.F90)

Initializes the gridDesc array for ECMWF-GDS runs.

INTERFACE:

```
subroutine def_gridDesc(gridDesci)
```

CONTENTS:

```
gridDesci(1) = 0
gridDesci(2) = ecmwf_nc
gridDesci(3) = ecmwf_nr
gridDesci(4) = input_slat
gridDesci(7) = input_nlat
gridDesci(5) = -180.000
gridDesci(6) = 128
gridDesci(8) = 179.750
gridDesci(9) = .250
gridDesci(10) = .250
gridDesci(20) = 255
```

1.48.4 init_ecmwf_ref_date (Source File: ecmwfopendap_module.F90)

Initializes the reference date for ECMWF forcing data

INTERFACE:

```
subroutine init_ecmwf_ref_date()
```

CONTENTS:

```
ecmwf_ref_date = esmf_dateinit(esmf_no_leap, ref_ymd, ref_tod, rc)
```

1.48.5 get_ecmwf_index (Source File: ecmwfopendap_module.F90)

Computes the time-based index for ECMWF forcing data

INTERFACE:

```
function get_ecmwf_index(offset)
  implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: offset   ! offset from current date in hours
```

CONTENTS:

```
if ( ref_data_uninit ) then
   print*,'DBG: get_ecmwf_index -- initializing ref date',' (', iam, ')'
   call init_ecmwf_ref_date()
   ref_data_uninit = .false.
endif
ymd = ( lis%t%yr * 10000 ) + ( lis%t%mo * 100 ) + lis%t%da
tod = ( lis%t%hr *  3600 ) + ( lis%t%mn *  60 ) + lis%t%ss
current_date = esmf_dateinit(esmf_no_leap, ymd, tod, rc)

diff = esmf_timeinit()
call esmf_datediff(current_date, ecmwf_ref_date, diff, islater, rc)
call esmf_timeget(diff, ndays, nsecs, rc)

get_ecmwf_index = ( (ndays * 24) + (nsecs / 3600) + offset ) / 3 + 1
print*,'DBG: get_ecmwf_index -- get_ecmwf_index = ', get_ecmwf_index, &
     ' (', iam, ')'
```

1.48.6 set_ecmwf_lat (Source File: ecmwfopendap_module.F90)

Computes the latitudes of the decomposed domain for ECMWF forcing data

INTERFACE:

```
subroutine set_ecmwf_lat(slat, nlat, islat, inlat)
   implicit none
```

OUTPUT PARAMETERS:

```
       integer, intent(out) :: slat, nlat, islat, inlat
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Set southern latitude index
!-----------------------------------------------------------------------
      print*, 'DBG: set_ecmwf_lat -- grid(1)%lat', grid(1)%lat,' (', iam, ')'
      lat = grid(1)%lat
      if ( lat < 0.0 ) then
         slat = int(lat) - 1 ! E.g. map -54.5 \to -55
      else
         slat = int(lat)     ! E.g. map  40.5 \to  40
      endif
      if ( slat < -90 ) then
         print*, 'ERR: set_ecmwf_lat -- Setting slat = -90', &
                 ' (', iam, ')'
         slat = -90
      endif
!-----------------------------------------------------------------------
! Set southern latitude boundary
!-----------------------------------------------------------------------
      islat = 250 * slat
!-----------------------------------------------------------------------
! Set northern latitude index
!-----------------------------------------------------------------------
      print*, 'DBG: set_ecmwf_lat -- grid(gdi(iam))', &
              gdi(iam), grid(gdi(iam))%lat,' (', iam, ')'
      lat = grid(gdi(iam))%lat
      if ( lat < 0.0 ) then
         nlat = int(lat)     ! E.g. map -10.5 \to -10
      else
         nlat = int(lat) + 1 ! E.g. map  10.5 \to  11
      endif
      if ( nlat > 90 ) then
         print*, 'ERR: set_ecmwf_lat -- Setting nlat = 90', &
                 ' (', iam, ')'
         nlat = 90
      endif
!-----------------------------------------------------------------------
! Set northern latitude boundary
!-----------------------------------------------------------------------
      inlat = 250 * nlat

      slat = slat + 90 + 0.25 ! map [-90,90] \to [1,181]
      nlat = nlat + 90 + 0.25 ! map [-90,90] \to [1,181]
      slat = 1
```

```
nlat = 601
islat = -90000
inlat = 90000
```

1.48.7 getecmwf.F90: (Source File: getecmwf.F90)

Opens, reads, and interpolates 10 LDAS forcing fields from operational ECMWF model output.
TIME1 = most recent past data TIME2 = nearest future data
The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day. Forcing fields are read in depending on data type. Time integrated data are treated differently than instantaneous data.

REVISION HISTORY:

```
18 Jun 2003: Urszula Jambor; original code based on getreanlecmwf.F90
```

INTERFACE:

```
subroutine getecmwf()
```

USES:

```
use lisdrv_module, only : lis, gindex
use baseforcing_module, only : glbdata1, glbdata2
use time_manager
use ecmwfdomain_module, only : ecmwfdrv
```

1.48.8 readecmwfcrd.F90 (Source File: readecmwfcrd.F90)

Routine to read ECMWF specific parameters from the card file.

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readecmwfcrd(ecmwfdrv,gridDesci)
```

USES:

```
  use ecmwfdrv_module
#if ( defined OPENDAP )
    use ecmwfopendap_module, only : opendap_ecmwf_init, &
         def_gridDesc
#endif
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=ecmwf)
  print*,'Using ECMWF forcing'
  print*, 'ECMWF forcing directory :',ecmwfdrv%ecmwfDIR
  ecmwfdrv%ecmwftime1 = 3000.0
  ecmwfdrv%ecmwftime2 = 0.0

#if ( defined OPENDAP )
    call opendap_ecmwf_init()
    call def_gridDesc(gridDesci)
#endif
#if (! defined OPENDAP)
    gridDesci(1) = 0
    gridDesci(2) = ecmwfdrv%ncold
    gridDesci(3) = ecmwfdrv%nrold
    gridDesci(4) = 90.000
    gridDesci(5) = -180.000
    gridDesci(6) = 128
    gridDesci(7) = -60.000
    gridDesci(8) = 179.750
    gridDesci(9) = 0.250
    gridDesci(10) = 0.250
    gridDesci(20) = 255
#endif
  close(11)
```

1.48.9 retecmwf (Source File: retecmwf.F90)

ECMWF model output variables used to force LDAS fall into 2 categories: –¿ inst3, Instantaneous values, available every 3 hours –¿ accum, Time integrated values (accumulations), updated every 3 hours
NOTE-1: be aware that ECMWF outputs large-scale and convective precipitation separately. For total precipitation, need to sum the two fields, LSP+CP=TP. Reversed traditional order of model-forcing precip fields for easier processing – interchange at end of routine to preserve traditional order. NOTE 2: only time2 SW flux accumulations used in interpolation NOTE-3: read in of Albedo is currently suppressed. This field is instantaneous and available every 6 hours. At this time, all LDAS LSMs replace this parameter.
ECMWF FORCING VARIABLES: 1. T inst3, near-surfcae temperature, 10 metres [K] 2. Q inst3, near-surface specific humidity, 10 metres[kg/kg] 3. SSRD accum, surface solar radiation downwards [W m**-2 s] 4. STRD accum, surface thermal radiation downwards [W m**-2 s] 5. U inst3, zonal wind, 10 metres [m/s] 6. V inst3, meridional wind, 10 metres[m/s] 7. SP inst3, surface pressure [Pa] 8. CP accum, convective precipitation [m] 9. LSP accum, large scale precipitation [m]

REVISION HISTORY:

```
18 Jun 2003: Urszula Jambor; original code
```

INTERFACE:

```
subroutine retecmwf( order, yr, mon, da, hr, ferror )
```

USES:

```
use lisdrv_module, only : lis, grid, gindex
use baseforcing_module, only : glbdata1, glbdata2
use time_manager
use ecmwfdomain_module, only : ecmwfdrv
```

1.48.10 ret_inst3 (Source File: retecmwf.F90)

This routine opens the corresponding ECMWF data file to extract the specified variable, which represents an instantaneous value. Should be used for near-surface temperature, specific humidity, winds, and surface pressure.
INPUT: dir – directory path name for ECMWF data id – index indicates forcing variable (T,Q,U,V,SP) yr,mo,da,hr – date information necwmf – number of elements of input grid OUTPUT: f – 1D forcing field, necmwf elements gridDesc – GDS array describing grid lb – unpacked bitmap fret – integer error return flag INTERFACE:

```
subroutine ret_inst3(dir,yr,mo,da,hr,necmwf,f,id,gridDesc,lb,fret)
```

1.48.11 time_interp_ecmwf.F90 (Source File: time_interp_ecmwf.F90)

Opens, reads, and interpolates ECMWF forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.49 Core Functions of time_interp_geos

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
1  Oct 1999: Jared Entin; Initial code
25 Oct 1999: Jared Entin; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; Fixed name construction error
             in Subroutine ETA6HRFILE
27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
             data with opposite sign convention from recent shortwave data.
```

```
                 Added capability to use time averaged shortwave & longwave data
                 Altered times which are passed into ZTERP--used to be GMT1
                 and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
    30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
    17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
    13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
     5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
 subroutine time_interp_ecmwf()
```

USES:

```
   use lisdrv_module, only : lis, grid
   use baseforcing_module, only : glbdata1, glbdata2
   use time_manager
   use grid_spmdMod
   use spmdMod
   use ecmwfdomain_module, only : ecmwfdrv
```

CONTENTS:

```
  if(masterproc) then
     if (get_nstep(lis%t) .eq. 0) then
        lis%f%nforce = lis%f%nmif
     else
        lis%f%nforce = lis%f%nf
     endif
  endif
#if(defined SPMD)
  call MPI_BCAST(ecmwfdrv%ecmwftime1,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(ecmwfdrv%ecmwftime2,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%time,1,MPI_REAL8,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%nforce,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%f%rstflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
#endif

  !=== Reset GMT times
  btime=ecmwfdrv%ecmwftime1
  call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
  btime=ecmwfdrv%ecmwftime2
  call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
```

```
!=== Need lower SW time boundary for call to zterp based on time2
inittime=btime
call time2date( inittime, idoy, igmt, iyr, imo, ida, ihr, imn )
select case (ihr)
case(00,12,24)
   its = -12*60*60
case(03,15)
   its =  -3*60*60
case(06,18)
   its =  -6*60*60
case(09,21)
   its =  -9*60*60
end select
call tick( inittime, idoy, igmt, iyr, imo, ida, ihr, imn, iss, its )

!===  Interpolate Data in Time
wt1=(ecmwfdrv%ecmwftime2-lis%t%time)/ &
     (ecmwfdrv%ecmwftime2-ecmwfdrv%ecmwftime1)
wt2=1.0-wt1
do f=1,lis%f%nforce
   if (f == 3) then     ! Time Averaged Shortwave
      do c = 1, gdi(iam)
         zdoy = lis%t%doy
         call zterp(0,grid(c)%lat,grid(c)%lon, &
              igmt,gmt2,lis%t%gmt,zdoy, &
              zw1,zw2,czb,cze,czm,lis)
         grid(c)%forcing(f) = zw1 * glbdata2(f,c)
         if ((grid(c)%forcing(f).ne.lis%d%udef).and.  &
              (grid(c)%forcing(f).lt.0)                ) then
            if (grid(c)%forcing(f) > -0.00001) then !arbitrary!!
               grid(c)%forcing(f) = 0.0             !threshold!!
            else
               print*, 'Stopping because SW forcing not udef but lt 0, '
               print*, f,c,r,grid(c)%forcing(f)
               stop
            endif
         endif
         if (grid(c)%forcing(f).gt.1367) then
            print*, '!!! ',grid(c)%forcing(f)
            print*, '!!! zterp produced d-SW-flux > solar constant!!!'
            stop
            grid(c)%forcing(f) = glbdata2(f,c)
         endif
      enddo ! c

   else if (f == 4) then
      do c=1,gdi(iam)
         grid(c)%forcing(f)=glbdata2(f,c)
```

```
        enddo

     else if (f == 8 .or. f == 9) then    ! precip variable Block Interpolation
        do c=1,gdi(iam)
           grid(c)%forcing(f)=glbdata2(f,c)
        enddo
     else     !Linearly interpolate everything else
        do c=1,gdi(iam)
           grid(c)%forcing(f)=glbdata1(f,c)*wt1+glbdata2(f,c)*wt2
        enddo
     endif
  enddo   !the f loop

end subroutine time_interp_ecmwf

\markboth{Left}{Source File: bergdomain\_module.F90,  Date: Fri Feb 11 15:41:04 EST 2005
}

!-------------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED !WARRANTY.NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-------------------------------------------------------------------------
```

1.50 Fortran: Module Interface bergdomain_module.F90 (Source File: bergdomain_module.F90)

Contains routines and variables that define the native domain for BERG model forcing

INTERFACE:

```
 module bergdomain_module
```

USES:

```
use bergdrv_module
```

1.50.1 defnatberg.F90 (Source File: bergdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for BERG data.

REVISION HISTORY:

```
26Jan2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatberg(gridDesci)
```

USES:

```
  implicit none
```

ARGUMENTS:

```
  real, intent(inout) :: gridDesci(50)
```

CONTENTS:

```
 call readbergcrd(bergdrv)
 gridDesci(1) = 0
 gridDesci(2) = bergdrv%ncold
 gridDesci(3) = bergdrv%nrold
 gridDesci(4) = -89.750
 gridDesci(5) = -179.750
 gridDesci(6) = 128
 gridDesci(7) = 89.750
 gridDesci(8) = 179.750
 gridDesci(9) = 0.500
 gridDesci(10) = 0.500
 gridDesci(20) = 255
 call readbergmask(bergdrv%remask1d)
 mi = bergdrv%ncold*bergdrv%nrold
 print*,'defnatberg mi ',mi
```

1.51 Fortran: Module Interface bergdrv_module.F90 (Source File: bergdrv_module.F90)

Module containing runtime specific BERG variables

REVISION HISTORY:

```
26 Jan 2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
module bergdrv_module
```

ARGUMENTS:

```
type bergdrvdec
   integer      :: ncold, nrold   !AWIPS 212 dimensions
   integer      :: nmif
   real*8       :: fmodeltime1,fmodeltime2
   integer      :: remask1d(720*360)
   character*40 :: emaskfile !1/2deg Reanal-BERG Land-Sea Mask File
end type bergdrvdec
```

1.51.1 geogfill2 (Source File: geogfill2.F90)

Fill in grid points that are invalid in LIS due to differences in geography between forcing and land surface.
Based on original geogfill.F90 for use with Reanalysis forcing fdata, to account for differences in land masks. NOTE** for Reanalysis fdata, v=6 is the V-component of wind, which is always set to ZERO since the U-component (v=5) is assigned the absolute value of wind speed.
For v=1:temperature, v=2:specific humidity, v=4:LW radiation, v=5:wind, and v=7:pressure, data values of zero are not allowed to contribute to average fill-value. For v=3:SW radiation, v=8,9:precipitation, use the mask defined by valid temperature data points to establish whether to seek fill-value for given column and row, rather than include vs exclude zeroes.
REVISION HISTORY: 20 Sep 2002: Urszula Jambor; Modified original geogfill routine for use with Reanalysis forcing sets prepared by Aaron Berg via NSIPP 29 Jan 2003: Urszula Jambor; Removed LDAS & GRID modules from list of arguments, instead pass only needed variables directly (nc,nr,fimask). 27 Jan 2004: Matt Rodell; Make sure all points are filled, rename data array to fdata. INTERFACE:

```
subroutine geogfill2(nc,nr,geogmask,fdata,v,tmask)
```

USES:

```
use lisdrv_module, only : gindex
```

1.51.2 getberg.F90 (Source File: getberg.F90)

Opens, reads, and interpolates 6-hrly, 1/2 degree Reanalysis ECMWF forcing.
TIME1 = most recent past data TIME2 = nearest future data
The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

REVISION HISTORY:

```
11 Apr 2002: Urszula Jambor; original code based on getgeos.f
22 Oct 2002: Urszula Jambor; Limited SW forcing processing to
             land-only grid points
24 Nov 2003: Sujay Kumar; Included ECMWF code in LIS
```

INTERFACE:

```
subroutine getberg()
```

USES:

```
use lisdrv_module, only : lis    ! LIS non-model-specific 1-D variables
use baseforcing_module, only : glbdata1,glbdata2
use bergdomain_module, only : bergdrv
use time_manager
```

1.51.3 readbergcrd.F90 (Source File: readbergcrd.F90)

Routine to read BERG specific parameters from the card file.

REVISION HISTORY:

```
26Jan2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readbergcrd(bergdrv)
```

USES:

```
use bergdrv_module
use lisdrv_module, only : lis
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=berg)
print*,'Using BERG forcing'
bergdrv%fmodeltime1 = 3000.0
bergdrv%fmodeltime2 = 0.0
bergdrv%nmif = lis%f%nmif
close(11)
```

1.51.4 recmwfmask.F90: (Source File: readbergmask.F90)

Reads in land-sea mask for 0.5 degree Reanalysis ECMWF data set, changes grid from ECMWF convention to GLIS convention by calling Recmwfgrid_2_gldasgrid, and transforms grid array into 1D vector for later use in ret_reanlecmwf.F90.

REVISION HISTORY:

```
16 Apr 2002: Urszula Jambor; Initial Code
24 Nov 2003: Sujay Kumar; Included in LIS
```

INTERFACE:

```
subroutine readbergmask()
```

USES:

```
use lisdrv_module, only :lis    ! LIS non-model-specific 1-D variables
use bergdomain_module, only : bergdrv
```

1.51.5 retberg.F90 (Source File: retberg.F90)

For the given time, reads in 8 parameters from 1/2 degree Reanalysis ECMWF data, transforms into 9 GLDAS forcing parameters and interpolates to the LDAS domain.
Reanal. ECMWF FORCING VARIABLES available every 6 hours: 1. 2T 2 metre air temperature [K], instantaneous 2. 2D 2 metre dew point temperature [K], instantaneous 3. SSRD Downward shortwave flux, surface [W/m^2 s], accumulation 4. STRD Downward longwave radiation, surface [W/m^2 s], accumulation 5. WIND Calculated absolute 10 metre wind speed [m/s], instantaneous 6. P Calculate surface pressure [Pa], instantaneous 7. LSP+CP Total precipitation [m], accumulation 8. CP Convective precipatation [m], accumulation

REVISION HISTORY:

```
09 Apr 2002: Urszula Jambor; Original code, based on readgeos.f
20 Dec 2002: Matt Rodell; Don't ipolate if running 1/2 deg, also fix
             error in setting v-wind to zero
25 Mar 2003: Urszula Jambor; Modified argument list passed to GEOGFILL2.
24 Nov 2003; Sujay Kumar; Included in LIS
15 Mar 2004; Matt Rodell; Fix test for beginning of restart run
```

INTERFACE:

```
subroutine retberg(order, yr, mon, da, hr, ferror)
  USES:
  use lisdrv_module, only : lis, grid,gindex
  use baseforcing_module, only: glbdata1, glbdata2
  use bergdomain_module, only : bergdrv,mi
  use bilinear_interpMod, only : w110,w120,w210,w220,n110,n120,n210,n220, &
       rlat0,rlon0
  use spmdMod, only : iam
```

1.51.6 berggrid_2_gldasgrid (Source File: retberg.F90)

Changes grid_data from ECMWF data convention to GLDAS convention
ECMWF: North-to-South around Greenwich Meridian Global grid. Data are written as flat binary from "upper left to lower right" starting at 0.5-degree grid point center coordinates: 0.25E,89.75N and going to 0.25W,89.75S. Here is the write statement:
do i = 1,360 write(14) (val(j,i),j=1,720) end do
GLDAS: South-to-North around Date Line Full global grid. Starts at the southernmost latitude and date line, going east and then north.

REVISION HISTORY:

```
10 Apr 2002: Urszula Jambor;  Code adapted from
             ecmwfgrid_2_grid2catgrid, by R. Reichle
```

INTERFACE:

```
subroutine berggrid_2_gldasgrid( nx, ny, grid_data )
```

1.51.7 fillgaps (Source File: retberg.F90)

Fills in values for NSIPP tilespace land points where no ECMWF reanalysis data is available via GEOGFILL by assigning most appropriate land-point value along the latitudinal circle of original tilespace point. Developed manually with 17 points in mind.

REVISION HISTORY:

```
23 Jul 2002: Urszula Jambor
03 Sep 2002: Urszula Jambor, revised points to reflect
             NSIPP land mask correction.
```

INTERFACE:

```
subroutine fillgaps( nc, nr, v, arr )
```

CONTENTS:

```
arr( 55, 1) = arr( 59, 4)
arr( 62, 2) = arr( 59, 4)
arr( 88, 8) = arr( 46, 8)
arr( 94, 8) = arr( 46, 8)
arr(  2, 9) = arr(142, 9)
arr(  3,20) = arr(133,20)
arr(139,20) = arr(133,20)
arr(140,20) = arr(133,20)
arr( 95,29) = arr( 89,29)
arr( 37,30) = arr( 41,30)
arr( 36,31) = arr( 41,31)
arr( 37,31) = arr( 41,31)
arr(136,34) = arr(123,34)
```

```
  arr( 62,50) = arr( 69,50)
  arr( 63,50) = arr( 69,50)
  arr(142,57) = arr(144,57)
  arr( 70,67) = arr( 83,67)

  if (v == 3) then
    arr( 88, 8) = arr( 84,14)
    arr( 94, 8) = arr( 84,14)
  endif

end subroutine fillgaps
```

!————————————————————————————————————- ! NASA GSFC Land Information Systems LIS 2.0 ! ! Released May 2004 ! ! See SOFTWARE DISTRIBUTION POLICY for software distribution policies ! ! The LIS source code and documentation are in the public domain, ! available without fee for educational, research, non-commercial and ! commercial purposes. Users may distribute the binary or source ! code to third parties provided this statement appears on all copies and ! that no charge is made for such copies. ! ! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE ! SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED AS IS WITHOUT EXPRESS OR ! IMPLIED !WARRANTY.NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE ! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE. ! ! See COPYRIGHT.TXT for copyright details. ! !————————————————————————————————————- #include ¡misc.h¿

1.52 Fortran: Module Interface time_interp_berg (Source File: time_interp_berg.F90)

Opens, reads, and interpolates BERG forcing.
TIME1 = most recent past data TIME2 = nearest future data
The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

REVISION HISTORY:

```
24 Nov 2003: Sujay Kumar; Initial version of code adapted from the GLDAS
                          version
```

INTERFACE:

```
subroutine time_interp_berg()
```

USES:

```
use lisdrv_module, only : lis, grid
use baseforcing_module, only : glbdata1,glbdata2
use time_manager
use bergdomain_module, only : bergdrv
use grid_spmdMod
use spmdMod
```

1.52.1 getnldas.F90 (Source File: getnldas.F90)

Opens, reads, and interpolates NCEP-LDAS forcing.
TIME1 = most recent past data TIME2 = most recent future data
The strategy for missing data is to backwards up to 10 days to get forcing at the same time of day.

REVISION HISTORY:

```
 1  Oct 1999: Jared Entin; Initial code
 15 Oct 1999: Paul Houser; Significant F90 Revision
 20 Dec 1999: Paul Houser; Allow for Eta Data Overwrite by NCEP data
 27 Apr 2000: Brian Cosgrove; Turned zenith angle weighting back on.
              Changed times supplied to ZTERP from GMT1 and GMT2 to
              LDAS%NLDASTIME1 and LDAS%NLDASTIME2
  4 May 2000: Added 15 minutes to GMT1 and GMT2 to accurately
              reflect the valid times of the NCEP radiation data
 18 Aug 2000: Brian Cosgrove; Fixed error in date calculations so that
              3600 second (1hr) timestep may be used.
              Added code to make any calculated radiation forcing
              values undefined if both ncep time 1 and ncep time 2
              radiation values are undefined
 27 Feb 2001: Brian Cosgrove; Added CZM into call for ZTERP subroutine
 07 Mar 2001: Brian Cosgrove; Added code to allow for use of NASA-LDAS data
 04 Sep 2001: Brian Cosgrove; Changed tempgmt1,tempgmt2 to real to match
              tick.f call, changed file name construction.
 21 Aug 2002: Brian Cosgrove; Removed code that adjusted for 15 minute
              offset in radiation fields supplied by NOAA or NASA
              NLDAS realtime or retrospective standard forcing files.
              This offset no longer exists as it is now dealt with
              during forcing file creation through zenith angle correction.
              The need for this fix was just discovered...unfortunately
              simulations before the date of this fix using
              this fortran subroutine have incorrectly shifted
              radiation data.
02Feb 2004 : Sujay Kumar ; Initial Version in LIS
```

INTERFACE:

```
subroutine getnldas()
```

USES:

```
use lisdrv_module, only : lis,grid
use baseforcing_module, only : glbdata1,glbdata2
use nldasdomain_module, only : nldasdrv
use time_manager, only : tick
```

1.52.2 ncepfile.f: (Source File: getnldas.F90)

This subroutine puts together the ncep data filename

REVISION HISTORY:

```
1  Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
04 Sep 2001: Brian Cosgrove; Use of NASA data enabled, updated
             reading of data directory structure to read new format
```

INTERFACE:

```
subroutine ncepfile(name,ncepdir,yr,mo,da,hr)
```

1.53 Fortran: Module Interface nldasdomain_module.F90 (Source File: nldasdomain_module.F90)

Contains routines and variables that define the native domain for NLDAS model forcing.

INTERFACE:

```
module nldasdomain_module
```

USES:

```
use nldasdrv_module
```

ARGUMENTS:

```
type(nldasdrvdec) :: nldasdrv
integer :: mi
```

1.53.1 defnatnldas.F90 (Source File: nldasdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for NLDAS data.

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatnldas(gridDesci)
```

USES:

```
use lisdrv_module, only: lis
use time_manager, only : date2time
implicit none
```

CONTENTS:

```
call readnldascrd(nldasdrv,gridDesci)
mi = nldasdrv%ncold*nldasdrv%nrold
```

1.54 Fortran: Module Interface nldasdrv_module.F90 (Source File: nldasdrv_module.F90)

Module containing runtime specific NLDAS variables

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
module nldasdrv_module
```

ARGUMENTS:

```
type nldasdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   integer :: nmif
   character*40 :: nldasdir   !NLDAS Forcing Directory
   real*8 :: nldastime1,nldastime2
end type nldasdrvdec
```

1.54.1 readnldascrd.F90 (Source File: readnldascrd.F90)

Routine to read NLDAS specific parameters from the card file.

REVISION HISTORY:

```
02Feb2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readnldascrd(nldasdrv,gridDesci)
```

USES:

```
use nldasdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=nldas)
print*,'Using NLDAS forcing'
print*, 'NLDAS forcing directory :',nldasdrv%NLDASDIR
nldasdrv%nldastime1 = 3000.0
nldasdrv%nldastime2 = 0.0

gridDesci(1) = 0
gridDesci(2) = nldasdrv%ncold
gridDesci(3) = nldasdrv%nrold
gridDesci(4) = 25.063
```

```
gridDesci(5) = -124.938
gridDesci(6) = 128
gridDesci(7) = 52.938
gridDesci(8) = -67.063
gridDesci(9) = 0.125
gridDesci(10) = 0.125
gridDesci(20) = 255
close(11)
```

1.54.2 retnldas.F90 (Source File: retnldas.F90)

Retrieves the name of a file from getncep.f and then gets the NCEP-LDAS forcing data using the zterp.f subroutine.
TIME1 = most recent past data TIME2 = most recent future data
The strategy for missing data is to backwards up to 10 days to get forcing at the same time of day.

REVISION HISTORY:

```
1  Oct 1999: Jared Entin; Initial code
15 Oct 1999: Paul Houser; Significant F90 Revision
11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With inland
             water filled in).  Deleteted unused variables.
27 Apr 2000: Brian Cosgrove; changed code to use the original
             mask again since that is the
             mask which  NCEP has already applied to the forcing data
             by the time NASA gets it......not possible to use the
             expanded NASA forcing mask
1  May 2000: Brian Cosgrove; changed code so that if parameter 11 (sw)
             is not found in hourly ncep data, it will just use
             edas-based shortwave from the hourly ncep files
20 Jun 2000: Brian Cosgrove; changed code so that it uses  LDAS%UDEF and
              not a hard-wired undefined value of -999.9 and -999.0
18 Aug 2000: Brian Cosgrove; changed code so that FMASK and not MASK
             is used when ungribbing.  NCEP data already has a mask applied
             to it and so may not be able to supply forcing data to
             all LDAS land forcing points.  In areas where LDAS
             forcing mask states that land exists, but where NCEP forcing
             data is non-existant, assign undefined value to forcing data.
22 Aug 2000: Brian Cosgrove; Altered code for US/Mexico/Canada Mask
05 Sep 2001: Brian Cosgrove; Removed dirnom and infile variables, changed
             call to ungribncep to match removal.  Added code to make use
             of precip weighting mask
02 Feb 2004: Sujay Kumar; Initial Specification in LIS
```

INTERFACE:

```
subroutine retnldas(order,name,ferror,ftype,dataflag)
```

USES:

```
use lisdrv_module, only : lis, grid, gindex
use nldasdomain_module, only : nldasdrv
use baseforcing_module, only : glbdata1, glbdata2
```

1.54.3 interp_nldas (Source File: retnldas.F90)

This subroutine interpolates a given NLDAS field to the LIS domain. Special treatment for some initialization fields. Code based on old ungribgdas.f

INTERFACE:

```
subroutine interp_nldas(kpds, nldas,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
use lisdrv_module, only : lis
use bilinear_interpMod, only : w110,w120,w210,w220,n110,n120,n210,n220,&
     rlat0,rlon0
use conserv_interpMod, only : w113,w123,w213,w223,n113,n123,n213,n223,&
     rlat3,rlon3
use nldasdomain_module, only : mi
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Setting interpolation options (ip=0,bilinear)
! (km=1, one parameter, ibi=1,use undefined bitmap
! (needed for soil moisture and temperature only)
! Use budget bilinear (ip=3) for precip forcing fields
!-----------------------------------------------------------------------
  mo = nc*nr
  if (kpds(5)==59 .or. kpds(5)==214) then
     ip=3
     ipopt(1)=-1
     ipopt(2)=-1
     km=1
     ibi=1
  else
     ip=0
     do i=1,20
        ipopt(i)=0
     enddo
     km=1
     ibi=1
```

```
  endif
!-----------------------------------------------------------------------
! Initialize output bitmap. Important for soil moisture and temp.
!-----------------------------------------------------------------------
  lo = .true.

!-----------------------------------------------------------------------
! Interpolate to LIS grid
!-----------------------------------------------------------------------
  if(lis%f%interp.eq.1) then
     call bilinear_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
          rlat0, rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
  elseif(lis%f%interp.eq.2) then
     if (kpds(5)==59 .or. kpds(5)==214) then
        call conserv_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
             rlat3,rlon3,w113,w123,w213,w223,n113,n123,n213,n223,iret)
     else
        call bilinear_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
             rlat0, rlon0,w110,w120,w210,w220,n110,n120,n210,n220,iret)
     endif
  endif
!-----------------------------------------------------------------------
! Create 2D array for main program. Also define a "soil" mask
! due to different geography between LDAS & LDAS. For LDAS land
! points not included in LDAS geography dataset only.
!-----------------------------------------------------------------------
  count = 0
  do j = 1, nr
     do i = 1, nc
        varfield(i,j) = lis1d(i+count)
     enddo
     count = count + nc
  enddo
!-----------------------------------------------------------------------
! Save air tempertaure interpolated field for later use in
! initialization of soil temp where geography differs
! between LDAS and LDAS
!-----------------------------------------------------------------------
!  if (kpds(5) .eq. 11 .and. kpds(6) .eq. 105) then
!     do i = 1, nc
!        do j = 1, nr
!           geogtemp(i,j) = varfield(i,j)
!        enddo
!     enddo
!  endif
```

1.54.4 time_interp_geos.F90 (Source File: time_interp_geos.F90)

Opens, reads, and interpolates GEOS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.55 Core Functions of time_interp_geos

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
  1  Oct 1999: Jared Entin; Initial code
  25 Oct 1999: Jared Entin; Significant F90 Revision
  11 Apr 2000: Brian Cosgrove; Fixed name construction error
               in Subroutine ETA6HRFILE
  27 Apr 2000: Brian Cosgrove; Added correction for use of old shortwave
               data with opposite sign convention from recent shortwave data.
               Added capability to use time averaged shortwave & longwave data
               Altered times which are passed into ZTERP--used to be GMT1
               and GMT2, now they are LDAS%ETATIME1 and LDAS%ETATIME2
  30 Nov 2000: Jon Radakovich; Initial code based on geteta.f
  17 Apr 2001: Jon Gottschalck; A few changes to allow model init.
  13 Aug 2001: Urszula Jambor; Introduced missing data replacement.
   5 Nov 2001: Urszula Jambor; Reset tiny negative SW values to zero.
```

INTERFACE:

```
 subroutine time_interp_nldas()

    btime=nldasdrv%nceptime1
    call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
   gmt1=GMT1+.25
   Make new decimal time that reflects added 15 minutes

         This was commented out so that it no longer shifts by 15 mins
         since this shift is no longer necessitated by forcing files
         Forcing files already contain this shift
  SUBROUTINE TICK(TIME,DOY,GMT,YR,MO,DA,HR,MN,SS,TS)

    tempbdoy=bdoy
   tempgmt1=gmt1-.25
    tempgmt1=gmt1
    tempbyr=byr
    tempbmo=bmo
    tempbda=bda
```

```
   tempbhr=bhr
   if (tempbhr.eq.24) tempbhr=0
   tempbmn=bmn
   tempbss=0
   tempbts=900
   call tick(newtime1,tempbdoy,tempgmt1,&
        tempbyr,tempbmo,tempbda,tempbhr,tempbmn, &
        tempbss,tempbts)

   btime=nldasdrv%nceptime2
   call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
  GMT2=GMT2+.25
        Make new decimal time that reflects added 15 minutes
   tempbdoy=bdoy
         TEMPGMT2=GMT2-.25
   tempgmt2=gmt2
   tempbyr=byr
   tempbmo=bmo
   tempbda=bda
   tempbhr=bhr
   if (tempbhr.eq.24) tempbhr=0
   tempbmn=bmn
   tempbss=0
   tempbts=900
   call tick(newtime2,tempbdoy,tempgmt2,&
     tempbyr,tempbmo,tempbda,tempbhr,tempbmn,&
     tempbss,tempbts)

 === Interpolate Data in time
   if(ldas%skipintp1.ne.1.or.ldas%skipintp2.ne.1)then !Skip interp if getETA routine on, and r
      wt1=(nldasdrv%nceptime2-lis%t%time)/(nldasdrv%nceptime2-nldasdrv%nceptime1)
      wt2=1.0-wt1
      swt1=(newtime2-lis%t%time)/(newtime2-newtime1)
      swt2=1.0-swt1

      do f=1,lis%d%nf
         if(f.eq.3)then
            do c=1,lis%d%nc
               do r=1,lis%d%nr
                  zdoy=ldas%doy

  compute and apply zenith angle weights

                  call zterp(1,grid(c,r)%lat,grid(c,r)%lon,&
                       gmt1,gmt2,ldas%gmt,zdoy,zw1,zw2,czb,cze,czm,ldas,grid)
                  grid(c,r)%forcing(f)=grid(c,r)%ncepdata1(f)*zw1+&
                      grid(c,r)%ncepdata2(f)*zw2
```

```
 In cases of small cos(zenith) angles, use linear weighting
       to avoid overly large weights

          if((grid(c,r)%forcing(f).gt.grid(c,r)%ncepdata1(f).and. &
             grid(c,r)%forcing(f).gt.grid(c,r)%ncepdata2(f)).and. &
             (czb.lt.0.1.or.cze.lt.0.1))then
           grid(c,r)%forcing(f)=grid(c,r)%ncepdata1(f)*swt1+ &
                grid(c,r)%ncepdata2(f)*swt2
 print *,'using swt1 and swt2',swt1,swt2
          endif

          if (grid(c,r)%forcing(f).gt.1367) then
             print *,'warning, sw radiation too high!!'
             print *,'it is',grid(c,r)%forcing(f)
             print *,'ncepdata1=',grid(c,r)%ncepdata1(f)
             print *,'ncepdata2=',grid(c,r)%ncepdata2(f)
             print *,'zw1=',zw1,'zw2=',zw2
             print *,'swt1=',swt1,'swt2=',swt2
             grid(c,r)%forcing(f)=grid(c,r)%ncepdata1(f)*swt1+ &
                  grid(c,r)%ncepdata2(f)*swt2
          endif

          if ( (grid(c,r)%ncepdata1(f).eq.lis%d%udef).and.
    &  (grid(c,r)%ncepdata2(f).eq.lis%d%udef) ) then
grid(c,r)%forcing(f)=lis%d%udef
endif
         enddo
        enddo
       else if(f.eq.8.or.f.eq.9)then !do block precipitation interpolation
        do c=1,lis%d%nc
         do r=1,lis%d%nr
          grid(c,r)%forcing(f)=grid(c,r)%ncepdata2(f)
         enddo
        enddo
       else !linearly interpolate everything else
        do c=1,lis%d%nc
         do r=1,lis%d%nr
          grid(c,r)%forcing(f)=grid(c,r)%ncepdata1(f)*wt1+ &
                               grid(c,r)%ncepdata2(f)*wt2
         enddo
        enddo
       endif
      enddo

=== ADJUST PRECIP TO VALUE PER SEC DATA SHOULD COME IN AS VALUE PER
=== TIME PERIOD ONE HOUR FOR NCEP, THREE HOUR FOR ETA

      do c=1,lis%d%nc
```

```
      do r=1,lis%d%nr
       grid(c,r)%forcing(8)=grid(c,r)%forcing(8)/(60.0*60.0)
       grid(c,r)%forcing(9)=grid(c,r)%forcing(9)/(60.0*60.0)
      enddo
     enddo

    endif !ldas%skipintp
```

1.55.1 time_interp_nldas.F90 (Source File: time_interp_nldas.F90)

Opens, reads, and interpolates NLDAS forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.56 Core Functions of time_interp_nldas

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
02Feb2004: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine time_interp_nldas()
```

USES:

```
use lisdrv_module, only : lis, grid
use baseforcing_module, only : glbdata1, glbdata2
use nldasdomain_module, only : nldasdrv
use grid_spmdMod
use time_manager, only : tick, time2date
implicit none
```

1.56.1 getgswp.F90 (Source File: getgswp.F90)

Opens, reads, and interpolates GSWP forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.57 Core Functions of getgswp

tick Determines GSWP data times

gswpfile Puts together appropriate file name for 3 hour intervals

readgswp Interpolates GSWP data to LDAS grid

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine getgswp()
```

USES:

```
use lisdrv_module, only : lis
use time_manager
use spmdMod
use tile_spmdMod
use baseforcing_module, only: glbdata1,glbdata2
use gswpdomain_module, only : gswpdrv
```

CONTENTS:

```
  if ( masterproc ) then
     nstep = get_nstep(lis%t)
  endif

!-----------------------------------------------------------------------
! Determine the correct number of forcing variables
!-----------------------------------------------------------------------
  if ( nstep .eq. 0) then
     nforce = gswpdrv%nmif
  else
     nforce = lis%f%nf
  endif
  lis%f%findtime1=0
  lis%f%findtime2=0
  movetime=0
!-----------------------------------------------------------------------
! Determine Required GSWP Data Times
! (The previous hour & the future hour)
!-----------------------------------------------------------------------
  yr1=lis%t%yr    !Time now
  mo1=lis%t%mo
  da1=lis%t%da
  hr1=lis%t%hr
  mn1=lis%t%mn
```

```
ss1=0
ts1=0

call tick(timenow,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

yr1=lis%t%yr    !Previous Hour
mo1=lis%t%mo
da1=lis%t%da
hr1=3*((lis%t%hr)/3)
mn1=0
ss1=0
ts1=0
call tick(time1,doy1,gmt1,yr1,mo1,da1,hr1,mn1,ss1,ts1)

yr2=lis%t%yr    !Next Hour
mo2=lis%t%mo
da2=lis%t%da
hr2=3*((lis%t%hr)/3)
mn2=0
ss2=0
ts2=3*60*60

call tick(time2,doy2,gmt2,yr2,mo2,da2,hr2,mn2,ss2,ts2)
if(timenow.gt.gswpdrv%gswptime2) then
   movetime=1
   lis%f%findtime2=1
endif

if ( nstep.eq.0 .or. nstep.eq.1 .or.lis%f%rstflag.eq.1 ) then
   lis%f%findtime1=1
   lis%f%findtime2=1
   glbdata1 = 0
   glbdata2 = 0
   movetime=0
   lis%f%rstflag = 0
endif
if(lis%f%findtime1 ==1) then
   print*, 'reading time1 data..'
   order = 1
   call readgswp(order, yr1,mo1,da1,hr1)
   gswpdrv%gswptime1 = time1
endif
if(movetime.eq.1) then
   gswpdrv%gswptime1=gswpdrv%gswptime2
   lis%f%findtime2=1
   do f=1,nforce
      do c=1,lis%d%ngrid
         glbdata1(f,c)=glbdata2(f,c)
```

```
        enddo
     enddo
  endif
  if(lis%f%findtime2.eq.1) then
     print*, 'reading time2 data..'
     order=2
     call readgswp(order, yr2,mo2,da2,hr2)
     gswpdrv%gswptime2 = time2
  endif

end subroutine getgswp


\markboth{Left}{Source File: gswpdomain\_module.F90,  Date: Fri Feb 11 15:41:04 EST 2005
}

!-----------------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED !WARRANTY.NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------------
```

1.58 Fortran: Module Interface gswpdomain_module.F90 (Source File: gswpdomain_module.F90)

Contains routines and variables that define the native domain for GSWP model forcing.

INTERFACE:

```
module gswpdomain_module
```

USES:

```
use gswpdrv_module
```

ARGUMENTS:

```
type(gswpdrvdec) :: gswpdrv
integer :: mi
```

1.58.1 defnatgswp.F90 (Source File: gswpdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for GSWP data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatgswp(gridDesci)
```

USES:

```
  use lisdrv_module, only: lis
  use time_manager, only : date2time
  implicit none
```

CONTENTS:

```
!    call readgswpcrd(gswpdrv,gridDesci)
    mi = gswpdrv%ncold*gswpdrv%nrold
    gswpdrv%gswptime1 = 3000.0
    gswpdrv%gswptime2 = 0.0
```

1.59 Fortran: Module Interface gswpdrv_module.F90 (Source File: gswpdrv_module.F90)

Module containing runtime specific GSWP variables

REVISION HISTORY:

```
20Feb2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
module gswpdrv_module
```

ARGUMENTS:

```
type gswpdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   integer :: nmif
   real*8 :: gswptime1,gswptime2
   character*40 :: gswpdir   !GEOS Forcing Directory
end type gswpdrvdec
```

1.59.1 readgswp.F90 (Source File: readgswp.F90)

Reads GSWP forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

REVISION HISTORY:

```
20Feb2004; Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine readgswp(order, yr,mo,da,hr)
```

USES:

```
use lisdrv_module, only : lis,gindex
use baseforcing_module, only: glbdata1,glbdata2
use gswp_module, only : getgswp_timeindex
```

1.59.2 time_interp_gswp.F90 (Source File: time_interp_gswp.F90)

Opens, reads, and interpolates GSWP forcing.
TIME1 = most recent past data
TIME2 = nearest future data

The strategy for missing data is to go backwards up to 10 days to get forcing at the same time of day.

1.60 Core Functions of time_interp_gswp

zterp Performs zenith angle-based temporal interpolation

REVISION HISTORY:

```
20Feb2004; Sujay Kumar : Initial Specification
```

INTERFACE:

```
subroutine time_interp_gswp()
```

USES:

```
   use lisdrv_module, only : lis, grid
   use baseforcing_module, only : glbdata1, glbdata2
   use time_manager
   use grid_spmdMod
   use spmdMod
   use gswpdomain_module, only :gswpdrv
```

CONTENTS:

```
  if(masterproc) then
     if (get_nstep(lis%t) .eq. 0) then
        lis%f%nforce = lis%f%nmif
     else
        lis%f%nforce = lis%f%nf
     endif
  endif

  btime=gswpdrv%gswptime1
  call time2date(btime,bdoy,gmt1,byr,bmo,bda,bhr,bmn)
  btime=gswpdrv%gswptime2
  call time2date(btime,bdoy,gmt2,byr,bmo,bda,bhr,bmn)
!-----------------------------------------------------------------------
!  Interpolate Data in Time
!-----------------------------------------------------------------------
  wt1=(gswpdrv%gswptime2-lis%t%time)/ &
       (gswpdrv%gswptime2-gswpdrv%gswptime1)
  wt2=1.0-wt1

  do f=1,lis%f%nforce
     if(f.eq.3) then
        if (lis%f%shortflag.eq.2) then
!-----------------------------------------------------------------------
! Got Time Averaged SW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              zdoy=lis%t%doy
              call zterp(0,grid(c)%lat,grid(c)%lon, &
                   gmt1,gmt2,lis%t%gmt,zdoy, &
                   zw1,zw2,czb,cze,czm,lis)
              grid(c)%forcing(f)=glbdata2(f,c)*zw1

              if ((grid(c)%forcing(f).ne.lis%d%udef).and. &
                   (grid(c)%forcing(f).lt.0) ) then
                 if (grid(c)%forcing(f) > -0.00001) then
                    grid(c)%forcing(f) = 0.0
```

```
                 else
                    print*,'ERR: time_interp_gswp -- Stopping because ', &
                         'forcing not udef but lt0,'
                    print*,'ERR: time_interp_gswp -- ', &
                         'f,c,grid(c)%forcing(f),glbdata2(f,c)',  &
                         f,c,grid(c)%forcing(f),glbdata2(f,c), &
                         ' (',iam,')'
                    call endrun
                 end if
              endif

              if (grid(c)%forcing(f).gt.1367) then
                 grid(c)%forcing(f)=glbdata2(f,c)
              endif
           enddo
        endif

     else if(f.eq.8.or.f.eq.9) then
!-----------------------------------------------------------------------
! precip variable Block Interpolation
!-----------------------------------------------------------------------
        do c=1,gdi(iam)
           grid(c)%forcing(f)=glbdata2(f,c)
        enddo

     else if (f.eq.4) then
        if (lis%f%longflag.eq.1) then
!-----------------------------------------------------------------------
!    Got Instantaneous LW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
                   glbdata2(f,c)*wt2
           enddo
        endif

        if (lis%f%longflag.eq.2) then
!-----------------------------------------------------------------------
!    Got Time Averaged LW
!-----------------------------------------------------------------------
           do c=1,gdi(iam)
              grid(c)%forcing(f)=glbdata2(f,c)

           enddo
          endif

       else
!-----------------------------------------------------------------------
```

```
!     Linearly interpolate everything else
!-----------------------------------------------------------------------
          do c=1,gdi(iam)
             grid(c)%forcing(f)=glbdata1(f,c)*wt1+ &
                  glbdata2(f,c)*wt2
          enddo
       endif
    enddo
84  format('now',i4,4i3,2x,'pvt ',a22,' nxt ',a22)
    return
```

1.61 Fortran: Module Interface cmapdomain_module.F90 (Source File: cmapdomain_module.F90)

Contains routines and variables that define the native domain for CMAP precipitation product.

INTERFACE:

```
module cmapdomain_module
```

USES:

```
use cmapdrv_module
```

ARGUMENTS:

```
type(cmapdrvdec) :: cmapdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:,:)
integer, allocatable :: n12(:,:)
integer, allocatable :: n21(:,:)
integer, allocatable :: n22(:,:)
real, allocatable ::  w11(:,:),w12(:,:)
real, allocatable ::  w21(:,:),w22(:,:)
```

1.61.1 defnatcmap.F90 (Source File: cmapdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for CMAP data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatcmap()
```

USES:

```
use lisdrv_module, only: lis
use time_manager, only : date2time
implicit none
```

ARGUMENTS:

```
real :: gridDesci(50)
integer :: updoy, yr1,mo1,da1,hr1,mn1,ss1
real :: upgmt
```

CONTENTS:

```
call readcmapcrd(cmapdrv)
gridDesci = 0
gridDesci(1) = 4
gridDesci(2) = 512
gridDesci(3) = 256
gridDesci(4) = 89.463
gridDesci(5) = 0
gridDesci(6) = 128
gridDesci(7) = -89.463
gridDesci(8) = -0.703
gridDesci(9) = 0.703
gridDesci(10) = 128
gridDesci(20) = 255
call allocate_cmap_ip(lis%d%lnc*lis%d%lnr)
call conserv_cmap_interp_input(gridDesci,lis%d%gridDesc,&
     lis%d%lnc*lis%d%lnr)
yr1 = 2002     !grid update time
mo1 = 10
da1 = 29
hr1 = 12
mn1 = 0; ss1 = 0
call date2time(cmapdrv%griduptime1,updoy,upgmt,yr1,mo1,da1,hr1,mn1,ss1 )
cmapdrv%gridchange1 = .true.
```

1.61.2 allocate_cmap_ip (Source File: cmapdomain_module.F90)

Allocates memory for CMAP interpolation variables

INTERFACE:

```
subroutine allocate_cmap_ip(N)
```

CONTENTS:

```
allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n,25))
allocate(n12(n,25))
allocate(n21(n,25))
allocate(n22(n,25))
allocate(w11(n,25))
allocate(w12(n,25))
allocate(w21(n,25))
allocate(w22(n,25))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.61.3 def_cmap_ip_input (Source File: cmapdomain_module.F90)

Calculates weights and neighbor information required for CMAP interpolation

INTERFACE:

```
subroutine conserv_cmap_interp_input(gridDesci,gridDesco,npts)
  real, intent(in) :: gridDesci(50)
  real             :: gridDesco(50)
  real, parameter     :: fill = -9999.0
  real                :: xpts(npts), ypts(npts)
  real                :: xptb(npts), yptb(npts)
  real                :: rlob(npts), rlab(npts)
  integer             :: npts
  integer             :: ipopt(20)
  integer             :: nb1, nb2, mo
  integer             :: i1, i2, j1, j2
  real                :: xi, xf, yi, yf
  integer             :: get_fieldpos
  integer             :: iret,ib,nb,jb,n,nv,lb,wb

  !  COMPUTE NUMBER OF OUTPUT POINTS AND THEIR LATITUDES AND LONGITUDES.
  mo = npts
  iret=0
  ipopt = 0
  ipopt(1) = -1
  ipopt(2) = -1
  if(gridDesco(1).ge.0) then
```

```
      call compute_coord(gridDesco, 0,mo,fill,xpts,ypts,rlon,rlat,nv,0)
      if(mo.eq.0) iret=3
   else
      iret=31
   endif
   ! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   !  SET PARAMETERS
   nb1=ipopt(1)
   if(nb1.eq.-1) nb1=2
   if(iret.eq.0.and.nb1.lt.0.) iret=32
   if(iret.eq.0.and.nb1.ge.20.and.ipopt(2).ne.-1) iret=32
   if(iret.eq.0) then
      nb2=2*nb1+1
      nb3=nb2*nb2
      nb4=nb3
      if(ipopt(2).ne.-1) then
         nb4=ipopt(2)
         do ib=1,nb1
            nb4=nb4+8*ib*ipopt(2+ib)
         enddo
      endif
   else
      nb2=0
      nb3=0
      nb4=0
   endif

   ! - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
   !  LOOP OVER SAMPLE POINTS IN OUTPUT GRID BOX
   do nb=1,nb3
      !  locate input points and compute their weights
      jb=(nb-1)/nb2-nb1
      ib=nb-(jb+nb1)*nb2-nb1-1
      lb=max(abs(ib),abs(jb))
      wb=1
      if(ipopt(2).ne.-1) wb=ipopt(2+lb)
      if(wb.ne.0) then
         do n=1,mo
            xptb(n)=xpts(n)+ib/real(nb2)
            yptb(n)=ypts(n)+jb/real(nb2)
         enddo
         call compute_coord(gridDesco, 1,mo,fill,xptb,yptb,rlob,rlab,nv,0)
         call compute_coord(gridDesci,-1,mo,fill,xptb,yptb,rlob,rlab,nv,0)
         if(iret.eq.0.and.nv.eq.0.and.lb.eq.0) iret=2
         do n=1,mo
            xi=xptb(n)
            yi=yptb(n)
            if(xi.ne.fill.and.yi.ne.fill) then
```

```
                i1=xi
                i2=i1+1
                j1=yi
                j2=j1+1
                xf=xi-i1
                yf=yi-j1
                n11(n,nb)=get_fieldpos(i1,j1,gridDesci)
                n21(n,nb)=get_fieldpos(i2,j1,gridDesci)
                n12(n,nb)=get_fieldpos(i1,j2,gridDesci)
                n22(n,nb)=get_fieldpos(i2,j2,gridDesci)
                if(min(n11(n,nb),n21(n,nb),n12(n,nb),n22(n,nb)).gt.0) then
                   w11(n,nb)=(1-xf)*(1-yf)
                   w21(n,nb)=xf*(1-yf)
                   w12(n,nb)=(1-xf)*yf
                   w22(n,nb)=xf*yf
                else
                   n11(n,nb)=0
                   n21(n,nb)=0
                   n12(n,nb)=0
                   n22(n,nb)=0
                endif
             else
                n11(n,nb)=0
                n21(n,nb)=0
                n12(n,nb)=0
                n22(n,nb)=0
             endif
              print*, 'def ',n,nb,n113(n,nb),n213(n,nb)
          enddo
       endif
    enddo
  end subroutine conserv_cmap_interp_input
#if 0
  subroutine def_cmap_ip_input (gridDesci)
```

USES:

```
    use spmdMod
    use lisdrv_module, only:lis
```

1.62 Fortran: Module Interface cmapdrv_module.F90 (Source File: cmapdrv_module.F90)

Module for runtime specific CMAP variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module cmapdrv_module
```

ARGUMENTS:

```
type cmapdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   character*40 :: cmapdir   !CMAP Forcing Directory
   real*8 :: cmaptime
   real*8 :: griduptime1
   logical :: gridchange1
end type cmapdrvdec
```

1.62.1 getcmap.F90 (Source File: getcmap.F90)

Opens and reads global precipitation forcing
CTIME = Current time
FTIMENRL = Nearest future data for NRL data
FTIMEHUFF = Nearest future data for HUFFMAN data
FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
              using a ratio of the model convective / total ratio
30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources
```

INTERFACE:

```
subroutine getcmap()
```

USES:

```
use lisdrv_module, only : lis, gindex
use time_manager
use cmapdomain_module, only : cmapdrv, conserv_cmap_interp_input
implicit none
```

CONTENTS:

```
  endtime_cmap = 0
!-------------------------------------------------------------------------
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-------------------------------------------------------------------------
  gap = 0.0001712328767098370
```

```
!-----------------------------------------------------------------------
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-----------------------------------------------------------------------
  yr1 = lis%t%yr  !current time
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = lis%t%mn
  ss1 = 0
  ts1 = 0
  call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
!-----------------------------------------------------------------------
! CMAP product end time
!-----------------------------------------------------------------------
  yr5 = lis%t%yr  !end accumulation time data
  mo5 = lis%t%mo
  da5 = lis%t%da
  hr5 = 6*(lis%t%hr/6)
  mn5 = 0
  ss5 = 0
  ts5 = 6*60*60
  call tick( ftime_cmap, doy5, gmt5, yr5, mo5, da5, hr5, mn5, ss5, ts5 )
  if(ctime > cmapdrv%griduptime1 .and. &
     cmapdrv%gridchange1 ) then
     print*, 'changing cmap grid to 2002 - '
     cmapdrv%ncold = 768
     cmapdrv%nrold = 384
!-------------------------------------------------------------------
! Reinitialize the weights and neighbors
!-------------------------------------------------------------------
     gridDesci = 0
     gridDesci(1) = 4
     gridDesci(2) = 768
     gridDesci(3) = 384
     gridDesci(4) = 89.462
     gridDesci(5) = 0
     gridDesci(6) = 128
     gridDesci(7) = -89.462
     gridDesci(8) = -0.469
     gridDesci(9) = 0.469
     gridDesci(10) = 192
     gridDesci(20) = 255
     call conserv_cmap_interp_input(gridDesci,lis%d%gridDesc,&
          lis%d%lnc*lis%d%lnr)
     cmapdrv%gridchange1 = .false.
  endif
```

```
!-----------------------------------------------------------------------
! Ensure that data is found during first time step
!-----------------------------------------------------------------------
  if ( lis%f%gpcpsrc.eq.4.and. get_nstep(lis%t).eq. 1 &
  .or.lis%f%rstflag .eq. 1) then
     endtime_cmap = 1
     lis%f%rstflag = 0
  endif
!-----------------------------------------------------------------------
! Check for and get CMAP CPC Precipitation data
!-----------------------------------------------------------------------
  if (lis%f%gpcpsrc==4) then
     if ( ctime > cmapdrv%cmaptime ) endtime_cmap = 1
     if ( endtime_cmap == 1 ) then  !get new time2 data
        ferror_cmap = 0
        call cmapfile( name, cmapdrv%cmapdir, yr5, mo5, da5, hr5 )
        print*, 'Getting new CMAP CPC precip data',name
        call glbprecip_cmap( name, ferror_cmap, hr5 )
        cmapdrv%cmaptime = ftime_cmap
     endif  !need new time2
  endif
  return
```

1.62.2 cmapfile (Source File: getcmap.F90)

This subroutine puts together CMAP file name for 6 hour file intervals

INTERFACE:

```
 subroutine cmapfile( name, cmapdir, yr, mo, da, hr)
```

CONTENTS:

```
91 format (a4)
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a10)
98 format (a1, i4, i2, a1)
99 format (8a1)
!-----------------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-----------------------------------------------------------------------
  uyr = yr
```

```
umo = mo
uda = da
uhr = 6*(hr/6)  !hour needs to be a multiple of 6 hours
umn = 0
uss = 0
ts1 = -24*60*60 !one day interval to roll back date.

open(unit=90, file='temp', form='formatted', access='direct', recl=80)
write(90, 96, rec=1) cmapdir
read(90, 92, rec=1) (fbase(i), i=1,80)

write(90, 98, rec=1) '/', uyr, umo, '/'
read(90, 99, rec=1) fdir
do i = 1, 8
   if ( fdir(i) == ' ' ) fdir(i) = '0'
end do

write(90, 97, rec=1) 'cmap_gdas_'
read (90, 92, rec=1) (fsubs(i), i=1,10)

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
   if ( ftime(i) == ' ' ) ftime(i) = '0'
end do

write(90, 94, rec=1) uyr, umo, uda, uhr
read(90, 95, rec=1) ftime
do i = 1, 10
   if ( ftime(i) == ' ' ) ftime(i) = '0'
end do

write(90, 91, rec=1) '.grb'
read (90, 92, rec=1) (fsubs2(i), i=1,4)
c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(90, 92, rec=1) (fbase(i), i=1,c), (fdir(i), i=1,8),  &
                     (fsubs(i), i=1,10),(ftime(i), i=1,10), &
                     (fsubs2(i), i=1,4)

read(90, 93, rec=1) name

close(90)
return
```

1.62.3 glbprecip_cmap.F90 (Source File: glbprecip_cmap.F90)

Includes reading routines for global CMAP precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
             observations with domain 3 (2x2.5)
30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data
```

INTERFACE:

```
subroutine glbprecip_cmap( fname, ferror_cmap, filehr)
```

USES:

```
use lisdrv_module, only : lis, gindex
use obsprecipforcing_module, only: obsprecip
use cmapdomain_module, only : cmapdrv
implicit none
```

ARGUMENTS:

```
character(len=80)  :: fname           ! Filename variable for datafile
integer            :: ferror_cmap
integer            :: filehr
```

CONTENTS:

```
  allocate (precip_regrid(lis%d%lnc,lis%d%lnr))
  obsprecip     = -1.0
  precip_regrid = -1.0
!-----------------------------------------------------------------------
! Set necessary parameters for call to interp_gdas
!-----------------------------------------------------------------------
  ism     = 0
  udef    = lis%d%udef
  jj      = 0
  if (mod((filehr),12).eq.0) then
     lugb=134
  else
     lugb=138
  endif
  ncmap = cmapdrv%ncold*cmapdrv%nrold
  allocate(cmapin(ncmap))
  allocate(lb(ncmap))
  lugi    = 0
  jpds    = -1
```

```
  jpds(5) = 59
  jpds(6) = 1
  jpds(7) = 0
  jgds    = 0
  cmapin = 0.0
  call baopen (lugb,fname,iret)
  if (iret == 0 ) then
     call getgb (lugb,lugi,ncmap,jj,jpds,jgds,kf,k,kpds,&
          gridDesccmap,lb,cmapin,iret)
!     do j=1,ncmap
!        if(cmapin(j).ne.0) print*, cmapin
!     enddo
     print*, iret
     call interp_cmap(kpds,ncmap,cmapin,lb,lis%d%gridDesc, &
          lis%d%lnc,lis%d%lnr,precip_regrid)
     do j = 1,lis%d%lnr
        do i = 1,lis%d%lnc
!           if(precip_regrid(i,j) .eq. -1) then
!              print*, j,i,precip_regrid(i,j)
!           endif
           if (precip_regrid(i,j) .ne. -1.0) then
              index = gindex(i,j)
              if(index .ne. -1) then
                 obsprecip(index) = precip_regrid(i,j)*3600.0
              endif
           endif
        enddo
     enddo

     call baclose (lugb,jret)

     ferror_cmap = 1
     close(10)
     print*, "Obtained CMAP CPC precipitation data ", fname
  else
     print*, "Missing CMAP CPC precipitation data ", fname
     ferror_cmap = 0
  endif
  deallocate (precip_regrid)
  deallocate(lb)
  deallocate(cmapin)
```

1.62.4 interp_cmap.F90 (Source File: interp_cmap.F90)

Interpolates CMAP observed precipitation forcing

INTERFACE:

```
subroutine interp_cmap(kpds,ngdas,f,lb,lis_gds,nc,nr, &
     varfield)
```

USES:

```
  use cmapdomain_module, only : mi,w11,w12,w21,w22,&
       n11,n12,n21,n22,rlat,rlon

  implicit none
```

ARGUMENTS:

```
  integer :: nc, nr, ngdas
  integer :: kpds(200)
  real :: lis_gds(50)
  real :: f(ngdas)
  logical*1 :: lb(ngdas)
  real, dimension(nc,nr) :: varfield
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Setting interpolation options (ip=0,bilinear)
! (km=1, one parameter, ibi=1,use undefined bitmap
! (needed for soil moisture and temperature only)
! Use budget bilinear (ip=3) for precip forcing fields
!-----------------------------------------------------------------------
  mo = nc*nr
  if (kpds(5)==59 .or. kpds(5)==214) then
     ip=3
     ipopt(1)=-1
     ipopt(2)=-1
     km=1
     ibi=1
  else
     ip=0
     do i=1,20
        ipopt(i)=0
     enddo
     km=1
     ibi=1
  endif
!-----------------------------------------------------------------------
! Initialize output bitmap. Important for soil moisture and temp.
!-----------------------------------------------------------------------
  lo = .true.


!  call ipolates (ip,ipopt,gridDesc,lis_gds,ngdas,nglis, &
```

```
!       km,ibi,lb,f,no,rlat,rlon,ibo,lo,lis1d,iret)
  mi = ngdas
!  call polates0 (lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,&
!       rlat, rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)
  call conserv_interp(lis_gds,ibi,lb,f,ibo,lo,lis1d,mi,mo,&
       rlat, rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)
!-----------------------------------------------------------------------
! Create 2D array for main program. Also define a "soil" mask
! due to different geography between GDAS & LDAS. For LDAS land
! points not included in GDAS geography dataset only.
!-----------------------------------------------------------------------
  count = 0
  do j = 1, nr
     do i = 1, nc
        varfield(i,j) = lis1d(i+count)
        geogmask(i,j) = lo(i+count)
     enddo
     count = count + nc
  enddo
!-----------------------------------------------------------------------
! Save air tempertaure interpolated field for later use in
! initialization of soil temp where geography differs
! between GDAS and LDAS
!-----------------------------------------------------------------------
  if (kpds(5) .eq. 11 .and. kpds(6) .eq. 105) then
     do i = 1, nc
        do j = 1, nr
           geogtemp(i,j) = varfield(i,j)
        enddo
     enddo
  endif
```

1.62.5 readcmapcrd.F90 (Source File: readcmapcrd.F90)

Routine to read CMAP specific parameters from the card file.

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readcmapcrd(cmapdrv)
```

USES:

```
use cmapdrv_module
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=cmap)
  print*,'Using CMAP forcing'
  print*, 'CMAP forcing directory :',cmapdrv%CMAPDIR
!-----------------------------------------------------------------------
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----------------------------------------------------------------------
  cmapdrv%cmaptime = 0.0
  close(11)
```

1.62.6 time_interp_cmap.F90 (Source File: time_interp_cmap.F90)

Calls the post processing utilities for handling observed precipitation data for CMAP

INTERFACE:

```
   subroutine time_interp_cmap
```

USES:

```
     use lisdrv_module, only:lis, grid
     use obsprecipforcing_module, only : obsprecip
     use grid_spmdMod
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Compute ratio between convective model precip and total model precip
! so that it can be applied to the observed global precip
!-----------------------------------------------------------------------
    do c = 1,gdi(iam)
       if (grid(c)%forcing(8) .ne. 0.0 .and.  &
            grid(c)%forcing(8) .ne. lis%d%udef .and.  &
            grid(c)%forcing(9) .ne. lis%d%udef) then
          ratio(c) = grid(c)%forcing(9) / grid(c)%forcing(8)
          if (ratio(c) .gt. 1.0) ratio(c) = 1.0
          if (ratio(c) .lt. 0.0) ratio(c) = 0.0
       else
          ratio(c) = 0.0
       endif
    enddo
    do c = 1, gdi(iam)
       if (obsprecip(c) .ne. -1.0) then
  grid(c)%forcing(8) = obsprecip(c) / 3600.0
  grid(c)%forcing(9) = ratio(c) * grid(c)%forcing(8)
```

```
      endif
   enddo
```

1.62.7 gethuff.F90 (Source File: gethuff.F90)

Opens and reads global precipitation forcing
CTIME = Current time
FTIMENRL = Nearest future data for NRL data
FTIMEHUFF = Nearest future data for HUFFMAN data
FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
              using a ratio of the model convective / total ratio
30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources
```

INTERFACE:

```
subroutine gethuff
```

USES:

```
use lisdrv_module, only : lis, gindex
use time_manager
use huffdomain_module, only : huffdrv
implicit none
```

CONTENTS:

```
!-------------------------------------------------------------------------
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-------------------------------------------------------------------------
  gap = 0.0001712328767098370
!-------------------------------------------------------------------------
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-------------------------------------------------------------------------
  yr1 = lis%t%yr  !current time
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = lis%t%mn
  ss1 = 0
  ts1 = 0
```

```
  call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

!-----------------------------------------------------------------------
! HUFFMAN product end time
!-----------------------------------------------------------------------
  yr3 = lis%t%yr  !end accumulation time data
  mo3 = lis%t%mo
  da3 = lis%t%da
  hr3 = 3*(lis%t%hr/3)
  mn3 = 0
  ss3 = 0
  ts3 = 3*60*60
  call tick( ftime_huff, doy3, gmt3, yr3, mo3, da3, hr3, mn3, ss3, ts3 )
  breaktime = ftime_huff - ctime
  datatime  = ftime_huff
  fnametime = ftime_huff
  if (lis%f%gpcpsrc == 3) then
     if (breaktime .ge. gap) then
        call time2date( datatime, kdoy3, kgmt3, kyr3, &
             kmo3, kda3, khr3, kmn3 )
        call time2date( fnametime, mdoy3, mgmt3, myr3, &
             mmo3, mda3, mhr3, mmn3 )
        flag1 = 1
        if (khr3 == 24) khr3 = 0
        if (mhr3 == 24) mhr3 = 0
        if (kgmt3 .eq. 0.0 .and. flag2 .eq. 2) then
           kts3 = -25.5*60*60
           call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                kda3, khr3, kmn3, kss3, kts3 )
           mts3 = -27*60*60
           call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                mda3, mhr3, mmn3, mss3, mts3 )
        else
           kts3 = -1.5*60*60
           call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                kda3, khr3, kmn3, kss3, kts3 )
           mts3 = -3*60*60
           call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                mda3, mhr3, mmn3, mss3, mts3 )
        endif
        flag2 = 1
     else
        if (get_nstep(lis%t).eq. 1) then
           call time2date( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                kda3, khr3, kmn3 )
           call time2date( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                mda3, mhr3, mmn3 )
           if (kgmt3 .eq. 0) then
```

```
            mts3 = -24*60*60
            call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = -22.5*60*60
            call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         else
            mts3 = 0
            call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = 1.5*60*60
            call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         endif
      else
         flag1 = 2
         if (flag2 .eq. 1) then
            mts3 = 3*60*60
            call tick( fnametime, mdoy3, mgmt3, myr3, mmo3, &
                 mda3, mhr3, mmn3, mss3, mts3 )
            kts3 = 3*60*60
            call tick( datatime, kdoy3, kgmt3, kyr3, kmo3, &
                 kda3, khr3, kmn3, kss3, kts3 )
         endif
         flag2 = 2
      endif
   endif
endif
!-----------------------------------------------------------------------
! Ensure that data is found during first time step
!-----------------------------------------------------------------------
if ( lis%f%gpcpsrc.eq.3 .and. get_nstep(lis%t).eq. 1 ) endtime_huff = 1

!-----------------------------------------------------------------------
! Check for and get HUFFMAN observed Precipitation data
!-----------------------------------------------------------------------
if (lis%f%gpcpsrc.eq.3) then
   if ( ctime > huffdrv%hufftime ) then
      endtime_huff = 1
      if ( endtime_huff == 1 ) then  !get new time2 data
         print*, 'Getting new HUFFMAN satellite precip data', endtime_huff
         ferror_huff = 0
         call hufffile( name, huffdrv%huffdir, myr3, mmo3, mda3, mhr3 )
         call glbprecip_huff( name, ferror_huff )
         huffdrv%hufftime = datatime
      endif
   endif
endif
```

```
return
```

1.62.8 hufffile (Source File: gethuff.F90)

This subroutine puts together HUFFMAN file name for 3 hour file intervals

INTERFACE:

```
 subroutine hufffile( name, huffdir, yr, mo, da, hr)
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a4)
98 format (a1, i4, i2, a1)
99 format (8a1)
89 format (a7)
!-------------------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed out
!-------------------------------------------------------------------------

  uyr = yr
  umo = mo
  uda = da
  uhr = 3*(hr/3)  !hour needs to be a multiple of 3 hours
  umn = 0
  uss = 0
  ts1 = -24*60*60 !one day interval to roll back date.

  open(unit=90, file='temp', form='formatted',access='direct', recl=80)
  write(90, 96, rec=1) huffdir
  read(90, 92, rec=1) (fbase(i), i=1,80)

  write(90, 98, rec=1) '/', uyr, umo, '/'
  read(90, 99, rec=1) fdir
  do i = 1, 8
   if ( fdir(i) == ' ' ) fdir(i) = '0'
  end do

  write(90, 94, rec=1) uyr, umo, uda, uhr
  read(90, 95, rec=1) ftime
  do i = 1, 10
```

```
    if ( ftime(i) == ' ' ) ftime(i) = '0'
  end do

  write(90, 97, rec=1) '.bin'
  read (90, 92, rec=1) (fsubs(i), i=1,4)


  write(90, 89, rec=1) '3B42RT.'
  read (90, 92, rec=1) (fprefix(i), i=1,7)
  c = 0
  do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  end do

  write(90, 92, rec=1) (fbase(i),i=1,c),(fdir(i),i=1,8),(fprefix(i), i=1,7),  &
    (ftime(i), i=1,10), (fsubs(i), i=1,4)

  read(90, 93, rec=1) name

  close(90)
  return
```

1.62.9 glbprecip_huff.F90 (Source File: glbprecip_huff.F90)

Includes reading routines for global HUFFMAN precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

```
    17 Jul 2001: Jon Gottschalck; Initial code
    04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
                 observations with domain 3 (2x2.5)
    30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data
```

INTERFACE:

```
 subroutine glbprecip_huff (name_huff, ferror_huff )
```

USES:

```
  use lisdrv_module, only : gindex
  use obsprecipforcing_module, only: obsprecip
```

CONTENTS:

```
 fname = name_huff
!-----------------------------------------------------------------------
! Fill necessary arrays to assure not using old HUFFMAN data
```

```
!-----------------------------------------------------------------------
 precip = -1.0
 obsprecip = -1.0
!-----------------------------------------------------------------------
! Find HUFFMAN precip data, read it in and assign to forcing precip array.
! Must reverse grid in latitude dimension to be consistent with LDAS grid
!-----------------------------------------------------------------------
  open(unit=10,file=fname, status='old', &
   &          access='direct',recl=xd*yd*4, &
   &          form='unformatted',iostat=ios)

  if (ios .eq. 0) then
     read (10,rec=1) head (1:2880),&
          ( ( rr (i, j), i = 1, xd ), j = 1, yd)
   do i = 1,yd
    do j = 1,xd
     if (j .lt. 721) then
      precip(j,i) = float(rr(j+720,yd+1-i)) / 100.0
      if ( rr(j+720,yd+1-i) .eq. ibad ) precip(j,i) = -1.0
      if ( rr(j+720,yd+1-i) .lt. 0.0 )  precip(j,i) = -1.0
     else
      precip(j,i) = float(rr(j-720,yd+1-i)) / 100.0
      if ( rr(j-720,yd+1-i) .eq. ibad ) precip(j,i) = -1.0
      if ( rr(j-720,yd+1-i) .lt. 0.0 )  precip(j,i) = -1.0
     endif
    enddo
  enddo
!-----------------------------------------------------------------------
! Interpolating to desired domain and resolution
! Global precip datasets not used currently to force NLDAS
!-----------------------------------------------------------------------
  do i = 1,yd
   do j = 1,xd
       index = gindex(j,i)
       if (index .ne. -1) then
         obsprecip(index) = precip(j,i)
       endif
   enddo
  enddo

  ferror_huff = 1
 close(10)
  print*, "Obtained HUFFMAN precipitation data ", fname
 else
   print*, "Missing HUFFMAN precipitation data ", fname
   ferror_huff = 0
 endif
```

1.63 Fortran: Module Interface huffdomain_module.F90 (Source File: huffdomain_module.F90)

Contains routines and variables that define the native domain for HUFF precipitation product.

INTERFACE:

```
module huffdomain_module
```

USES:

```
use huffdrv_module
```

ARGUMENTS:

```
type(huffdrvdec) :: huffdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:)
integer, allocatable :: n12(:)
integer, allocatable :: n21(:)
integer, allocatable :: n22(:)
real, allocatable ::  w11(:),w12(:)
real, allocatable ::  w21(:),w22(:)
```

1.63.1 defnathuff.F90 (Source File: huffdomain_module.F90)

Defines the kgds array describing the native forcing resolution for HUFF data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnathuff()
```

USES:

```
  use lisdrv_module, only: lis
  implicit none
```

ARGUMENTS:

```
  integer :: kgdsi(200)
```

CONTENTS:

```
call readhuffcrd(huffdrv)
kgdsi = 0
call allocate_huff_ip(lis%d%lnc*lis%d%lnr)
```

1.63.2 allocate_huff_ip (Source File: huffdomain_module.F90)

Allocates memory for HUFF interpolation variables

INTERFACE:

```
subroutine allocate_huff_ip(N)
```

CONTENTS:

```
allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n))
allocate(n12(n))
allocate(n21(n))
allocate(n22(n))
allocate(w11(n))
allocate(w12(n))
allocate(w21(n))
allocate(w22(n))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.64 Fortran: Module Interface huffdrv_module.F90 (Source File: huffdrv_module.F90)

Module for runtime specific HUFF variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module huffdrv_module
```

ARGUMENTS:

```
type huffdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   character*40 :: huffdir   !HUFF Forcing Directory
   real*8 :: hufftime
end type huffdrvdec
```

1.64.1 readhuffcrd.F90 (Source File: readhuffcard.F90)

Routine to read HUFF specific parameters from the card file.

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readhuffcrd(huffdrv)
```

USES:

```
use huffdrv_module
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=huff)
  print*,'Using HUFF forcing'
  print*, 'HUFF forcing directory :',huffdrv%HUFFDIR
!-----------------------------------------------------------------------
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----------------------------------------------------------------------
  huffdrv%hufftime = 0.0
  close(11)
```

1.64.2 time_interp_huff.F90 (Source File: time_interp_huff.F90)

Calls the post processing utilities for handling observed precipitation data for HUFF

INTERFACE:

```
subroutine time_interp_huff
```

CONTENTS:

```
! To be implemented
```

1.64.3 getpers.F90 (Source File: getpers.F90)

Opens and reads global precipitation forcing
CTIME = Current time
FTIMENRL = Nearest future data for NRL data
FTIMEHUFF = Nearest future data for HUFFMAN data
FTIMEPERS = Nearest future data for PERSIANN data

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
10 Oct 2001: Jon Gottschalck; Modified to adjust convective precip
              using a ratio of the model convective / total ratio
30 Jul 2002: Jon Gottschalck; Added PERSIANN and HUFFMAN global observed precip data sources
```

INTERFACE:

```
subroutine getpers
```

USES:

```
use lisdrv_module, only : lis, gindex
use time_manager
use persdomain_module, only : persdrv
implicit none
```

CONTENTS:

```
!-------------------------------------------------------------------------
! Set parameter to measure 1.5 hour time offset when using HUFFMAN
!-------------------------------------------------------------------------
  gap = 0.0001712328767098370
!-------------------------------------------------------------------------
! Determine required observed precip data times
! (current, accumulation end time)
! Model current time
!-------------------------------------------------------------------------
  yr1 = lis%t%yr  !current time
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = lis%t%mn
  ss1 = 0
  ts1 = 0
  call tick( ctime, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

!-------------------------------------------------------------------------
! PERSIANN product end time
!-------------------------------------------------------------------------
  yr4 = lis%t%yr  !end accumulation time data
```

```
  mo4 = lis%t%mo
  da4 = lis%t%da
  hr4 = 1*(lis%t%hr/1)
  mn4 = 0
  ss4 = 0
  ts4 = 1*60*60
  call tick( ftime_pers, doy4, gmt4, yr4, mo4, da4, hr4, mn4, ss4, ts4 )

!-----------------------------------------------------------------------
! Ensure that data is found during first time step
!-----------------------------------------------------------------------
if ( lis%f%gpcpsrc.eq.2 .and. get_nstep(lis%t).eq. 1 ) endtime_pers = 1
!-----------------------------------------------------------------------
! Check for and get Persiann Precipitation data
!-----------------------------------------------------------------------
if (lis%f%gpcpsrc.eq.2) then
   if ( ctime > persdrv%perstime ) endtime_pers = 1
   if ( endtime_pers == 1 ) then  !get new time2 data
      ferror_pers = 0
      call persfile( name, persdrv%persdir, yr4, mo4, da4, hr4 )
      print*, 'Getting new PERSIANN precip data',name
      call glbprecip_pers( name, ferror_pers )
      persdrv%perstime = ftime_pers
   endif  !need new time2
endif

return
```

1.64.4 persfile (Source File: getpers.F90)

This subroutine puts together PERSIANN file name for 1 hour file intervals

INTERFACE:

```
 subroutine persfile( name, persdir, yr, mo, da, hr)
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (i4, i2, i2, i2)
95 format (10a1)
96 format (a40)
97 format (a8)
98 format (a1, i4, i2, a1)
99 format (8a1)
89 format (a7)
```

```
!-----------------------------------------------------------------------
! Make variables for the time used to create the file
! We don't want these variables being passed ou
!-----------------------------------------------------------------------
  uyr = yr
  umo = mo
  uda = da
  uhr = 1*(hr/1)  !hour needs to be a multiple of 3 hours
  umn = 0
  uss = 0
  ts1 = -24*60*60 !one day interval to roll back date.

  open(unit=90, file='temp', form='formatted',access='direct', recl=80)
  write(90, 96, rec=1) persdir
  read(90, 92, rec=1) (fbase(i), i=1,80)

  write(90, 98, rec=1) '/', uyr, umo, '/'
  read(90, 99, rec=1) fdir
  do i = 1, 8
     if ( fdir(i) == ' ' ) fdir(i) = '0'
  end do

  write(90, 94, rec=1) uyr, umo, uda, uhr
  read(90, 95, rec=1) ftime
  do i = 1, 10
   if ( ftime(i) == ' ' ) ftime(i) = '0'
  end do

  write(90, 97, rec=1) '.lr_budi'
  read (90, 92, rec=1) (fsubs(i), i=1,8)
  c = 0
  do i = 1, 80
    if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
  end do

  write(90, 92, rec=1) (fbase(i),i=1,c),(fdir(i),i=1,8),  &
                       (ftime(i), i=1,10), (fsubs(i), i=1,8)

  read(90, 93, rec=1) name

  close(90)

  return
```

1.64.5 glbprecip_pers.F90 (Source File: glbprecip_pers.F90)

Includes reading routines for global PERSIANN precipitation product Used instead of GDAS/GEOS precipitation forcing

REVISION HISTORY:

```
17 Jul 2001: Jon Gottschalck; Initial code
04 Feb 2002: Jon Gottschalck; Added necessary code to use global precip
             observations with domain 3 (2x2.5)
30 Jul 2002: Jon Gottschalck; Added code to use Huffman and Persiann precip data
```

INTERFACE:

```
subroutine glbprecip_pers ( ld, gindex, name_pers, ferror_pers )
```

USES:

```
  use lis_module
  use obsprecipforcing_module, only: obsprecip
```

CONTENTS:

```
  fname = name_pers
  obsprecip = -1.0
!-----------------------------------------------------------------------
! Determine offset in number of rows from 60 S
! since PERSIANN starts at 50 S
!-----------------------------------------------------------------------
  open(unit=10,file=fname, status='old',access='direct', &
      form='unformatted',recl=xd*yd*4,iostat=ios)

  if (ios .eq. 0) then
    read (10,rec=1) precip

    do i = 1,yd
       do j = 1,xd
          index = gindex(j,i)
          col(index) = j
          row(index) = i
          if (index .ne. -1) then
             obsprecip(index) = precip(j,i)
          endif
       enddo
    enddo
    ferror_pers = 1
    close(10)
    print*, "Obtained PERSIANN precipitation data ", fname
 else
    print*, "Missing PERSIANN precipitation data ", fname
    ferror_pers = 0
 endif
```

1.65 Fortran: Module Interface persdomain_module.F90 (Source File: persdomain_module.F90)

Contains routines and variables that define the native domain for PERS precipitation product.

INTERFACE:

```
module persdomain_module
```

USES:

```
use persdrv_module
```

ARGUMENTS:

```
type(persdrvdec) :: persdrv
integer :: mi
real, allocatable :: rlat(:)
real, allocatable :: rlon(:)
integer, allocatable :: n11(:)
integer, allocatable :: n12(:)
integer, allocatable :: n21(:)
integer, allocatable :: n22(:)
real, allocatable ::  w11(:),w12(:)
real, allocatable ::  w21(:),w22(:)
```

1.65.1 defnatpers.F90 (Source File: persdomain_module.F90)

Defines the kgds array describing the native forcing resolution for PERS data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatpers()
```

USES:

```
use lisdrv_module, only: lis
implicit none
```

ARGUMENTS:

```
integer :: kgdsi(200)
```

CONTENTS:

```
call readperscrd(persdrv)
kgdsi = 0
call allocate_pers_ip(lis%d%lnc*lis%d%lnr)
```

1.65.2 allocate_pers_ip (Source File: persdomain_module.F90)

Allocates memory for PERS interpolation variables

INTERFACE:

```
subroutine allocate_pers_ip(N)
```

CONTENTS:

```
allocate(rlat(n))
allocate(rlon(n))
allocate(n11(n))
allocate(n12(n))
allocate(n21(n))
allocate(n22(n))
allocate(w11(n))
allocate(w12(n))
allocate(w21(n))
allocate(w22(n))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.66 Fortran: Module Interface persdrv_module.F90 (Source File: persdrv_module.F90)

Module for runtime specific PERS variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module persdrv_module
```

ARGUMENTS:

```
type persdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   character*40 :: persdir   !PERS Forcing Directory
   real*8 :: perstime
end type persdrvdec
```

1.66.1 readperscrd.F90 (Source File: readperscard.F90)

Routine to read PERS specific parameters from the card file.

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readperscrd(persdrv)
```

USES:

```
use persdrv_module
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=pers)
  print*,'Using PERS forcing'
  print*, 'PERS forcing directory :',persdrv%PERSDIR
!-----------------------------------------------------------------------
! Setting global observed precip times to zero to ensure
! data is read in during first time step
!-----------------------------------------------------------------------
  persdrv%perstime = 0.0
  close(11)
```

1.66.2 time_interp_pers.F90 (Source File: time_interp_pers.F90)

Calls the post processing utilities for handling observed precipitation data for PERS

INTERFACE:

```
subroutine time_interp_pers
```

CONTENTS:

```
! To be implemented
```

1.66.3 agrlwdn.F90 (Source File: agrlwdn.F90)

to compute the net downward longwave radiation at the earth's surface.
method:
======
- calculate the emissivity of the clear sky.
- calculate downwelling longwave radiation of the clear sky.
- add contribution of low, middle and high clouds to the
clear sky portion.
process narrative: flux3 - located in the flux3 sdf in dnxm
==========

references:
========== dr idso's paper in the j. of geophys. research,
no 74, pp 5397-5403.

dr r.f.wachtmann's paper in the digest of preprints, topical meeting on remote sensing of the atmosphere, anaheim,ca, optical society of america, entitled, "expansion of atmospheric temperature-moisture profiles in empirical orthogonal functions for remote sensing applications", 1975

INTERFACE:

```
subroutine agrlwdn( sfctmp, e, iclamt, rldown )
```

REVISION HISTORY:

```
15 may 1988  initial version.......................capt rice/sddc
07 sep 1999  ported to ibm sp-2.  added intent attributes to
             arguments.  updated prolog..............mr gayno/dnxm
25 oct 2001  implement in LDAS....................jesse meng/ncep

implicit none
```

INPUT PARAMETERS:

```
real,     intent(in)         :: iclamt  ( 3 )
real,     intent(in)         :: e
real,     intent(in)         :: sfctmp
```

OUTPUT PARAMETERS:

```
real,     intent(out)        :: rldown
```

CONTENTS:

```
!     ------------------------------------------------------------------
!     executable code starts here...compute the cloud amount
!     in fraction of overcast (.1 to 1.0).
!     ------------------------------------------------------------------
```

```
      cldfrt(1) =  iclamt(1)  / 100.0
      cldfrt(2) =  iclamt(2)  / 100.0
      cldfrt(3) =  iclamt(3)  / 100.0

!     ------------------------------------------------------------------
!     convert vapor pressure units from pascals to millibars for use
!     in determining emissivities.
!     ------------------------------------------------------------------

      emb = e * 0.01

!     ------------------------------------------------------------------
!     compute the effective clr sky emissivity for all wavelengths
!     (emissa) using idso's equation.
!     ------------------------------------------------------------------

      emissa = 0.700 + (5.95e-5 * emb * exp(1500 / sfctmp))

!     ------------------------------------------------------------------
!     use emissa in wachtmann's model for sky irradiance to calc a
!     resultant longwave downward radiation value.  first calc a sasc
!     emmisivity (emissb), which is an adjusted idso emmisivity.
!     then use emissb to calculate the blackbody irradiance of the
!     clear sky (the 1st term of wachtmann's equation).
!     ------------------------------------------------------------------

      emissb = -0.792 + (3.161 * emissa) - (1.573 * emissa * emissa)
      clrsky =  emissb * sigma * ( sfctmp * sfctmp * sfctmp * sfctmp )

!     ------------------------------------------------------------------
!     now compute the irradiance contribution from the low, middle,
!     and hi cloud layers (the 2nd thru 4th terms in wachtmann' eqn).
!     ------------------------------------------------------------------

      lcterm = (80.0 - (5.0 * zl)) * cldfrt(1)
      mcterm = (80.0 - (5.0 * zm)) * (1.0 - cldfrt(1)) * cldfrt(2)
      hcterm = (80.0 - (5.0 * zh)) * (1.0 - cldfrt(1)) * &
     &          (1.0 - cldfrt(2)) * cldfrt(3)

!     ------------------------------------------------------------------
!     put it all together to get a resultant downwrd longwave irrad.
!     ------------------------------------------------------------------

      rldown = clrsky + hcterm + mcterm + lcterm

      return
```

1.67 Fortran: Module Interface agrmetdomain_module.F90 (Source File: agrmetdomain_module.F90)

Contains routines and variables that define theb native domain for AGRMET observed radiation forcing

INTERFACE:

```
module agrmetdomain_module
```

USES:

```
use agrmetdrv_module
```

1.67.1 defnatagrmet.F90 (Source File: agrmetdomain_module.F90)

Defines the gridDesc array describing the native forcing resolution for AGRMET data.

REVISION HISTORY:

```
11Dec2003: Sujay Kumar; Initial Specification
```

INTERFACE:

```
subroutine defnatagrmet()
```

USES:

```
use lisdrv_module, only :lis
use lis_indices_module
implicit none
```

ARGUMENTS:

```
real :: gridDesci(50)
```

CONTENTS:

```
call readagrmetcrd(agrmetdrv)
gridDesci = 0
gridDesci(1) = 0
gridDesci(2) = 1440
gridDesci(3) = 600
gridDesci(4) = -59.875
gridDesci(5) = -179.875
gridDesci(6) = 128
gridDesci(7) = 89.875
gridDesci(8) = 179.875
gridDesci(9) = 0.250
```

```
gridDesci(10) = 0.250
gridDesci(11) = 64
gridDesci(20) = 255
call allocate_agr_ip(lis_nc_working*lis_nr_working)
mo =lis_nc_working*lis_nr_working
call def_agr_ip_input(gridDesci)
```

1.67.2 allocate_agr_ip (Source File: agrmetdomain_module.F90)

Allocate memory for AGRMET interpolation variables

INTERFACE:

```
subroutine allocate_agr_ip(N)
```

CONTENTS:

```
allocate(rlat(N))
allocate(rlon(N))
allocate(N11(N))
allocate(N12(N))
allocate(N21(N))
allocate(N22(N))
allocate(w11(N))
allocate(w12(N))
allocate(w21(N))
allocate(w22(N))
mo = n
nn = n
w11 = 0.0
w12 = 0.0
w21 = 0.0
w22 = 0.0
```

1.67.3 def_agr_ip_input (Source File: agrmetdomain_module.F90)

Calculates weights and neighbor information required for AGRMET interpolation

INTERFACE:

```
subroutine def_agr_ip_input (gridDesc)
```

USES:

```
use spmdMod
use lisdrv_module, only:lis
use lis_indices_module
```

CONTENTS:

```
!-----------------------------------------------------------------------
!  Calls the routines to decode the grid description and
!  calculates the weights and neighbor information to perform
!  spatial interpolation. This routine eliminates the need to
!  compute these weights repeatedly during interpolation.
!-----------------------------------------------------------------------
#if ( ! defined OPENDAP )
    if(masterproc) then
#endif
       gridDesco = lis%d%gridDesc
       mo = lis_nc_working*lis_nr_working
       if(gridDesco(1).ge.0) then
          call compute_coord(gridDesco, 0,mo,fill,xpts,ypts,rlon,rlat,nn,0)
       endif

       call compute_coord(gridDesc,-1,nn,fill,xpts,ypts,rlon,rlat,nv,0)
       do n=1,nn
          xi=xpts(n)
          yi=ypts(n)
          if(xi.ne.fill.and.yi.ne.fill) then
             i1=xi
             i2=i1+1
             j1=yi
             j2=j1+1
             xf=xi-i1
             yf=yi-j1
             n11(n)=get_fieldpos(i1,j1,gridDesc)
             n21(n)=get_fieldpos(i2,j1,gridDesc)
             n12(n)=get_fieldpos(i1,j2,gridDesc)
             n22(n)=get_fieldpos(i2,j2,gridDesc)
             if(min(n11(n),n21(n),n12(n),n22(n)).gt.0) then
                w11(n)=(1-xf)*(1-yf)
                w21(n)=xf*(1-yf)
                w12(n)=(1-xf)*yf
                w22(n)=xf*yf
             else
                n11(n)=0
                n21(n)=0
                n12(n)=0
                n22(n)=0
             endif
          else
             n11(n)=0
             n21(n)=0
             n12(n)=0
             n22(n)=0
```

```
          endif
       enddo
       mi = 864000
#if ( ! defined OPENDAP )
    endif
#endif
```

1.68 Fortran: Module Interface agrmetdrv_module.F90 (Source File: agrmetdrv_module.F90)

Module containing runtime specific AGRMET variables

REVISION HISTORY:

```
11 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module agrmetdrv_module
```

ARGUMENTS:

```
type agrmetdrvdec
   integer :: ncold, nrold   !AWIPS 212 dimensions
   integer :: nmif
   character*40 :: agrmetdir   !AGRMET Forcing Directory
   real*8 :: agrmtime1,agrmtime2
end type agrmetdrvdec
```

1.69 Fortran: Module Interface agrmetopendap_module.F90 (Source File: agrmetopendap_module.F90)

This module contains routines needed to initialize and control variables required for the execution of GDS-based I/O specfific to AGRMET forcing routines

REVISION HISTORY:

```
29 Apr 2004: James Geiger;  Initial Specification
```

INTERFACE:

```
module agrmetopendap_module
#if ( defined OPENDAP )
  TODO: Stubs to allow the code to compile
  TODO: AGRMET does not yet work in GDS-mode
   integer :: agrmet_time_index
```

```
    contains
    subroutine set_agrmet_index(offset)
      integer, intent(in) :: offset   ! offset from current date in hours
      agrmet_time_index = -9999
    end subroutine set_agrmet_index
    subroutine opendap_agrmet_init()
    end subroutine opendap_agrmet_init
 #endif
 #if ( defined OPENDAP_KLUGE )
   TODO: Correct this section
```

USES:

```
   use lisdrv_module,       only : lis
   use spmdMod,             only : iam
   use agrmetdomain_module, only : agrmetdrv
   use opendap_module,      only : opendap_data_prefix, ciam

   implicit none
    type(esmf_date), save :: agrmet_ref_date !Reference date for AGRMET forcing data
    integer :: agrmet_time_index

    contains

%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\

\subsubsection{opendap\_agrmet\_init (Source File: agrmetopendap\_module.F90)}



    Initializes the AGRMET-GDS variables

\bigskip{\sf INTERFACE:}
\begin{verbatim}    subroutine opendap_agrmet_init()
```

CONTENTS:

```
     !call init_agrmet_vars()
     call reset_agrmet_filepaths()
```

1.69.1 reset_agrmet_filepaths (Source File: agrmetopendap_module.F90)

Resets input data filenames for AGRMET forcing for execution through GDS

INTERFACE:

```
subroutine reset_agrmet_filepaths()
```

CONTENTS:

```
agrmetdrv%agrmetdir  = trim(opendap_data_prefix)//'/'// &
                       trim(adjustl(ciam))//'/'//agrmetdrv%agrmetdir
```

1.69.2 init_agrmet_ref_date (Source File: agrmetopendap_module.F90)

Initializes the reference date for AGRMET forcing data

INTERFACE:

```
subroutine init_agrmet_ref_date()
```

CONTENTS:

```
agrmet_ref_date = esmf_dateinit(esmf_no_leap, ref_ymd, ref_tod, rc)
```

1.69.3 set_agrmet_index (Source File: agrmetopendap_module.F90)

Computes the time-based index for AGRMET forcing data

INTERFACE:

```
subroutine set_agrmet_index(offset)
  implicit none
```

INPUT PARAMETERS:

```
integer, intent(in) :: offset   ! offset from current date in hours
```

CONTENTS:

```
if ( ref_data_uninit ) then
   print*,'DBG: set_agrmet_index -- initializing ref date',' (', iam, ')'
   call init_agrmet_ref_date()
   ref_data_uninit = .false.
endif
ymd = ( lis%t%yr * 10000 ) + ( lis%t%mo * 100 ) + lis%t%da
tod = ( lis%t%hr *  3600 ) + ( lis%t%mn *  60 ) + lis%t%ss
current_date = esmf_dateinit(esmf_no_leap, ymd, tod, rc)

diff = esmf_timeinit()
call esmf_datediff(current_date, agrmet_ref_date, diff, islater, rc)
call esmf_timeget(diff, ndays, nsecs, rc)

agrmet_time_index = ( (ndays * 24) + (nsecs / 3600) + offset ) + 1
print*,'DBG: set_agrmet_index -- agrmet_time_index = ', &
       agrmet_time_index, ' (', iam, ')'
```

1.69.4 getgrad.F90 (Source File: getgrad.F90)

Opens, reads, interpolates and overlays radiation forcing.
TIME1 = most recent past data TIME2 = most recent future data

REVISION HISTORY:

```
28  Oct 1999: Brian Cosgrove; Initial code, see getrad.f
27  Apr 2000: Brian Cosgrove' Disabled zenith angle correction cutoff for
              cos(zen) less than .2
11  May 2000: Brian Cosgrove; Enabled correction cutoffs for cos(zen) less
              than .1, stop model if computed value is greater than 1367 w/m2
08  Jan 2001: Brian Cosgrove; Added check to see if czb or czm is equal
              to zero before trying to divide by czm or czb.  If it is
              zero, set radiation value to zero
06  Mar 2001: Brian Cosgrove; Changed computation of WT1 and WT2 in cases
              where the previous hour or the next hour of observed radiation
              is not available.  Substituted TIME1 for LDAS%PINKTIME1 and
              LDAS%NESTIME1 and TIME2 for LDAS%PINKTIME2 and LDAS%NESTIME2
15  Jun 2001: Urszula Jambor; Reworked algorithm for AGRMET data & GLDAS.
15  Oct 2001: Jesse Meng; Replace lis%agrmet flag by lis%agrmetsw and
              lis%agrmetlw;  added call retagrlw() to calculate
              AGRMET LW;
25  Feb 2002: Urszula Jambor; check on both SW & LW file status before
              using
04  Jun 2002: Urszula Jambor; allowed fall back to model SW.
10  Dec 2002: Urszula Jambor; replaced lis%astat1,2 with local sstat1,2
              Reorganized routine to mirror other get-routines,and
              corrected bug in file status check for initial time1.
```

INTERFACE:

```
subroutine getgrad
```

USES:

```
  use lisdrv_module, only : lis, grid
  use obsradforcing_module, only : obswdata1,obswdata2,oblwdata1,oblwdata2,&
       sstat1,sstat2,lstat1,lstat2
  use time_manager
  use agrmetdomain_module, only : agrmetdrv
#if ( defined OPENDAP )
  use agrmetopendap_module, only : set_agrmet_index
  use spmdMod
#endif
  implicit none
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Determine Required Observed Radiation Data Times
!-----------------------------------------------------------------------
  yr1 = lis%t%yr    !Previous Hour
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = 0
  ss1 = 0
  ts1 = 0
  call tick ( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )

  yr2 = lis%t%yr    !Next Hour
  mo2 = lis%t%mo
  da2 = lis%t%da
  hr2 = lis%t%hr
  mn2 = 0
  ss2 = 0
  ts2 = 60*60
  call tick ( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )

  lis%f%findagrtime1=0
  lis%f%findagrtime2=0
  movetime=0

  if(lis%t%time.ge.agrmetdrv%agrmtime2) then
     movetime=1
     lis%f%findagrtime2=1
  endif

#if ( defined OPENDAP )
  if ( masterproc ) then
     nstep = get_nstep(lis%t)
  endif
#if ( defined SPMD )
  call MPI_BCAST(nstep, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
#endif
#else
  nstep = get_nstep(lis%t)
#endif
  if ( nstep == 0 .or. nstep == 1 ) then
     lis%f%findagrtime1=1
     lis%f%findagrtime2=1
     movetime=0
  endif

  if (lis%f%findagrtime1==1) then
#if ( defined OPENDAP )
```

```
     call set_agrmet_index(0)
#endif
     sstat1 = 0
     sstat2 = 0
     lstat1 = 0
     lstat2 = 0
     call agrSWfile ( nameSH, lis, yr1, mo1, da1, hr1 )
     print*, 'using AGRMET ',nameSH
     sstat1 = 0
     call retglbSW ( 1, nameSH, sstat1, 1 )
     lstat1 = 0
     call retagrlw ( 1, yr1, mo1, da1, hr1, lstat1, 1 )
     if ((sstat1 + lstat1) < 2) then
        sstat1 = 0
        lstat1 = 0
     end if
     if (sstat1 /= 0) agrmetdrv%agrmtime1 = time1
  endif
  if(movetime.eq.1) then
     agrmetdrv%agrmtime1 = agrmetdrv%agrmtime2
     sstat1 = sstat2
     lstat1 = lstat2
     do c=1, lis%d%ngrid
           obswdata1(c) = obswdata2(c)
           oblwdata1(c) = oblwdata2(c)
     end do
  endif

  if(lis%f%findagrtime2.eq.1) then
#if ( defined OPENDAP )
     call set_agrmet_index(1)
#endif
     sstat2 = 0
     lstat2 = 0
     call agrSWfile ( nameSH, lis, yr2, mo2, da2, hr2 )
     sstat2 = 0
     print*, 'using AGRMET ',nameSH
     call retglbSW ( 2, nameSH, sstat2, 1 )
     lstat2 = 0
     call retagrlw ( 2, yr2, mo2, da2, hr2, lstat2, 1 )

     if ((sstat2 + lstat2) < 2) then
        sstat2 = 0
        lstat2 = 0
     end if
     if (sstat2 /= 0) agrmetdrv%agrmtime2 = time2
  endif
!-----------------------------------------------------------------------
```

```
! Print out Status of data holdings
!-----------------------------------------------------------------------
  if (lis%t%time == time1) then
     if (sstat1==0) write(*,*) 'NO AGR SW USED',mo1,da1,yr1,hr1
     if (sstat1/=0) write(*,*) 'USED AGRMET SW',mo1,da1,yr1,hr1
     if (sstat2==0) write(*,*) 'NO AGR SW USED',mo2,da2,yr2,hr2
     if (sstat2/=0) write(*,*) 'USED AGRMET SW',mo2,da2,yr2,hr2

     if (lstat1==0) write(*,*) 'NO AGR LW USED',mo1,da1,yr1,hr1
     if (lstat1/=0) write(*,*) 'USED AGRMET LW',mo1,da1,yr1,hr1
     if (lstat2==0) write(*,*) 'NO AGR LW USED',mo2,da2,yr2,hr2
     if (lstat2/=0) write(*,*) 'USED AGRMET LW',mo2,da2,yr2,hr2
  endif
  return
```

1.69.5 agrSWfile: (Source File: getgrad.F90)

This subroutine puts together the radiation data filenames.

REVISION HISTORY:

```
    28  Oct 1999: Brian Cosgrove; Initial code
    18  Jun 2001: Urszula Jambor; Modified for AGRMET data use in GLDAS
    24  Oct 2001: Jesse Meng; Modified for AGRMET directory structure
    15  Aug 2003: Sujay Kumar: Modified to create a global filename
                  instead of two filenames for each hemisphere.
```

INTERFACE:

```
 subroutine agrSWfile ( nameSH, lis, yr, mo, da, hr )
```

USES:

```
   use lis_module
   use agrmetdomain_module, only : agrmetdrv
```

CONTENTS:

```
92 format (80a1)
93 format (a80)
94 format (a5, i4, i2, i2, i2)
95 format (15a1)
96 format (a40)
97 format (a1)
98 format (a6, i4, i2, a1)
99 format (13a1)

  open(unit=81, file='temp', form='formatted', access='direct', recl=80)
```

```
write(81,96,rec=1) agrmetdrv%agrmetdir
read(81,92,rec=1) (fbase(i), i=1,80)

write(81,98,REC=1)'/SWDN/',yr,mo,'/'
read(81,99,rec=1) fdir
do i = 1, 13
   if ( fdir(i) == ' ' ) fdir(i) = '0'
end do
write(81,94,rec=1) 'swdn_', yr, mo, da, hr
read(81,95,rec=1) ftime
do i = 1, 15
   if ( ftime(i) == ' ' ) ftime(i) = '0'
end do
c = 0
do i = 1, 80
   if ( (fbase(i) == ' ') .and. (c == 0) ) c = i-1
end do

write(81, 92, rec=1) (fbase(i),i=1,c), (fdir(i),i=1,13), &
     (ftime(i),i=1,15)
read(81, 93, rec=1) nameSH
close(81)

return
```

1.69.6 subroutine interp_agrmet_lw.F90 (Source File: interp_agrmet_lw.F90)

Opens, reads, and interpolates AGRMET longwave radiation
GRIB IPOLATES UTILITY TO INTERPOLATE DATA FOR NH AND SH IN RT-NEPH (512X512) POLAR STEREOGRAPHIC GRIDS TO MERGED GLOBAL DATA IN GLDAS-SPECIFIED LAT/LON GRIDS;

REVISION HISTORY:

```
26 Jun 2001: Urszula Jambor; Initial code, based on Jesse Meng's
             RTNEPH2LATLON.F code.
08 Feb 2002: Urszula Jambor; Modified declarations of arrays
             dependant on domain & resolution to allocatable.
             Pass in values for latmax.
11 Dec 2002: Urszula Jambor; Added 1/2 & 1 degree resolution GDS arrays
```

INTERFACE:

```
subroutine interp_agrmet_lw( pdata1, outdata, ferror )
```

USES:

```
use lisdrv_module,only : lis,gindex
use agrmetdomain_module, only : rlat,rlon,w11,w12,w21,w22,n11,n12,n21,&
     n22, mi,mo
use lis_indices_module
implicit none

integer, parameter :: nagrc = 1440, nagrr=600
```

ARGUMENTS:

```
real :: pdata1(nagrc, nagrr)
real :: outdata(lis%d%ngrid)  !output array matching grid(c,r)
integer :: ferror                 !set to zero if error found
```

CONTENTS:

```
!-----------------------------------------------------------------------
! READ INPUT DATA
!-----------------------------------------------------------------------
  allocate(pdata(mi))
  allocate(lo1(lis_nc_working*lis_nr_working))
  allocate(ldata1(lis_nc_working*lis_nr_working))
  if (ferror == 0) then
     do i=1,lis%d%ngrid
        outdata(i) = lis%d%udef
     end do
  else
     ferror = 1
     ibi = 1
     count = 0
     li1 = .false.
     do j=1,nagrr
        do i=1,nagrc
           pdata(count+i) = pdata1(i,j)
        enddo
        count = count+nagrc
     enddo
     do i=1,mi
        if(pdata(i).eq.-9999) then
           li1(i) = .false.
        else
           li1(i) = .true.
        endif
     enddo
     gridDesco = 0
     gridDesco = lis%d%gridDesc
     call bilinear_interp(gridDesco,ibi,li1,pdata,ibo,lo1,ldata1,mi,mo,&
          rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)
```

```
     if(iret .NE. 0) then
        print*, "IPOLATES ERROR!! PROGRAM STOP!!"
        call exit(iret)
     end if
!-----------------------------------------------------------------------
! COMBINE LDATA1 AND LDATA2 INTO LDATA
!-----------------------------------------------------------------------
     count = 0
     do j=1,lis_nr_working
        do i=1,lis_nc_working
           if(gindex(i,j).ne. -1) then
              outdata(gindex(i,j)) = ldata1(i+count)
           endif
        enddo
        count = count+lis_nc_working
     enddo
  endif
  deallocate(pdata)
  deallocate(lo1)
  deallocate(ldata1)
```

1.69.7 interp_agrmet_sw.F90 (Source File: interp_agrmet_sw.F90)

Opens, reads, and interpolates AGRMET shortwave radiation forcing

REVISION HISTORY:

```
    26 Jun 2001: Urszula Jambor; Initial code, based on Jesse Meng's
                 RTNEPH2LATLON.F code.
    08 Feb 2002: Urszula Jambor; Modified declarations of arrays
                 dependant on domain & resolution to allocatable.
                 Pass in values for latmax.
    11 Dec 2002: Urszula Jambor; Added 1/2 & 1 degree resolution GDS arrays
```

INTERFACE:

```
 subroutine interp_agrmet_sw( nameSH, outdata, ferror )
```

USES:

```
   use lisdrv_module,only : lis,gindex
   use agrmetdomain_module, only : rlat,rlon,w11,w12,w21,w22,n11,n12,n21,&
        n22, mi, mo
   use lis_openfileMod
   use lis_indices_module
   implicit none
```

ARGUMENTS:

```
   character*80 :: nameSH
   real :: outdata(lis%d%ngrid)
   integer :: ferror
```

CONTENTS:

```
  allocate(pdata(mi))
  allocate(lo1(lis_nc_working*lis_nr_working))
  allocate(ldata1(lis_nc_working*lis_nr_working))
! Add file existance test to avoid hung up due to missing file 12/21/04 HK
  inquire(file=nameSH,exist=ex)
  if ( ex ) then     !file exists
!<kluge -- swdn has little endian record count, big endian data>
!  call lis_open_file(11, file=nameSH, form='unformatted', &
!       script='getagrmet_sw.pl')
!  read(11, iostat=readerrS) pdata1
!  close(11)
  print*, 'Reading AGRMET file : ',nameSH
call lis_open_file(11, file=nameSH, form='unformatted', &
       access='direct',recl=4,script='getagrmet_sw.pl')
read(11,rec=1) count
count = 2
do j=1,nagrr
do i=1,nagrc
read(11,rec=count) pdata1(i,j)
count = count+1
enddo
enddo
count = 0
close(11)
!</kluge>
  print*,'read SW agrmet',readerrS
  else              ! file doesn't exists
  openerrN = 20
  endif             ! if ( ex ) then

  if ((openerrN+openerrS+readerrN+readerrS) > 0) then
     ferror = 0
     if ((openerrS+readerrS) > 0) then
        print*, 'AGRMET file problem:', nameSH
     end if
     if (openerrN == 20) then
        print*, 'AGRMET file missing:', nameSH
     end if
     do i=1,lis%d%ngrid
        outdata(i) = lis%d%udef
     end do
```

```
      openerrN = 0
   else
      ferror = 1
      ibi = 1
      count = 0
      li1 = .false.
      do j=1,nagrr
         do i=1,nagrc
            pdata(count+i) = pdata1(i,j)
         enddo
         count = count+nagrc
      enddo
      do i=1,mi
         if(pdata(i).eq.-9999) then
            li1(i) = .false.
         else
            li1(i) = .true.
         endif
      enddo
      gridDesco = 0
      gridDesco = lis%d%gridDesc
      call bilinear_interp(gridDesco,ibi,li1,pdata,ibo,lo1,ldata1,mi,mo,&
           rlat,rlon,w11,w12,w21,w22,n11,n12,n21,n22,iret)

      if(iret .NE. 0) then
         print*, "IPOLATES ERROR!! PROGRAM STOP!!"
         call exit(iret)
      end if
      count = 0
      do j=1,lis_nr_working
         do i=1,lis_nc_working
            if(gindex(i,j).ne. -1) then
               outdata(gindex(i,j)) = ldata1(i+count)
            endif
         enddo
         count = count+lis_nc_working
      enddo
   endif
   deallocate(pdata)
   deallocate(lo1)
   deallocate(ldata1)
```

1.69.8 readagrmetcrd.F90 (Source File: readagrmetcrd.F90)

Routine to read AGRMET specific parameters from the card file.

REVISION HISTORY:

```
   11 Dec 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine readagrmetcrd(agrmetdrv)
```

USES:

```
   use agrmetdrv_module
 #if ( defined OPENDAP )
     use agrmetopendap_module, only : opendap_agrmet_init
 #endif
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=agrmet)
  print*,'Using AGRMET forcing'
  print*, 'AGRMET forcing directory :',agrmetdrv%AGRMETDIR
  agrmetdrv%AGRMTIME1  = 3000.0
  agrmetdrv%AGRMTIME2  = 0.0

  close(11)
#if ( defined OPENDAP )
    call opendap_agrmet_init()
#endif
```

1.69.9 retagrlw.F90 (Source File: retagrlw.F90)

Opens, reads, interpolates and overlays LW radiation forcing.
TIME1 = most recent past data
TIME2 = most recent future data

1. Load AGRMET 3 level cloud amount (convertted from RTNEPH)
2. Convert 2 m forcing specific humidity to vapor pressure
3. Use AGRMET subroutine longwv() to calculate LW DOWN. Arguments are:
cloud amount [%], 2 m temperature [K], and 2 m vapor pressure [Pa]

REVISION HISTORY:

```
    28 Oct 1999: Brian Cosgrove; Initial code
    11 Apr 2000: Brian Cosgrove; changed code to use Forcing Mask (With inland
                 water filled in).  Deleted unused variables.
    20 Jun 2000: Brian Cosgrove; changed code so that it uses LDAS%UDEF and
                  not a hard-wired undefined value of -999.9 and -999.0
    25 Jun 2001: Urszula Jambor; Modified for AGRMET data use in GLDAS.
    25 Oct 2001: Jesse Meng; Modified for AGRMET LW scheme implementation
```

INTERFACE:

```
subroutine retagrlw ( order,yr, mo, da, hr, ferror, flag )
```

USES:

```
   use lisdrv_module, only : lis, grid
   use obsradforcing_module, only : oblwdata1,oblwdata2
   use agrmetdomain_module, only : agrmetdrv
   use lis_openfilemod
   implicit none
```

ARGUMENTS:

```
   integer :: yr, mo, da, hr
   integer :: order              !retrieve data for time1 or time2
   integer :: ferror             !0=no radiation data found
                                 !1=found observational data
```

CONTENTS:

```
!-------------------------------------------------------------------------
! If using AGRMET data, open, read in and interpolate AGRMET files
!   to appropriate GLDAS resolution;
! If error reading file, or data completely missing, ferror=0;
!------------------------------------------------------------------------
  ferror = 0
  do c = 1, lis%d%ngrid
     if (order == 1) then
        oblwdata1(c) = lis%d%udef
     else if (order == 2) then
        oblwdata2(c) = lis%d%udef
     end if
  end do

  if (flag == 1) then
!------------------------------------------------------------------------
! 1. Generate AGRMET cloud filenames
! Load 3 layers cldamt data
!------------------------------------------------------------------------

     write(cyr, '(I4.4)') yr
     write(cmo, '(I2.2)') mo
     write(cda, '(I2.2)') da
     write(chr, '(I2.2)') hr

     nameSH = trim(agrmetdrv%agrmetdir)//'/CloudAGR/'//cyr//cmo// &
          '/cldamt_'//cyr//cmo//cda//chr
!     print*, 'Reading AGRMET file : ',nameSH
!     open(11, file=nameSH, status='old',access='direct',&
!          recl=3456000, iostat=openerrN)
```

```
     call lis_open_file(11, file=nameSH, status='old',access='direct',&
          recl=3456000, script='getagrmet_lw.pl')

     read(11, rec=1,iostat=readerrN) pdata1H
     read(11, rec=2,iostat=readerrN) pdata1M
     read(11, rec=3,iostat=readerrN) pdata1L

     close(11)
     if ((openerrN+readerrN) > 0) then
        ferror = 0
        print*, 'AGRMET file problem: ',nameSH
     else
        ferror = 1
     endif
     call interp_agrmet_lw( pdata1H,cldamtH, ferror )
     call interp_agrmet_lw( pdata1M,cldamtM, ferror )
     call interp_agrmet_lw(pdata1L,cldamtL, ferror )
     if ( ferror .EQ. 0 ) then
        call lis_log_msg('ERR: retagrlw -- ferror == 0')
        return
     endif
!-----------------------------------------------------------------------
! If AGRMET cloud data is not undefined, set ferror=1
!-----------------------------------------------------------------------
     fvalid = 0
     do c=1, lis%d%ngrid
        if (cldamtH(c) >= 0.0) fvalid = 1
        if (cldamtM(c) >= 0.0) fvalid = 1
        if (cldamtL(c) >= 0.0) fvalid = 1
     end do
     if ( fvalid .EQ. 0 ) then
        call lis_log_msg('ERR: retagrlw -- fvalid == 0')
        return
     endif
     do c=1,lis%d%ngrid
        rldown = 0.
           tair = grid(c)%forcing(1)
!-----------------------------------------------------------------------
! 2. CONVERT 2 METER SPECIFIC HUMIDITY TO VAPOR PRESSURE
!-----------------------------------------------------------------------
           vaporP = grid(c)%forcing(2) * grid(c)%forcing(7) / &
                    ( 0.622 + grid(c)%forcing(2) * (1-0.622) )
!-----------------------------------------------------------------------
! If tair, vaporP, and ALL 3 AGRMET LAYERS cldamt are defined,
! calculate rldown; transfer to proper output array
!-----------------------------------------------------------------------
           fvalid = 1
   if (tair.LT. 0.0) fvalid = 0
```

```
   if (vaporP.LT. 0.0) fvalid = 0
           if (cldamtH(c) .LT. 0.0) fvalid = 0
           if (cldamtM(c) .LT. 0.0) fvalid = 0
           if (cldamtL(c) .LT. 0.0) fvalid = 0

   if (fvalid == 1) then
              cldamt1d(1) = cldamtL(c)
      cldamt1d(2) = cldamtM(c)
      cldamt1d(3) = cldamtH(c)
!-----------------------------------------------------------------------
! 3. Calculate lw down
!-----------------------------------------------------------------------
              call agrlwdn( tair, vaporp, cldamt1d, rldown )
              if (order==1) then
                 oblwdata1(c) = rldown
              else if (order==2) then
                 oblwdata2(c) = rldown
              end if
           else
              if (order==1) then
                 oblwdata1(c) = lis%d%udef
              else if (order==2) then
                 oblwdata2(c) = lis%d%udef
              end if
           end if
        end do
     end if
  return
```

1.69.10 retglbSW.F90 (Source File: retglbSW.F90)

Opens, reads, and interpolates radiation forcing.
TIME1 = most recent past data
TIME2 = most recent future data

REVISION HISTORY:

```
    28 Oct 1999: Brian Cosgrove; Initial code, retrad.f
    25 Jun 2001: Urszula Jambor; Renamed & modified for AGRMET data
                 use in GLDAS.
    21 Nov 2001: Urszula Jambor; Added iostat check in open & read stmts;
                 renamed to distinguish SW from LW routines.
    27 Feb 2002: Urszula Jambor; Added call to fill_land to compensate
                 deficiencies in interpolation to coarse 2x2.5 grid.
    16 Oct 2002: Urszula Jambor; Corrected array ranges used in 2x2.5 case.
```

```
                Previously, array mismatch occurred.
    11 Dec 2002: Urszula Jambor; Added domain 4 and 5 in attached routine
```

INTERFACE:

```
 subroutine retglbSW ( order, nameSH, ferror, flag )
```

USES:

```
   use lisdrv_module,only  : lis, grid       ! LDAS non-model-specIFic 1-D variables
   use obsradforcing_module, only : obswdata1, obswdata2
   implicit none
```

ARGUMENTS:

```
   character*80 :: nameSH
   integer :: flag               !data source, 1=AGRMET
   integer :: order              !retrieve data for time1 or time2
   integer :: ferror             !0=no radiation data found
                                 !1=found observational data (may be udef)
```

CONTENTS:

```
!-----------------------------------------------------------------------
! If using AGRMET data, open, read in and interpolate AGRMET files
! to appropriate GLDAS resolution
! If error reading file, ferror=0 else ferror=1
!-----------------------------------------------------------------------
  if (flag == 1) then
     call interp_agrmet_sw( nameSH, outdata, ferror )
     do c=1, lis%d%ngrid
        if (order == 1) then
           obswdata1(c) = outdata(c)
        else if (order == 2) then
           obswdata2(c) = outdata(c)
        end if
     end do
  end if !flag=1
```

1.70 Fortran: Module Interface time_interp_agrmet.F90 (Source File: time_interp_agrmet.F9

This routine consists of utilities for temporal interpolation of radiation forcing
TIME1 = most recent past data TIME2 = most recent future data

REVISION HISTORY:

```
    28  Oct 1999: Brian Cosgrove; Initial code, see getrad.f
    27  Apr 2000: Brian Cosgrove' Disabled zenith angle correction cutoff for
                  cos(zen) less than .2
```

```
11  May 2000: Brian Cosgrove; Enabled correction cutoffs for cos(zen) less
              than .1, stop model if computed value is greater than 1367 w/m2
08  Jan 2001: Brian Cosgrove; Added check to see if czb or czm is equal
              to zero before trying to divide by czm or czb.  If it is
              zero, set radiation value to zero
06  Mar 2001: Brian Cosgrove; Changed computation of WT1 and WT2 in cases
              where the previous hour or the next hour of observed radiation
              is not available.  Substituted TIME1 for LDAS%PINKTIME1 and
              LDAS%NESTIME1 and TIME2 for LDAS%PINKTIME2 and LDAS%NESTIME2
15  Jun 2001: Urszula Jambor; Reworked algorithm for AGRMET data & GLDAS.
15  Oct 2001: Jesse Meng; Replace lis%agrmet flag by lis%agrmetsw and
              lis%agrmetlw;  added call retagrlw() to calculate
              AGRMET LW;
25  Feb 2002: Urszula Jambor; check on both SW & LW file status before
              using
04  Jun 2002: Urszula Jambor; allowed fall back to model SW.
10  Dec 2002: Urszula Jambor; replaced lis%astat1,2 with local sstat1,2
              Reorganized routine to mirror other get-routines,and
              corrected bug in file status check for initial time1.
```

INTERFACE:

```
subroutine time_interp_agrmet
```

USES:

```
  use lisdrv_module, only : lis, grid
  use obsradforcing_module, only : obswdata1, obswdata2,oblwdata1,oblwdata2,&
       sstat1,sstat2,lstat1,lstat2
  use grid_spmdMod
  use time_manager
  use agrmetdomain_module, only : agrmetdrv
  implicit none
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Loop through and replace data as possible with AGRMET data
! This depends on options specified in lis.crd as well as actual
! data holdings.
!-----------------------------------------------------------------------
  allocate(obsw(gdi(iam)))
  yr1 = lis%t%yr    !Previous Hour
  mo1 = lis%t%mo
  da1 = lis%t%da
  hr1 = lis%t%hr
  mn1 = 0
  ss1 = 0
  ts1 = 0
  call tick ( time1, doy1, gmt1, yr1, mo1, da1, hr1, mn1, ss1, ts1 )
```

```
  yr2 = lis%t%yr    !Next Hour
  mo2 = lis%t%mo
  da2 = lis%t%da
  hr2 = lis%t%hr
  mn2 = 0
  ss2 = 0
  ts2 = 60*60
  call tick ( time2, doy2, gmt2, yr2, mo2, da2, hr2, mn2, ss2, ts2 )
#if(defined SPMD)
  call MPI_BCAST(agrmetdrv%agrmtime1,1,MPI_REAL8,0,&
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(agrmetdrv%agrmtime2,1,MPI_REAL8,0,&
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%time,1,MPI_REAL8,0,&
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(lis%t%gmt,1,MPI_REAL,0,&
       MPI_COMM_WORLD,ier)
#endif
  if ((sstat1 == 1) .and. (sstat2 == 1)) then
!-----------------------------------------------------------------------------
! Compute weights and zenith angle information.  Replace forcing
! with AGRMET data.
!-----------------------------------------------------------------------------
     wt1 = (agrmetdrv%agrmtime2 - lis%t%time) / (agrmetdrv%agrmtime2 - agrmetdrv%agrmtime1)
     wt2 = 1.0 - wt1
     do c=1,gdi(iam)
        zdoy = lis%t%doy
        call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
             lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
        obsw(c) = lis%d%udef
        if ((obswdata1(c)>=0.0) .and. (obswdata2(c)>=0.0)) then
           obsw(c) = obswdata1(c)*zw1+obswdata2(c)*zw2
           if ((obsw(c) > obswdata1(c)  .and. &
                obsw(c) > obswdata2(c)) .and. &
                (czb<0.1 .or. cze<0.1)) then
              obsw(c) = obswdata1(c)*wt1+obswdata2(c)*wt2
           end if
        end if
        if ((obswdata1(c) >= 0.0) .and. &
             (obswdata2(c) < 0.0)      ) then
           if (czb > 0.0) then
              obsw(c) = obswdata1(c) * (czm/czb)
           else
              obsw(c) = obswdata1(c) * (0.0)
           end if
           if ((obsw(c) > obswdata1(c) .and. &
                obsw(c) > 0.0) .and. (czb<0.1 .or. cze<0.1)) then
```

```
            obsw(c) = obswdata1(c)*wt1 + 0.0*wt2
         end if
      end if
      if ((obswdata1(c) < 0.0) .and. (obswdata2(c) >= 0.0)) then
         if (cze > 0.0) then
            obsw(c) = obswdata2(c) * (czm/cze)
         else
            obsw(c) = obswdata2(c) * (0.0)
         end if
         if ((obsw(c)>0.0 .and. obsw(c)> obswdata2(c)) &
              .and. (czb < 0.1  .or.  cze < 0.1)) then
            obsw(c) = 0.0*wt1 + obswdata2(c)*wt2
         end if
      end if

      if (obsw(c) >= 0.0) then
         if (obsw(c) .gt. 1367) then
            print *,'FALLING BACK TO MODEL SW'
         else
            grid(c)%forcing(3) = obsw(c)
         end if
      end if
   end do !c
 end if

 if ((sstat1 == 1) .and. (sstat2 /= 1)) then
!-------------------------------------------------------------------------
! Compute weights and zenith angle information.  Replace forcing
! with zenith extrapolated AGRMET data
!-------------------------------------------------------------------------
   wt1 = (time2 - lis%t%time) / (time2 - agrmetdrv%agrmtime1)
   wt2 = 1.0 - wt1
   do c=1, gdi(iam)
      zdoy = lis%t%doy
      call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
           lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
      obsw(c) = lis%d%udef
      if (obswdata1(c) >= 0.0) then

         if (czb > 0.0) then
            obsw(c) = obswdata1(c)  * (czm/czb)
         else
            obsw(c) = obswdata1(c) * (0.0)
         end if
         if ((obsw(c) > 400.0) .and. &
              (czb < 0.1  .or.  cze < 0.1)  ) then
            obsw(c) = obswdata1(c)*wt1
         end if
```

```
          if (obsw(c) >= 0.0) then
             if (obsw(c) .gt. 1367) then
                print *,'warning, OBSERVED SW RADIATION HIGH'
                print *,'it is',grid(c)%forcing(3),' at',c
                print *,'agrm1=',obswdata1(c)
                print *,'agrm2=',obswdata2(c)
                print *,'wt1,wt2,czb,cze,czm,zw1,zw2'
                print *,wt1,wt2,czb,cze,czm,zw1,zw2
                print *,'FALLING BACK TO MODEL SW'
             else
                grid(c)%forcing(3) = obsw(c)
             end if
          end if
       end if
    end do !c
  end if

  if ((sstat1 /= 1) .and. (sstat2 == 1)) then
!-------------------------------------------------------------------------
! Compute weights and zenith angle information.  Replace forcing
! with zenith extrapolated AGRMET data
!-------------------------------------------------------------------------
     wt1 = (agrmetdrv%agrmtime2 - lis%t%time) / (agrmetdrv%agrmtime2 - time1)
     wt2 = 1.0 - wt1
     do c=1, gdi(iam)
        zdoy = lis%t%doy
        call zterp ( 1, grid(c)%lat, grid(c)%lon, gmt1, gmt2, &
             lis%t%gmt, zdoy, zw1, zw2, czb, cze, czm, lis)
        obsw(c) = lis%d%udef
        if (obswdata2(c) >= 0.0) then
           if (cze > 0.0) then
              obsw(c) = obswdata2(c) * (czm/cze)
           else
              obsw(c) = obswdata2(c) * (0.0)
           end if
           if ((obsw(c) > 400.0) .and. &
                (czb < 0.1  .or.  cze <0.1)   ) then
              obsw(c) = 0.0*wt1 + obswdata2(c)*wt2
           end if

           if (obsw(c) >= 0.0) then
              if (obsw(c) .gt. 1367) then
              else
                 grid(c)%forcing(3) = obsw(c)
              end if
           end if
        end if
```

```
    end do !c
 end if
 if ((sstat1 /= 1) .and. (sstat2 /= 1)) then
    do c=1, gdi(iam)
       obsw(c) = lis%d%udef
    end do
 end if
 if ((lstat1 == 1) .and. (lstat2 == 1)) then
    wt1 = (agrmetdrv%agrmtime2 - lis%t%time) / &
    (agrmetdrv%agrmtime2  - agrmetdrv%agrmtime1)
    wt2 = 1.0 - wt1
    do c=1,gdi(iam)
       if ( (oblwdata1(c) > 0.0) .AND. &
            (oblwdata2(c) > 0.0)) then
          grid(c)%forcing(4)=  oblwdata1(c)*wt1 &
               + oblwdata2(c)*wt2
       end if
    end do
 end if
 deallocate(obsw)
 return
```

1.70.1 readtemplatecrd.F90 (Source File: readtemplatecrd.F90)

Routine to read Template specific parameters from the card file.

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readtemplatecrd(templatedrv)
```

USES:

```
use templatedrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=template)
print*,'Running TEMPLATE LSM:'

close(11)
```

1.70.2 template_binout.F90 (Source File: template_binout.F90)

LIS data writer: Writes template output in binary format

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine template_binout(ftn)
```

USES:

```
use lisdrv_module, only : lis
use drv_output_mod, only : drv_writevar_bin
use template_varder

implicit none
real :: rainf(lis%d%glbnch)
real :: snowf(lis%d%glbnch)
integer :: ftn
```

CONTENTS:

```
do t=1,lis%d%glbnch
   if(template(t)%forcing(1) < 273.15) then
      rainf(t) = 0.0
      snowf(t) = template(t)%forcing(8)
   else
      rainf(t) = template(t)%forcing(8)
      snowf(t) = 0.0
   endif
enddo
 if(lis%o%wfor.eq.1) then
    call drv_writevar_bin(ftn, sqrt(template%forcing(5)*template%forcing(5)+ &
         template%forcing(6)*template%forcing(6)))
    call drv_writevar_bin(ftn,rainf)
    call drv_writevar_bin(ftn,snowf)
    call drv_writevar_bin(ftn,template%forcing(1))
    call drv_writevar_bin(ftn,template%forcing(2))
    call drv_writevar_bin(ftn,template%forcing(7))
    call drv_writevar_bin(ftn,template%forcing(3))
    call drv_writevar_bin(ftn,template%forcing(4))
 endif
```

1.71 Fortran: Module Interface templatedrv_module.F90 (Source File: templatedrv_module.F90)

Template Module for runtime specific variables for a lsm

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
module templatedrv_module
```

ARGUMENTS:

```
type templatedrvdec
   integer :: templateopen       !Keeps track of opening files
   integer :: numout       !Counts number of output times for Template
   real :: writeint        !TEMPLATE Output Interval (hours)
end type templatedrvdec
```

1.71.1 template_dynsetup.F90 (Source File: template_dynsetup.F90)

Updates the time dependent variables for the template

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine template_dynsetup()
```

USES:

CONTENTS:

1.71.2 template_f2t.F90 (Source File: template_f2t.F90)

template for calling the forcing transfer routines.

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine template_f2t(t, forcing)
```

USES:

```
use template_varder
use lisdrv_module, only : lis
```

CONTENTS:

```
do f=1,lis%f%nforce
   template(t)%forcing(f)=forcing(f)
enddo
```

1.71.3 template_main.F90 (Source File: template_main.F90)

calls the run routines for a template

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine template_main()
```

USES:

CONTENTS:

1.72 Fortran: Module Interface template_module.F90 (Source File: template_module.F90)

template for 1-D land surface model variables

REVISION HISTORY:

```
21 Jul  2004: Sujay Kumar; nitial Specification
```

INTERFACE:

```
module template_module
```

CONTENTS:

```
  type templatedec
     integer :: count
     real :: forcing(10)
  end type templatedec
```

1.72.1 template_almaout.F90 (Source File: template_out.F90)

LIS TEMPLATE data writer: Binary and stat files in ALMA convention

REVISION HISTORY:

```
   21 Jul 2004: Sujay Kumar, Initial Specification
```

INTERFACE:

```
subroutine template_out
```

USES:

```
   use lisdrv_module, only : lis, gindex, tile
   use template_varder      ! TEMPLATE-specific variables

   implicit none
```

ARGUMENTS:

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
  if(mod(lis%t%gmt,templatedrv%writeint).eq.0)then
     templatedrv%numout=templatedrv%numout+1
     write(unit=temp1,fmt='(i4,i2,i2)')lis%t%yr,lis%t%mo,lis%t%da
     read(unit=temp1,fmt='(8a1)') ftime
     do i=1,8
        if(ftime(i).eq.(' '))ftime(i)='0'
     enddo
     write(unit=temp1,fmt='(i4)')lis%t%yr
     read(unit=temp1,fmt='(8a1)')ftimec
     do i=1,4
        if(ftimec(i).eq.(' '))ftimec(i)='0'
     enddo
```

```
     write(unit=temp1,fmt='(a40)') lis%o%odir
     read(unit=temp1,fmt='(40a1)') (fbase(i),i=1,40)
     c=0
     do i=1,40
        if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
     enddo

     write(unit=temp1,fmt='(a4,i3,a7,i4,a1,i4,i2,i2)')'/EXP', &
          lis%o%expcode,'/FVDAS/', &
          lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
     read(unit=temp1,fmt='(80a1)') (fyrmodir(i),i=1,27)
     do i=1,27
        if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
     enddo

     write(unit=temp1,fmt='(a9)')'mkdir -p '
     read(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9)

     write(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
          (fyrmodir(i),i=1,27)
     read(unit=temp1,fmt='(a80)')mkfyrmo
     call system(mkfyrmo)
!-----------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------
     write(unit=fbinname, fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
          lis%t%da,lis%t%hr
     read(unit=fbinname,fmt='(10a1)') ftimeb
     do i=1,10
        if(ftimeb(i).eq.(' '))ftimeb(i)='0'
     enddo
     if(lis%o%wout.eq.1) then
        write(unit=fbinname,fmt='(a5)') '.gs4r'
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     elseif(lis%o%wout.eq.2) then
        write(unit=fbinname,fmt='(a5)') '.grb '
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     elseif(lis%o%wout.eq.3) then
        write(unit=fbinname,fmt='(a5)') '.nc  '
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     endif
        write(unit=fbinname,fmt='(69a1)')(fbase(i),i=1,c), &
             (fyrmodir(i),i=1,27),'/',&
             (ftimeb(i),i=1,10), &
             (fsubgb(i),i=1,5)
        read(unit=fbinname,fmt='(a69)')filengb
!-----------------------------------------------------------------------
! Open statistical output file
```

```
!-----------------------------------------------------------------------
     if(templatedrv%templateopen.eq.0)then
        file='FvDASstats.dat'
        call openfile(name,lis%o%odir,lis%o%expcode,file)
        if(lis%o%startcode.eq.1)then
           open(65,file=name,form='formatted',status='unknown', &
                position='append')
        else
           open(65,file=name,form='formatted',status='replace')
        endif
        templatedrv%templateopen=1
     endif

       write(65,996)'       Statistical Summary of Template output for:  ', &
            lis%t%mo,'/',lis%t%da,'/',lis%t%yr,lis%t%hr,':',lis%t%mn,':',lis%t%ss
996    format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
       write(65,*)
       write(65,997)
997    format(t27,'Mean',t41,'Stdev',t56,'Min',t70,'Max')
       ftn = 58
       if(lis%o%wout.eq.1) then
          open(ftn,file=filengb,form='unformatted')
          call template_binout(ftn)
          close(58)
       endif
       call template_writestats(lis,65)
       template%count=0  !reset counters
       write(65,*)
       write(65,*)
    endif
```

1.72.2 template_output.F90 (Source File: template_output.F90)

Template for calling the output routines.

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine template_output()
```

USES:

```
use lisdrv_module, only : lis, tile, glbgindex
use spmdMod, only : masterproc,npes
use template_varder, only : templatedrv
```

CONTENTS:

```
if(mod(lis%t%gmt, templatedrv%writeint).eq.0)then
   if(masterproc) then
      call template_out()
   endif
endif
```

1.72.3 templaterst.F90 (Source File: templaterst.F90)

calls the restart reading routines for a template

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine templaterst()
```

USES:

CONTENTS:

1.72.4 template_setup.F90 (Source File: template_setup.F90)

Complete the setup routines for template

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar: Initial Specification
```

INTERFACE:

```
subroutine template_setup()

end subroutine template_setup
\markboth{Left}{Source File: template\_varder.F90,  Date: Fri Feb 11 15:41:05 EST 2005
}
```

```
   NASA Goddard Space Flight Center Land Information System (LIS) V3.0
   Released May 2004
   See SOFTWARE DISTRIBUTION POLICY for software distribution policies
   The LIS source code and documentation are in the public domain,
   available without fee for educational, research, non-commercial and
   commercial purposes.  Users may distribute the binary or source
   code to third parties provided this statement appears on all copies and
   that no charge is made for such copies.
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See COPYRIGHT.TXT for copyright details.
  ---------------------------------------------------------------------------
 #include <misc.h>
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\


  \subsubsection{template\_varder.F90 (Source File: template\_varder.F90)}



    Module for 1-D land model driver variable initialization

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}   21 July 2004; Sujay Kumar, Initial Code
```

INTERFACE:

```
 module template_varder
```

USES:

```
   use tile_spmdMod
   use template_module
   use templatedrv_module
```

1.72.5 template_varder_ini (Source File: template_varder.F90)

Reads in runtime template parameters, allocates memory for variables

INTERFACE:

```
   subroutine template_varder_ini(nch)
```

USES:

```
     integer :: nch
%/////////////////////////////////////////////////////////////
{\sf CONTENTS:}
 \begin{verbatim}
    if(masterproc) then
       call readtemplatecrd(templatedrv)
    endif

    if(masterproc) then
       allocate(template(nch))
    else
       allocate(template(di_array(iam)))
    endif
  end subroutine template_varder_ini
```

1.72.6 template_writerst.F90 (Source File: template_writerst.F90)

Template for calling the write restart routines.

REVISION HISTORY:

```
21 Jul 2004: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine template_writerst()
```

USES:

CONTENTS:

1.72.7 template_writevars.F90 (Source File: template_writestats.F90)

LIS data writer: Writes template output

REVISION HISTORY:

```
21 Jul 2004; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine template_writestats(ld,ftn_stats)
```

USES:

```
use lisdrv_module, only : gindex
use lis_module
use template_varder

implicit none

type(lisdec) :: ld
integer :: ftn_stats,t
```

CONTENTS:

```
do t=1,ld%d%glbnch
   if(template(t)%forcing(1) < 273.15) then
      rainf(t) = 0.0
      snowf(t) = template(t)%forcing(8)
   else
      rainf(t) = template(t)%forcing(8)
      snowf(t) = 0.0
   endif
enddo
 if(ld%o%wfor.eq.1) then
    call stats(sqrt(template%forcing(5)*template%forcing(5)+ &
         template%forcing(6)*template%forcing(6)), &
         ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
    write(ftn_stats,999) 'Wind(m/s)', &
         vmean,vstdev,vmin,vmax
    call stats(rainf, &
         ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
    write(ftn_stats,998) 'Rainf(kg/m2s)', &
         vmean,vstdev,vmin,vmax
    call stats(snowf, &
         ld%d%udef,ld%d%glbnch,vmean,vstdev,vmin, vmax)
    write(ftn_stats,998) 'Snowf(kg/m2s)', &
         vmean,vstdev,vmin,vmax
    call stats(template%forcing(1),ld%d%udef,ld%d%glbnch,vmean,vstdev, &
         vmin, vmax)
    write(ftn_stats,999) 'Tair(K)', &
         vmean,vstdev,vmin,vmax
    call stats(template%forcing(2),ld%d%udef,ld%d%glbnch,vmean,vstdev, &
         vmin, vmax)
    write(ftn_stats,999) 'Qair(kg/kg)', &
         vmean,vstdev,vmin,vmax
    call stats(template%forcing(7),ld%d%udef,ld%d%glbnch,vmean,vstdev, &
         vmin, vmax)
    write(ftn_stats,999) 'PSurf(Pa)', &
         vmean,vstdev,vmin,vmax
    call stats(template%forcing(3),ld%d%udef,ld%d%glbnch,vmean,vstdev, &
```

```
            vmin, vmax)
      write(ftn_stats,999) 'SWdown (W/m2)', &
            vmean,vstdev,vmin,vmax
      call stats(template%forcing(4),ld%d%udef,ld%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'LWdown(W/m2)', &
            vmean,vstdev,vmin,vmax
   endif
998    FORMAT(1X,A18,4E14.3)
999    FORMAT(1X,A18,4F14.3)
```

1.72.8 mapvegc.F90 (Source File: mapvegc.F90)

This subroutine converts the UMD classes to the SIB classes used by NOAH LSM (v 2.5). (Originally from Dag Lohmann at NCEP)

REVISION HISTORY:

```
28 Apr 2002, K Arsenault:  Added NOAH LSM to LDAS.
```

INTERFACE:

```
subroutine mapvegc(vegt)
```

CONTENTS:

```
!-----------------------------------------------------------------
!  Convert UMD Classes to SIB Classes.
!-----------------------------------------------------------------
  if (vegt .eq. 1)  sibveg = 4
  if (vegt .eq. 2)  sibveg = 1
  if (vegt .eq. 3)  sibveg = 5
  if (vegt .eq. 4)  sibveg = 2
  if (vegt .eq. 5)  sibveg = 3
  if (vegt .eq. 6)  sibveg = 3
  if (vegt .eq. 7)  sibveg = 6
  if (vegt .eq. 8)  sibveg = 8
  if (vegt .eq. 9)  sibveg = 9
  if (vegt .eq. 10) sibveg = 7
  if (vegt .eq. 11) sibveg = 12
  if (vegt .eq. 12) sibveg = 11
  if (vegt .eq. 13) sibveg = 11
  if (vegt .gt. 13) then
     sibveg = 7
  end if
  vegt=sibveg
  return
```

1.72.9 noah_alb.F90: (Source File: noah_alb.F90)

This subroutine takes quarterly surface albedo (snow-free) data and day to interpolate and determine the actual value of the albedo for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the dates of January 31, April 30, July 31, and October 31.

REVISION HISTORY:

```
    28 Apr 2002: K. Arsenault; Added NOAH LSM to LDAS, initial code
```

INTERFACE:

```
 subroutine noah_alb
```

USES:

```
   use time_manager
   use noah_varder      ! NOAH tile variables
   use time_manager
   use lisdrv_module, only : grid,tile,lis
   use lis_openfileMod
   use lis_indices_module
 #if ( defined OPENDAP )
   use opendap_module
 #endif
   implicit none
```

CONTENTS:

```
  zeroi=0
  noahdrv%noah_aflag = 0
!-----------------------------------------------------------------------------
! Determine Dates of the quarters in terms of Year (e.g., 1999.3)
!-----------------------------------------------------------------------------
  time=lis%t%time
  yr=lis%t%yr
!-----------------------------------------------------------------------------
!  January 31
!-----------------------------------------------------------------------------
  janda=31
  janmo=01
  call date2time(jan31,doy1,gmt1,yr,janmo,&
       janda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------------
!  April 30
!-----------------------------------------------------------------------------
  aprda=30
  aprmo=04
```

```
  call date2time(apr30,doy1,gmt1,yr,aprmo,&
       aprda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  July 31
!-----------------------------------------------------------------------
  julda=31
  julmo=07
  call date2time(jul31,doy1,gmt1,yr,julmo,&
       julda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  October 31
!-----------------------------------------------------------------------
  octda=31
  octmo=10
  call date2time(oct31,doy1,gmt1,yr,octmo,&
       octda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
! Determine which two quarterly albedo files book-end model time.
!-----------------------------------------------------------------------

  if ( time.ge.jan31 .and. time.le.apr30 ) then
     qq1="01"
     qq2="02"
     qdif = apr30-jan31
     timdif = time-jan31
     albflag = 1
  elseif ( time.ge.apr30 .and. time.le.jul31 ) then
     qq1="02"
     qq2="03"
     qdif = jul31-apr30
     timdif = time-apr30
     albflag = 2
  elseif ( time.ge.jul31 .and. time.le.oct31 ) then
     qq1="03"
     qq2="04"
     qdif = oct31-jul31
     timdif = time-jul31
     albflag = 3
  elseif ( time.ge.oct31 ) then
     qq1="04"
     qq2="01"
     qdif = (jan31+1.0)-oct31
     timdif = time-oct31
     albflag = 4
  elseif ( time.lt.jan31) then
     qq1="04"
     qq2="01"
     oct31=oct31-1.0
```

```
     qdif = jan31-oct31
     timdif = time-oct31
     albflag = 5
  endif

  if(noahdrv%noah_albtime .ne. albflag) then
     noahdrv%noah_albtime = albflag
     noahdrv%noah_aflag = 1
!-----------------------------------------------------------------------
!  Open the needed two quarterly snow-free albedo files
!-----------------------------------------------------------------------
!     if(lis%d%gridDesc(9) .eq. 0.01) then
        file1 = trim(noahdrv%noah_albfile)//'alb_'//QQ1//'.1gd4r'
        call lis_set_filename(file1,time_offset=qq1)
        call lis_open_file(10, &
             file=file1,status='old',form='unformatted',recl=4, &
             access='direct',script='getalbedo.pl', time_offset=qq1)
        file2 = trim(noahdrv%noah_albfile)//'alb_'//QQ2//'.1gd4r'
        call lis_set_filename(file2,time_offset=qq2)
        call lis_open_file (11, &
             file=file2, status='old',form='unformatted',recl=4, &
             access='direct',script='getalbedo.pl',time_offset=qq2)
!     else
!        file1 = trim(noahdrv%noah_albfile)//'albedo_'//QQ1//'.bfsa'
!        call lis_set_filename(file1, time_offset=qq1)
!        call lis_open_file(10, &
!             file=file1,status='old',form='unformatted',&
!             script='getalbedo.pl', time_offset=qq1)
!        file2=trim(noahdrv%noah_albfile)//'albedo_'//QQ2//'.bfsa'
!        call lis_set_filename(file2, time_offset=qq2)
!        call lis_open_file (11, &
!             file=file2, status='old',form='unformatted',&
!             script='getalbedo.pl', time_offset=qq2)
!     endif
     call lis_read_file(10,value1)
     call lis_read_file(11,value2)
     close(10)
     close(11)

!-----------------------------------------------------------------------
! Assign quarterly albedo fractions to each tile.
!-----------------------------------------------------------------------
     do i=1,lis%d%nch
        if((value1(tile(i)%col, tile(i)%row-lis_tnroffset).ne.-9999.000) &
             .and. (value2(tile(i)%col,tile(i)%row-lis_tnroffset)&
             .ne.-9999.000)) then
           noah(i)%albsf1= value1(tile(i)%col, tile(i)%row-lis_tnroffset)
           noah(i)%albsf2= value2(tile(i)%col, tile(i)%row-lis_tnroffset)
```

```
        endif
     enddo
  endif    ! End albflag selection

!-----------------------------------------------------------------------
! Assign albedo fractions to each tile and interpolate daily.
!-----------------------------------------------------------------------
  if (noahdrv%noah_albdchk .ne. lis%t%da) then
     noahdrv%noah_aflag = 1
     do i=1,lis%d%nch
        if (noah(i)%albsf1 .ne. -9999.000) then
           valdif(i) = noah(i)%albsf2 - noah(i)%albsf1
           noah(i)%albsf = (timdif*valdif(i)/qdif)+noah(i)%albsf1
        endif
     end do
     noahdrv%noah_albdchk=lis%t%da

     if(lis%o%wparam.eq.1) then
        allocate(albout(lis%d%lnc,lis%d%lnr))
        albout = -9999.0
        if(lis%d%gridDesc(9) .ne. 0.01 ) then
           do i=1,lis%d%nch
              if(grid(i)%lat.ge.lis%d%gridDesc(4).and. &
                   grid(i)%lat.le.lis%d%gridDesc(7).and. &
                   grid(i)%lon.ge.lis%d%gridDesc(5).and. &
                   grid(i)%lon.le.lis%d%gridDesc(8)) then
                 rindex = tile(i)%row - nint((lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
                      /lis%d%gridDesc(9))
                 cindex = tile(i)%col - nint((lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
                      /lis%d%gridDesc(10))
                 albout(cindex,rindex) = noah(i)%albsf
              endif
           enddo
        elseif(lis%d%gridDesc(9) .eq.0.01) then
           do i=1,lis%d%nch
              albout(tile(i)%col,tile(i)%row) = noah(i)%albsf
           enddo
        endif

        open(32,file="albout.bin",form='unformatted')
        write(32) albout
        close(32)
        deallocate(albout)
     end if
  endif ! End daily interpolation
  return
```

1.72.10 noah_almaout.F90 (Source File: noah_almaout.F90)

LIS NOAH data writer: Binary and stat files in ALMA convention

REVISION HISTORY:

```
 4 Nov. 1999: Jon Radakovich; Initial Code
28 Apr. 2002: Kristi Arsenault; Added NOAH LSM to LDAS
15 Jun  2003: Sujay Kumar; ALMA version
```

INTERFACE:

```
subroutine noah_almaout
```

USES:

```
#if ( defined USE_NETCDF )
  use netcdf
#endif
  use lisdrv_module, only : lis, gindex, tile
  use grid_module      ! LIS non-model-specific grid variables
  use noah_varder      ! NOAH-specific variables

  implicit none
```

ARGUMENTS:

```
  logical*1 :: lismask(lis%d%lnc,lis%d%lnr)
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
  if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
     noahdrv%numout=noahdrv%numout+1
     write(unit=temp1,fmt='(i4,i2,i2)')lis%t%yr,lis%t%mo,lis%t%da
     read(unit=temp1,fmt='(8a1)') ftime
     do i=1,8
        if(ftime(i).eq.(' '))ftime(i)='0'
     enddo
     write(unit=temp1,fmt='(i4)')lis%t%yr
     read(unit=temp1,fmt='(8a1)')ftimec
     do i=1,4
        if(ftimec(i).eq.(' '))ftimec(i)='0'
     enddo
!     write(unit=temp1,fmt='(a6,i3,a1)') '/LIS.E',lis%o%expcode,'.'
!     read(unit=temp1,fmt='(80a1)') (fname(i),i=1,10)
!     do i=1,10
```

```
!        if(fname(i).eq.(' '))fname(i)='0'
!     enddo
     write(unit=temp1,fmt='(a40)') lis%o%odir
     read(unit=temp1,fmt='(40a1)') (fbase(i),i=1,40)
     c=0
     do i=1,40
        if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
     enddo

     write(unit=temp1,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)')'/EXP', &
          lis%o%expcode,'/NOAH/', &
          lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
     read(unit=temp1,fmt='(80a1)') (fyrmodir(i),i=1,26)
     do i=1,26
        if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
     enddo

     write(unit=temp1,fmt='(a9)')'mkdir -p '
     read(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9)

     write(unit=temp1,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
          (fyrmodir(i),i=1,26)
     read(unit=temp1,fmt='(a80)')mkfyrmo
     call system(mkfyrmo)
!-----------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------
     write(unit=fbinname, fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
          lis%t%da,lis%t%hr
     read(unit=fbinname,fmt='(10a1)') ftimeb
     do i=1,10
        if(ftimeb(i).eq.(' '))ftimeb(i)='0'
     enddo
     if(lis%o%wout.eq.1) then
        write(unit=fbinname,fmt='(a5)') '.gs4r'
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     elseif(lis%o%wout.eq.2) then
        write(unit=fbinname,fmt='(a5)') '.grb '
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     elseif(lis%o%wout.eq.3) then
        write(unit=fbinname,fmt='(a5)') '.nc  '
        read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
     endif
        write(unit=fbinname,fmt='(69a1)')(fbase(i),i=1,c), &
             (fyrmodir(i),i=1,26),'/',&
             (ftimeb(i),i=1,10), &
             (fsubgb(i),i=1,5)
        read(unit=fbinname,fmt='(a69)')filengb
```

```
!-----------------------------------------------------------------------
! Open statistical output file
!-----------------------------------------------------------------------
     if(noahdrv%noahopen.eq.0)then
        file='Noahstats.dat'
        call openfile(name,lis%o%odir,lis%o%expcode,file)
        if(lis%o%startcode.eq.1)then
           open(65,file=name,form='formatted',status='unknown', &
                position='append')
        else
           open(65,file=name,form='formatted',status='replace')
        endif
        noahdrv%noahopen=1
     endif

       write(65,996)'       Statistical Summary of Noah output for:  ', &
            lis%t%mo,'/',lis%t%da,'/',lis%t%yr,lis%t%hr,':',lis%t%mn,':',lis%t%ss
996    format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
       write(65,*)
       write(65,997)
997    format(t27,'Mean',t41,'Stdev',t56,'Min',t70,'Max')
       ftn = 58

!-----------------------------------------------------------------------
! Write Binary Output
!-----------------------------------------------------------------------
       if(lis%o%wout.eq.1) then
          open(ftn,file=filengb,form='unformatted')
          call noah_binout(ftn)
          close(58)
!-----------------------------------------------------------------------
! Write Grib Output
!-----------------------------------------------------------------------
       elseif(lis%o%wout.eq.2) then
          call baopen (ftn,filengb, iret)
          call noah_gribout(ftn)
          call baclose(ftn,iret)
       elseif(lis%o%wout.eq.3) then !netcdf
#if ( defined USE_NETCDF )
!-----------------------------------------------------------------------
! Write NetCDF Output
!-----------------------------------------------------------------------
          iret = nf90_create(path=trim(filengb),cmode=nf90_clobber,ncid=ftn)
          call noah_netcdfout(ftn)
          iret = nf90_close(ftn)
#endif
       endif
       call noah_writestats(65)
```

```
      noah%count=0  !reset counters
      write(65,*)
      write(65,*)
   endif
```

1.72.11 noah_atmdrv.F90 (Source File: noah_atmdrv.F90)

Transfer forcing from grid to tile space.

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial Code
28 Jan 2002: Jon Gottschalck; Added option for different number of forcing variables
```

INTERFACE:

```
subroutine noah_f2t(t, forcing)
```

USES:

```
use lisdrv_module , only : lis
use spmdMod
use tile_spmdMod
use noah_varder
```

CONTENTS:

```
do f=1,lis%f%nforce
   noah(t)%forcing(f)=forcing(f)
enddo
```

1.72.12 noah_binout.F90 (Source File: noah_binout.F90)

LIS NOAH data writer: Writes noah output in binary format

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine noah_binout(ftn)
```

USES:

```
   use lisdrv_module, only : lis
   use drv_output_mod, only : drv_writevar_bin
   use noah_varder

   implicit none
   integer :: ftn
```

CONTENTS:

```
  do t=1,lis%d%glbnch
     if(noah(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = noah(t)%forcing(8)
     else
        rainf(t) = noah(t)%forcing(8)
        snowf(t) = 0.0
     endif
  enddo
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
   noah%swnet = noah%swnet/float(noah%count)
   call drv_writevar_bin(ftn,noah%swnet) !Net shortwave radiation (surface) (W/m2)
   noah%lwnet = (-1)*noah%lwnet/float(noah%count)
   call drv_writevar_bin(ftn,noah%lwnet)!Net longwave radiation (surface) (W/m2)
   noah%qle = noah%qle/float(noah%count)
   call drv_writevar_bin(ftn,noah%qle) !Latent Heat Flux (W/m2)
   noah%qh = noah%qh/float(noah%count)
   call drv_writevar_bin(ftn,noah%qh) !Sensible Heat Flux (W/m2)
   noah%qg = noah%qg/float(noah%count)
   call drv_writevar_bin(ftn,noah%qg)
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   noah%snowf = noah%snowf/float(noah%count)
   call drv_writevar_bin(ftn,noah%snowf)
   noah%rainf = noah%rainf/float(noah%count)
   call drv_writevar_bin(ftn,noah%rainf)
   noah%evap = noah%evap/float(noah%count)
   call drv_writevar_bin(ftn,noah%evap)
   noah%qs = noah%qs/float(noah%count)
   call drv_writevar_bin(ftn,noah%qs)
   noah%qsb = noah%qsb/float(noah%count)
   call drv_writevar_bin(ftn,noah%qsb)
   noah%qsm = noah%qsm/float(noah%count)
   call drv_writevar_bin(ftn,noah%qsm)
   call drv_writevar_bin(ftn,(noah%smc(1)*1000.0*0.1+ &
        noah%smc(2)*1000.0*0.3 + &
        noah%smc(3)*1000.0*0.6 + &
```

```
        noah%smc(4)*1000.0*1.0 -noah%soilm_prev)/float(noah%count))
   call drv_writevar_bin(ftn,(noah%sneqv*1000.0-noah%swe_prev)/float(noah%count))
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call drv_writevar_bin(ftn,noah%avgsurft)
   call drv_writevar_bin(ftn,noah%albedo)
   call drv_writevar_bin(ftn,noah%swe)
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   call drv_writevar_bin(ftn,noah%stc(1))
   call drv_writevar_bin(ftn,noah%stc(2))
   call drv_writevar_bin(ftn,noah%stc(3))
   call drv_writevar_bin(ftn,noah%stc(4))
   call drv_writevar_bin(ftn,noah%soilmoist1)
   call drv_writevar_bin(ftn,noah%soilmoist2)
   call drv_writevar_bin(ftn,noah%soilmoist3)
   call drv_writevar_bin(ftn,noah%soilmoist4)
   call drv_writevar_bin(ftn,noah%soilwet)
!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
   noah%ecanop = noah%ecanop/float(noah%count)
   call drv_writevar_bin(ftn, noah%ecanop)
   noah%tveg= noah%tveg/float(noah%count)
   call drv_writevar_bin(ftn,noah%tveg)
   noah%esoil= noah%esoil/float(noah%count)
   call drv_writevar_bin(ftn,noah%esoil)
   call drv_writevar_bin(ftn, noah%rootmoist)
   call drv_writevar_bin(ftn, noah%canopint)
!-----------------------------------------------------------------------------
! Forcing Components
!-----------------------------------------------------------------------------
   if(lis%o%wfor.eq.1) then
      call drv_writevar_bin(ftn, sqrt(noah%forcing(5)*noah%forcing(5)+ &
           noah%forcing(6)*noah%forcing(6)))
      call drv_writevar_bin(ftn,rainf)
      call drv_writevar_bin(ftn,snowf)
      call drv_writevar_bin(ftn,noah%forcing(1))
      call drv_writevar_bin(ftn,noah%forcing(2))
      call drv_writevar_bin(ftn,noah%forcing(7))
      call drv_writevar_bin(ftn,noah%forcing(3))
      call drv_writevar_bin(ftn,noah%forcing(4))
   endif
```

1.72.13 noah_coldstart.F90 (Source File: noah_coldstart.F90)

Routine for noah initialization from cold start

INTERFACE:

```
subroutine noah_coldstart()
```

USES:

```
use lisdrv_module, only: lis
use noah_varder
use time_manager
use spmdMod, only: iam
```

CONTENTS:

```
if ( lis%o%startcode == 2 ) then
   print*,'MSG: noah_coldstart -- cold-starting noah', &
          '...using ics from card file',' (', iam, ')'

   print*,'DBG: noah_coldstart -- nch',lis%d%nch, &
        ' (', iam, ')'
   do t=1,lis%d%nch
      noah(t)%t1=noahdrv%noah_it
      noah(t)%t1=280.0
      noah(t)%cmc=0.0004
      noah(t)%snowh=0.0
      noah(t)%sneqv=0.0
      noah(t)%ch=0.0150022404
      noah(t)%cm=0.0205970779
      do l=1,4
         noah(t)%stc(l)=noahdrv%noah_it
      enddo
      noah(t)%smc(1)=0.3252287
      noah(t)%smc(2)=0.3194746
      noah(t)%smc(3)=0.3172167
      noah(t)%smc(4)=0.3078052
      noah(t)%sh2o(1)=0.1660042
      noah(t)%sh2o(2)=0.2828006
      noah(t)%sh2o(3)=0.3172163
      noah(t)%sh2o(4)=0.3078025
   enddo
   lis%t%yr=lis%t%syr
   lis%t%mo=lis%t%smo
   lis%t%da=lis%t%sda
   lis%t%hr=lis%t%shr
   lis%t%mn=lis%t%smn
```

```
      lis%t%ss=lis%t%sss

      call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr,&
                     lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
      write(*,*)'MSG: noah_coldstart -- Using lis.crd start time ',&
                lis%t%time, ' (', iam, ')'
   endif
```

1.73 Fortran: Module Interface noahdrv_module.F90 (Source File: noahdrv_module.F90)

Module for runtime specific Noah variables

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module noahdrv_module
```

ARGUMENTS:

```
type noahdrvdec
   integer :: noahopen       !Keeps track of opening files
   integer :: numout       !Counts number of output times for Noah
   integer :: noah_nvegp     !Number of static vegetation parameter
   integer :: noah_nsoilp    !Number of static soil parameters
   integer :: noah_zst       !Number of Zobler soil classes
   integer :: noah_gflag     !Time flag to update gfrac files
   integer :: noah_albtime   !Time flag to update albedo files
   integer :: noah_aflag     !Time flag to update albedo files
   integer :: noah_albdchk   !Day check to interpolate alb values
   integer :: noah_gfracdchk !Day check to interpolate gfrac value
   integer :: varid(21)      !For netcdf output
   character*40 :: noah_rfile !NOAH Active Restart File
   character*40 :: noah_vfile !NOAH Static Vegetation Parameter File
   character*40 :: noah_sfile !NOAH Soil Parameter File
   character*40 :: noah_mgfile !NOAH Monthly Veg. Green Frac.
   character*40 :: noah_albfile !NOAH Quart. Snow-free albedo
   character*50 :: noah_mxsnal !NOAH GLDAS max snow albedo
   character*50 :: noah_tbot   !NOAH GLDAS Bottom Temp
   real*8 :: noah_gfractime  !Time flag to update gfrac files
   real :: noah_ism          !NOAH Initial Soil Moisture (m3/m3)
   real :: noah_it           !NOAH Initial Soil Temperature (K)
   real :: writeintn         !NOAH Output Interval (hours)
end type noahdrvdec
```

1.73.1 noah_dynsetup.F90 (Source File: noah_dynsetup.F90)

Updates the time dependent NOAH variables

REVISION HISTORY:

```
15 Apr 2002: Sujay Kumar   Initial Specification
```

INTERFACE:

```
subroutine noah_dynsetup()
```

USES:

```
use lisdrv_module, only: lis,tile
use noah_varder
use spmdMod, only : masterproc, npes
use noahpardef_module
```

CONTENTS:

```
  integer :: t, n,ier
#if ( ! defined OPENDAP )
  if ( npes > 1 ) then
     call noah_gather
  endif
  if(masterproc) then
#endif
     call noah_gfrac()
     call noah_alb()
#if ( ! defined OPENDAP )
  endif
#if (defined SPMD)
  call MPI_BCAST(noahdrv%noah_gflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  call MPI_BCAST(noahdrv%noah_aflag,1,MPI_INTEGER,0, &
       MPI_COMM_WORLD,ier)
  if( npes > 1 .and. ( noahdrv%noah_gflag==1 .or. &
       noahdrv%noah_aflag ==1 ) ) then
     call noah_scatter
  endif
#endif
#endif
```

1.73.2 noah_gather.F90 (Source File: noah_gather.F90)

Gathers noah tiles

REVISION HISTORY:

```
    Apr 2003 ; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine noah_gather()
```

USES:

```
   use tile_spmdMod
   use noah_varder
   use noahpardef_module
```

CONTENTS:

```
#if  (defined SPMD)
  call MPI_GATHERV(noah(1:di_array(iam)),di_array(iam), &
       MPI_NOAH_STRUCT,noah,di_array,displs,MPI_NOAH_STRUCT, &
       0,MPI_COMM_WORLD, ierr)
#endif
```

1.73.3 noah_gfrac.F90 (Source File: noah_gfrac.F90)

This subroutine takes vegetation greenness fraction data and the date to interpolate and determine the actual value of the greenness fraction for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the 16th of the given month, at 00Z.

REVISION HISTORY:

```
    28 Apr 2002: K. Arsenault; Added NOAH LSM to LDAS, initial code
```

INTERFACE:

```
 subroutine noah_gfrac
```

USES:

```
   use noah_varder
   use time_manager
   use lisdrv_module, only : grid,tile,lis
   use lis_openfileMod
   use lis_indices_module
```

CONTENTS:

```
  noahdrv%noah_gflag = 0
  zeroi=0
  numi=16
!-----------------------------------------------------------------------
! Determine Monthly data Times (Assume Monthly value valid at DA=16)
```

```
!-----------------------------------------------------------------------
  if(lis%t%da.lt.16)then
     mo1=lis%t%mo-1
     yr1=lis%t%yr
     if(mo1.eq.0)then
        mo1=12
        yr1=lis%t%yr-1
     endif
     mo2=lis%t%mo
     yr2=lis%t%yr
  else
     mo1=lis%t%mo
     yr1=lis%t%yr
     mo2=lis%t%mo+1
     yr2=lis%t%yr
     if(mo2.eq.13)then
        mo2=1
        yr2=lis%t%yr+1
     endif
  endif

  call date2time(time1,doy1,gmt1,yr1,mo1,&
       numi,zeroi,zeroi,zeroi)
  call date2time(time2,doy2,gmt2,yr2,mo2,&
       numi,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  Weights to be used to interpolate greenness fraction values.
!-----------------------------------------------------------------------
  wt1= (time2-lis%t%time)/(time2-time1)
  wt2= (lis%t%time-time1)/(time2-time1)
!-----------------------------------------------------------------------
!  Determine if GFRAC files need to be updated
!-----------------------------------------------------------------------
  if(time2 .gt. noahdrv%noah_gfractime) then
     gfrac_flag = 1
  else
     gfrac_flag = 0
  endif

  if(gfrac_flag .eq. 1) then
     noahdrv%noah_gfractime = time2
     noahdrv%noah_gflag = 1
!-----------------------------------------------------------------------
! Open greenness fraction dataset of months corresponding to
! time1 and time2 for selected LDAS domain and read data.
!-----------------------------------------------------------------------
     write(mm1,3) mo1
     write(mm2,3) mo2
```

```
3    format(i2.2)

!     if(lis%d%gridDesc(9) .eq. 0.01 ) then
        file1 = trim(noahdrv%noah_mgfile)//'gfrac_'//mm1//'.1gd4r'
        call lis_set_filename(file1,time_offset=mm1)
        call lis_open_file (10, &
             file=file1,status='old', form='unformatted',&
                  access='direct',recl=4, script='getgfrac.pl',time_offset=mm1)
        file2 = trim(noahdrv%noah_mgfile)//'gfrac_'//mm2//'.1gd4r'
        call lis_set_filename(file2,time_offset=mm2)
        call lis_open_file (11, &
             file=file2,status='old', form='unformatted',recl=4, &
             access='direct',script='getgfrac.pl',time_offset=mm2)
!     else
!        file1 = trim(noahdrv%noah_mgfile)//'gfrac_'//mm1//'.bfsa'
!        call lis_set_filename(file1,time_offset=mm1)
!        call lis_open_file (10, &
!             file=file1, status='old', form='unformatted',&
!             script='getgfrac.pl',time_offset=mm1)
!        file2 = trim(noahdrv%noah_mgfile)//'gfrac_'//mm2//'.bfsa'
!        call lis_set_filename(file2,time_offset=mm2)
!        call lis_open_file (11,file=file2,status='old', form='unformatted',&
!             script='getgfrac.pl',time_offset=mm2)
!     endif

     call lis_read_file(10,value1)
     call lis_read_file(11,value2)
     close(10)
     close(11)

!-----------------------------------------------------------------------
! Assign MONTHLY vegetation greenness fractions to each tile.
!-----------------------------------------------------------------------
     do i=1,lis%d%nch
        if((value1(tile(i)%col, tile(i)%row-lis_tnroffset) .ne. -9999.000) &
         .and.(value2(tile(i)%col, tile(i)%row-lis_tnroffset).ne.-9999.000)) &
         then
           noah(i)%vegmp1=value1(tile(i)%col, tile(i)%row-lis_tnroffset)
           noah(i)%vegmp2=value2(tile(i)%col, tile(i)%row-lis_tnroffset)
        endif
     end do
  endif

!-----------------------------------------------------------------------
!  Interpolate greenness fraction values once daily
!-----------------------------------------------------------------------

  if (noahdrv%noah_gfracdchk .ne. lis%t%da) then
```

```
    noahdrv%noah_gflag = 1
    do i=1,lis%d%nch
       noah(i)%vegip = (wt1*noah(i)%vegmp1)+(wt2*noah(i)%vegmp2)
    end do

    noahdrv%noah_gfracdchk = lis%t%da
    print*, 'Done noah_gfrac',' (',iam,')'

    if(lis%o%wparam.eq.1) then
       allocate(gfracout(lis%d%lnc,lis%d%lnr))
       gfracout = -9999.0
       if(lis%d%gridDesc(9) .ne. 0.01) then
          do i=1,lis%d%nch
             if(grid(i)%lat.ge.lis%d%gridDesc(4).and. &
                  grid(i)%lat.le.lis%d%gridDesc(7).and. &
                  grid(i)%lon.ge.lis%d%gridDesc(5).and. &
                  grid(i)%lon.le.lis%d%gridDesc(8)) then
                rindex = tile(i)%row - nint((lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
                     /lis%d%gridDesc(9))
                cindex = tile(i)%col - nint((lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
                     /lis%d%gridDesc(10))
                gfracout(cindex,rindex) = noah(i)%vegip*1.0
             endif
          enddo
       elseif(lis%d%gridDesc(9) .eq.0.01) then
          do i=1,lis%d%nch
             gfracout(tile(i)%col,tile(i)%row) = noah(i)%vegip*1.0
          enddo
       endif

       open(32,file="gfracout.bin",form='unformatted')
       write(32) gfracout
       close(32)
       deallocate(gfracout)
    endif
  end if
  return
```

1.73.4 noah_binout.F90 (Source File: noah_gribout.F90)

LIS NOAH data writer: Writes noah output in binary format

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine noah_gribout(ftn)
```

USES:

```
use lisdrv_module, only : lis, gindex
use drv_output_mod, only : drv_writevar_grib
use noah_varder
use time_manager, only : tick

implicit none

integer :: ftn
```

CONTENTS:

```
interval = noahdrv%writeintn
hr1=lis%t%hr
da1=lis%t%da
mo1=lis%t%mo
yr1=lis%t%yr
mn1=lis%t%mn
ss1=0
ts1=-3600*24
dummygmt=1.0
dummytime=1.0
write(unit=temp1,fmt='(i4,i2,i2)')yr1,mo1,da1
read(unit=temp1,fmt='(8a1)')tod
do i=1,8
   if(tod(i).eq.(' '))tod(i)='0'
enddo
today=tod(1)//tod(2)//tod(3)//tod(4)//tod(5) &
     //tod(6)//tod(7)//tod(8)

call tick(dummytime,doy1,dummygmt,yr1,mo1,da1,hr1,mn1,ss1,ts1)
write(unit=temp1,fmt='(i4,i2,i2)')yr1,mo1,da1
read(unit=temp1,fmt='(8a1)')yes
do i=1,8
   if(yes(i).eq.(' '))yes(i)='0'
enddo
yesterday=yes(1)//yes(2)//yes(3)//yes(4)//yes(5) &
     //yes(6)//yes(7)//yes(8)
do i=1,25
   kpds(i)=0
enddo
kpds(1)=221               !id for gsfc products
kpds(2)=221               !id for noah model (change value for other models)
kpds(4)=192               !bms flag... don't worry about this.
kpds(12)=0                !assume output time minute always = 0
```

```
  kpds(13)=1                !forecast time unit (hours)
  kpds(17)=int((noahdrv%writeintn*3600.0)/lis%t%ts) !number of time steps in
  !averaged/accum variables
  kpds(18)=0                !grib version -- left as 0 in ncep products
  kpds(19)=1                !version number of kpds.tbl for lisas.
  kpds(20)=0                !none missing from averages/accumulations (always4)
  kpds(23)=221              !gsfc id#
  kpds(24)=0                !does not apply to lisas output
  kpds(25)=0

  open (unit = 69, file = './src/tables/KPDS_completenoah.tbl')
  do k = 1, 42
     read(69,*)
  end do

  do c=1,lis%d%lnc
     do r=1,lis%d%lnr
        if(gindex(c,r).gt.0) then
           lismask(c,r)=.true.
        else
           lismask(c,r)=.false.
        endif
     enddo
  enddo

  do t=1,lis%d%glbnch
     if(noah(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = noah(t)%forcing(8)
     else
        rainf(t) = noah(t)%forcing(8)
        snowf(t) = 0.0
     endif
  enddo
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
   call readkpds(69,kpds)
   noah%swnet = noah%swnet/float(noah%count)
   call drv_writevar_grib(ftn,noah%swnet,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%lwnet = (-1)*noah%lwnet/float(noah%count)
   call drv_writevar_grib(ftn,noah%lwnet,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%qle = noah%qle/float(noah%count)
   call drv_writevar_grib(ftn,noah%qle,kpds,lismask,interval,today,yesterday)
```

```
   call readkpds(69,kpds)
   noah%qh = noah%qh/float(noah%count)
   call drv_writevar_grib(ftn,noah%qh,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%qg = noah%qg/float(noah%count)
   call drv_writevar_grib(ftn,noah%qg,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   call readkpds(69,kpds)
   noah%snowf = noah%snowf/float(noah%count)
   call drv_writevar_grib(ftn,noah%snowf,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%rainf = noah%rainf/float(noah%count)
   call drv_writevar_grib(ftn,noah%rainf,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%evap = noah%evap/float(noah%count)
   call drv_writevar_grib(ftn,noah%evap,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%qs = noah%qs/float(noah%count)
   call drv_writevar_grib(ftn,noah%qs,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%qsb = noah%qsb/float(noah%count)
   call drv_writevar_grib(ftn,noah%qsb,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%qsm = noah%qsm/float(noah%count)
   call drv_writevar_grib(ftn,noah%qsm,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,(noah%smc(1)*1000.0*0.1+ &
        noah%smc(2)*1000.0*0.3 + &
        noah%smc(3)*1000.0*0.6 + &
        noah%smc(4)*1000.0*1.0 -noah%soilm_prev)/float(noah%count),kpds,lismask,interval,today

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,(noah%sneqv*1000.0-noah%swe_prev)/float(noah%count),kpds,lismask
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%avgsurft,kpds,lismask,interval,today,yesterday)
```

```
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%albedo,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%swe,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%stc(1),kpds,lismask,interval,today,yesterday)
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%stc(2),kpds,lismask,interval,today,yesterday)
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%stc(3),kpds,lismask,interval,today,yesterday)
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%stc(4),kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%soilmoist1,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%soilmoist2,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%soilmoist3,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%soilmoist4,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   call drv_writevar_grib(ftn,noah%soilwet,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
   call readkpds(69,kpds)
   noah%ecanop = noah%ecanop/float(noah%count)
   call drv_writevar_grib(ftn,noah%ecanop,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%tveg= noah%tveg/float(noah%count)
   call drv_writevar_grib(ftn,noah%tveg,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
   noah%esoil= noah%esoil/float(noah%count)
   call drv_writevar_grib(ftn,noah%esoil,kpds,lismask,interval,today,yesterday)

   call readkpds(69,kpds)
```

```
   call drv_writevar_grib(ftn, noah%rootmoist,kpds,lismask,interval,today,yesterday)
   call readkpds(69,kpds)
   call drv_writevar_grib(ftn, noah%canopint,kpds,lismask,interval,today,yesterday)

!-----------------------------------------------------------------------------
! Forcing Components
!-----------------------------------------------------------------------------
   if(lis%o%wfor.eq.1) then
      call readkpds(69,kpds)
      call drv_writevar_grib(ftn, sqrt(noah%forcing(5)*noah%forcing(5)+ &
           noah%forcing(6)*noah%forcing(6)),kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,rainf,kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,snowf,kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,noah%forcing(1),kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,noah%forcing(2),kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,noah%forcing(7),kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,noah%forcing(3),kpds,lismask,interval,today,yesterday)

      call readkpds(69,kpds)
      call drv_writevar_grib(ftn,noah%forcing(4),kpds,lismask,interval,today,yesterday)
   endif
   close(69)
```

1.73.5 noah_main.F90 (Source File: noah_main.F90)

NOAH LAND-SURFACE MODEL, UNCOUPLED 1-D COLUMN: VERSION 2.5 OCT 2001
THIS MAIN PROGRAM AND ITS FAMILY OF SUBROUTINES COMPRISE VERSION 2.5 OF THE PUBLIC RELEASE OF THE UNCOUPLED 1-D COLUMN VERSION OF THE "NOAH" LAND-SURFACE MODEL (LSM). THE NOAH LSM IS A DESCENDANT OF AN EARLIER GENERATION OF THE OREGON STATE UNIVERSITY (OSU) LSM, BUT IT INCLUDES SUBSTANTIAL PHYSICS EXTENSIONS AND RECODING ACCOMPLISHED ALONG THE WAY BY NCEP, HL (NWS), AFGWC,

AND AFGL/AFPL/AFRL. HENCE THE ACRONYM "NOAH" DENOTES N-NCEP, O-OSU, A-AIR FORCE, H-HYDRO LAB. ——————————————————————————————————————- FOR DOCUMENTATION OF THIS CODE AND INSTRUCTIONS ON ITS EXECUTION AND INPUT/OUTPUT FILES, SEE "NOAH LSM USERS GUIDE" IN FILE README_2.5 IN THE SAME PUBLIC SERVER DIRECTORY AS THIS SOURCE CODE. ——————————————————————————————————————- PROGRAM HISTORY LOG VERSION 1.0 – 01 MAR 1999
VERSION 1.1 – 08 MAR 1999
VERSION 2.0 – 27 JUL 1999
VERSION 2.1 – 23 OCT 2000
VERSION 2.2 – 07 FEB 2001
VERSION 2.3 – 07 MAY 2001 = operational Eta implementation
VERSION 2.4 – 27 JUN 2001 = ops Eta with NO physics changes
VERSION 2.5 – 18 OCT 2001
LDAS VERSION – 28 APR 2002 = NOAH Main added to LDAS Driver (NASA GSFC)
VERSION 2.5.1– 28 MAY 2002 = Updated changes in NOAH LSM along with correction to SOILRZ and SOIL1M.
VERSION 2.6 – 24 JUN 2003 = Updated to Noah LSM v2.6

Physics changes:
in SUBROUTINE SFLX change CSOIL from 1.26E+6 to 2.00E+6
in SUBROUTINE SFLX change ZBOT from -3.0 to -8.0
Replaced de-bugged SUBROUTINE TDFCND
Removed SUBROUTINE REDPRM and moved the parameters to other locations throughout noah_main and noah_physics subroutines.
VERSION 2.5.2 – 31 MAY 2002
Fix in FUNCTION DEVAP related to FX calculation
VERSION 2.6 – Includes changes to certain parameters and snow-soil physics

INTERFACE:

```
subroutine noah_main()
```

USES:

```
  use lisdrv_module, only : lis
  use noah_varder      ! NOAH tile variables
  use tile_spmdMod
```

CONTENTS:

```
  soilmtc = 0
!=== Convert LDAS Timestep varname to NOAH timestep varname (DT) (sec)
  do t = 1, di_array(iam)
      dt = lis%t%ts
!=== Bottom Temperature Field
      tbot = noah(t)%tempbot
!=== VEGETATION CLASS
```

```
!=== STATIC VEGETATION PARAMETERS
! ----------------------------------------------------------------------
! VEGETATION PARAMETERS ARE DEPENDENT ON VEGETATION TYPE (INDEX)
!   SHDFAC: VEGETATION GREENNESS FRACTION
!   RSMIN:  MIMIMUM STOMATAL RESISTANCE
!   RGL:    PARAMETER USED IN SOLAR RAD TERM OF
!           CANOPY RESISTANCE FUNCTION
!   HS:     PARAMETER USED IN VAPOR PRESSURE DEFICIT TERM OF
!           CANOPY RESISTANCE FUNCTION
!   SNUP:   THRESHOLD SNOW DEPTH (IN WATER EQUIVALENT M) THAT
!           IMPLIES 100% SNOW COVER
!   Z0:     Roughness Length
!   XLAI:   Leaf Area Index
! ----------------------------------------------------------------------
! SSIB VEGETATION TYPES (DORMAN AND SELLERS, 1989; JAM)
!  1:  BROADLEAF-EVERGREEN TREES  (TROPICAL FOREST)
!  2:  BROADLEAF-DECIDUOUS TREES
!  3:  BROADLEAF AND NEEDLELEAF TREES (MIXED FOREST)
!  4:  NEEDLELEAF-EVERGREEN TREES
!  5:  NEEDLELEAF-DECIDUOUS TREES (LARCH)
!  6:  BROADLEAF TREES WITH GROUNDCOVER (SAVANNA)
!  7:  GROUNDCOVER ONLY (PERENNIAL)
!  8:  BROADLEAF SHRUBS WITH PERENNIAL GROUNDCOVER
!  9:  BROADLEAF SHRUBS WITH BARE SOIL
! 10:  DWARF TREES AND SHRUBS WITH GROUNDCOVER (TUNDRA)
! 11:  BARE SOIL
! 12:  CULTIVATIONS (THE SAME PARAMETERS AS FOR TYPE 7)
! 13:  GLACIAL (THE SAME PARAMETERS AS FOR TYPE 11)
! ----------------------------------------------------------------------

      NROOT = NOAH(T)%VEGP(1)
      RSMIN = NOAH(T)%VEGP(2)
      RGL   = NOAH(T)%VEGP(3)
      HS    = NOAH(T)%VEGP(4)
      SNUP  = NOAH(T)%VEGP(5)
      Z0    = NOAH(T)%VEGP(6)
      XLAI  = NOAH(T)%VEGP(7)
!=== MONTHLY VEGETATION PARAMETERS

      SHDFAC = NOAH(T)%VEGIP
!    Minimum greenness fraction
      SHDMIN=0.0

!    If ground surface is bare:
      IF (noah(t)%VEGT .EQ. 11) SHDFAC = 0.0
```

```
!=== STATIC SOIL PARAMETERS
! ----------------------------------------------------------------------
!  SOIL PARAMETERS ARE DEPENDENT ON SOIL TYPE (INDEX)
!    SMCMAX: MAX SOIL MOISTURE CONTENT (POROSITY)
!    SMCREF: REFERENCE SOIL MOISTURE (ONSET OF SOIL MOISTURE
!             STRESS IN TRANSPIRATION)
!    SMCWLT: WILTING PT SOIL MOISTURE CONTENT
!    SMCDRY: AIR DRY SOIL MOIST CONTENT LIMITS
!    PSISAT: SATURATED SOIL POTENTIAL
!    DKSAT:  SATURATED SOIL HYDRAULIC CONDUCTIVITY
!    BEXP:   THE 'B' PARAMETER
!    DWSAT:  SATURATED SOIL DIFFUSIVITY
!    F1:     USED TO COMPUTE SOIL DIFFUSIVITY/CONDUCTIVITY
!    QUARTZ: SOIL QUARTZ CONTENT
! ----------------------------------------------------------------------
! SOIL TYPES   ZOBLER (1986)      COSBY ET AL (1984) (quartz cont.(1))
!  1        COARSE            LOAMY SAND         (0.82)
!  2        MEDIUM            SILTY CLAY LOAM    (0.10)
!  3        FINE              LIGHT CLAY         (0.25)
!  4        COARSE-MEDIUM     SANDY LOAM         (0.60)
!  5        COARSE-FINE       SANDY CLAY         (0.52)
!  6        MEDIUM-FINE       CLAY LOAM          (0.35)
!  7        COARSE-MED-FINE   SANDY CLAY LOAM    (0.60)
!  8        ORGANIC           LOAM               (0.40)
!  9        GLACIAL LAND ICE  LOAMY SAND         (NA using 0.82)
! ----------------------------------------------------------------------

      NSOIL  =  4             ! 4 soil layers in NOAH

      SLDPTH(1) = 0.1         ! Soil layer thicknesses (m)
      SLDPTH(2) = 0.3
      SLDPTH(3) = 0.6
      SLDPTH(4) = 1.0

      SOILTYP = NOAH(T)%ZOBSOIL(1)    ! Zobler Soil Class Value

      SMCMAX =  NOAH(T)%SOILP(1)
      PSISAT =  NOAH(T)%SOILP(2)
      DKSAT  =  NOAH(T)%SOILP(3)
      BEXP   =  NOAH(T)%SOILP(4)
      QUARTZ =  NOAH(T)%SOILP(5)

!   The following 5 parameters are just given here for reference
!    and to force static storage allocation.

!       SMCREF =  NOAH%SOILP(6)
!       SMCWLT =  NOAH%SOILP(7)
!       SMCDRY =  NOAH%SOILP(8)
```

```
!       DWSAT  =  NOAH%SOILP(9)
!       F1     =  NOAH%SOILP(10)

!-- Here is where the above five parameters are actually derived.
!    SET TWO SOIL MOISTURE WILT, SOIL MOISTURE REFERENCE PARAMETERS
      SMLOW = 0.5
!     Changed in 2.6 from 3 to 6 on June 2nd 2003
!       SMHIGH = 3.0
      SMHIGH = 6.0

      DWSAT  = BEXP * DKSAT * (PSISAT/SMCMAX)
      F1     = ALOG10(PSISAT) + BEXP*ALOG10(SMCMAX) + 2.0
      SMCREF1 = SMCMAX*(5.79E-9/DKSAT)**(1.0/(2.0*BEXP+3.0))
      SMCREF = SMCREF1 + (SMCMAX-SMCREF1) / SMHIGH
      SMCWLT1 = SMCMAX * (200.0/PSISAT)**(-1.0/BEXP)
      SMCWLT = SMCWLT1 - SMLOW * SMCWLT1
!    Current version SMCDRY values equate to SMCWLT
      SMCDRY = SMCWLT

! ----------------------------------------------------------------------
! KDT IS DEFINED BY REFERENCE REFKDT AND DKSAT; REFDK=2.E-6 IS THE SAT.
! DK. VALUE FOR THE SOIL TYPE 2
! ----------------------------------------------------------------------
      REFDK=2.0E-6
      REFKDT=3.0
      KDT = REFKDT * DKSAT/REFDK

! ----------------------------------------------------------------------
! FROZEN GROUND PARAMETER, FRZK, DEFINITION: ICE CONTENT THRESHOLD ABOVE
! WHICH FROZEN SOIL IS IMPERMEABLE REFERENCE VALUE OF THIS PARAMETER FOR
! THE LIGHT CLAY SOIL (TYPE=3) FRZK = 0.15 M.
! ----------------------------------------------------------------------
      FRZK=0.15
! ----------------------------------------------------------------------
! TO ADJUST FRZK PARAMETER TO ACTUAL SOIL TYPE: FRZK * FRZFACT
! ----------------------------------------------------------------------
      FRZFACT = (SMCMAX / SMCREF) * (0.412 / 0.468)
      FRZX = FRZK * FRZFACT

!=== SLOPE TYPE
! ----------------------------------------------------------------------
! CLASS PARAMETER 'SLOPETYP' WAS INCLUDED TO ESTIMATE LINEAR RESERVOIR
! COEFFICIENT 'SLOPE' TO THE BASEFLOW RUNOFF OUT OF THE BOTTOM LAYER.
! LOWEST CLASS (SLOPETYP=0) MEANS HIGHEST SLOPE PARAMETER = 1.
! DEFINITION OF SLOPETYP FROM 'ZOBLER' SLOPE TYPE:
! SLOPE CLASS  PERCENT SLOPE
! 1            0-8
! 2            8-30
```

```
! 3            > 30
! 4            0-30
! 5            0-8 & > 30
! 6            8-30 & > 30
! 7            0-8, 8-30, > 30
! 8            GLACIAL ICE
! 9            OCEAN/SEA
! ----------------------------------------------------------------------

!--  SLOPETYP = 3
      SLOPE = 1.0

!=== MONTHLY (QUARTERLY, for now) ALBEDO (SNOW-FREE)

      ALB =  NOAH(T)%ALBSF

!    Maximum Albedo over very Deep Snow

      SNOALB = NOAH(T)%MXSNALB

!=== THE FOLLOWING BREAKS DOWN THE FORCING VARIABLES
      SFCTMP = noah(t)%FORCING(1)
      Q2     = noah(t)%FORCING(2)
      SOLDN  = noah(t)%FORCING(3)
      LWDN   = noah(t)%FORCING(4)
      UWIND  = (noah(t)%FORCING(5))*(noah(t)%FORCING(5))
      VWIND  = (noah(t)%FORCING(6))*(noah(t)%FORCING(6))
      SFCSPD = SQRT( UWIND + VWIND )
      SFCPRS = noah(t)%FORCING(7)
      PRCP   = noah(t)%FORCING(8)
      CPCP   = noah(t)%FORCING(9)   !Convective Precipitation (kg/m2sec)
!-- Height of observations (this needs to be modified)
      Z = 6.0       ! Height of observations (m)

!-- Prevent Numerical Instability for Wind Speed
      if(SFCSPD.le.0.01) SFCSPD=0.01

!-- Prevent Numerical Instability with HUMIDITY

      IF (Q2 .LT. 0.1E-5)  Q2 = 0.1E-5

! Calculate Saturation Specific Humidity (Kg/Kg) and
!    Saturation vapor pressure for water (Pa) based on Specific
!    Humidity(Kg/Kg), Temperature(K), and Pressure (Pa)
!
!      FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989: 'A
!      SHORT COURSE IN CLOUD PHYSICS', PERGAMON PRESS, 3rd ED.
!        Pablo J. Grunmann, 3/6/98.
```

```
!
!      QSAT  = Saturation Specific humidity (Kg/Kg)
!      ESAT  = Saturation vapor pressure for water (Pa)
!      EPS   = Water/(dry air) molecular mass ratio (epsilon)
!      E     = Saturation vapor pressure
!
!-- Function E(SFCTMP) = Sat. vapor pressure (in Pascal) at
!                   temperature T (uses Clausius-Clapeyron).

      ESAT = E(SFCTMP)

!-- CALCULATE SATURATION MIXING RATIO (PABLO GRUNMANN, 05/28/98)

      Q2SAT = 0.622 * ESAT /(SFCPRS - (1.-0.622)*ESAT)
      IF (Q2 .GE.  Q2SAT)  Q2 = Q2SAT*0.99

!-- CALCULATE SLOPE OF SAT. SPECIFIC HUMIDITY CURVE FOR PENMAN: DQSDT2

      DQSDT2 = DQSDT (SFCTMP, SFCPRS)

!-- CALC VIRTUAL TEMPS AND POTENTIAL TEMPS AT GRND (SUB 1) AND AT
!    THE 1ST MDL LVL ABV THE GRND (SUB 2). EXPON IS CP DIVD BY R.

      TH2 = SFCTMP + ( 0.0098 * Z )
      T2V = SFCTMP * (1.0 + 0.61 * Q2 )

      T1V  =  noah(t)%T1 * (1.0 + 0.61 * Q2 )
      TH2V = TH2 * (1.0 + 0.61 * Q2 )

!==  OPTIONAL SUBROUTINE:  Calculate LW Radiation (Down)  =================
!      CALL OBTLWDN(SFCTMP,LWDN)

!--  Phota Thermal Unit (PTU)

      PTU    = 0.10

!--  Initialize SOILM for 1st timestep water balance
!       SOILM = 0.0
!--  Initialize ROOT ZONE COLUMN SOIL MOISTURE IN METERS (SOILRZ)
      SOILRZ = 0.0
!--  Initialize TOP 1-METER COLUMN SOIL MOISTURE IN METERS (SOIL1M)
      SOIL1M = 0.0

!==  CALCULATE CH (EXCHANGE COEFFICIENT)  ================================
!
!   CH IS THE SFC EXCHANGE COEFFICIENT FOR HEAT/MOISTURE
!   CM IS THE SFC EXCHANGE COEFFICIENT FOR MOMENTUM
!
```

```
! IMPORTANT NOTE: TO CALCULATE THE SFC EXCHANGE COEF (CH) FOR HEAT AND
!                  MOISTURE, SUBROUTINE SFCDIF BELOW CAN:
!
!    A)  BE CALLED HERE FROM THE DRIVER, THUS CH IS INPUT TO SFLX
!            (AS IS TYPICAL IN A COUPLED ATMOSPHERE/LAND MODEL), OR
!  * B)  BE CALLED INTERNALLY IN ROUTINE SFLX (THUS CH IS OUTPUT FROM SFLX),
!          BEFORE THE CALL TO ROUTINE "PENMAN"
!
!   OPTION B IS THE DEFAULT HERE.  THAT IS, IN THE UNCOUPLED, OFF-LINE LSM
!   REPRESENTED HEREIN BY THIS DRIVER, WE CALL SFCDIF LATER IN ROUTINE SFLX.
!
!   THE ROUTINE SFCDIF REPRESENTS THE SO-CALLED "SURFACE LAYER" OR THE
!   "CONSTANT FLUX LAYER" (THE LOWEST 20-100 M OF THE ATMOSPHERE).
!   HENCE ROUTINE SFCDIF EMBODIES THE "ATMOSPHERIC AERODYNAMIC RESISTANCE".
!
!   TO ENABLE THE FLEXIBILITY OF EITHER OPTION A OR B, WE PASS
!   THE ARGUMENTS "CH", "CM", AND "SFCSPD" (WIND SPEED:JUST CALCULATED ABOVE)
!   TO ROUTINE SFLX TO SUPPORT OPTION B -- THAT IS, FOR INPUT TO THE CALL TO
!   ROUTINE SFCDIF THEREIN.  IN OPTION A, THE ARGUMENTS "SFCSPD" AND "CM"
!   ARE NEITHER NEEDED IN ROUTINE SFLX, NOR ALTERED BY ROUTINE SFLX.
!
!     IF ONE CHOOSES OPTION A, THEN ONE MUST
!      1 - ACTIVATE (UNCOMMENT) THE CALL TO SFCDIF BELOW,
!      2 - ACTIVATE (UNCOMMENT) THE ASSIGNMENT OF "Z0" AND "CZIL" NEXT BELOW
!      3 - DE-ACTIVATE (COMMENT OUT) THE CALL TO SFCDIF IN ROUTINE SFLX.
!
!   Z0 and CZIL:
!
!   THE ROUGHNESS LENGTH PARAMETERS "Z0" AND "CZIL" MUST BE SET HERE IN THE
!   DRIVER TO SUPPORT THE "OPTION-A", I.E. THE CALL TO SFCDIF BELOW.  IN SO
!   DOING, THE "Z0" AND "CZIL" ASSIGNED HERE MUST CORRESPOND TO THEIR
!   VALUES, CALLED BY SFLX JUST BEFORE CALL SFCDIF.
!   THUS THE VALUE OF "Z0" ASSIGNED HERE MUST CORRESPOND TO THAT ASSIGNED
!   FOR THE CHOSEN VEG CLASS THAT WAS ALREADY INPUT EARLIER IN THE DRIVER.
!
!   BECAUSE OF THE IMPLICIT ITERATIVE NATURE OF THE "PAULSON" SURFACE-LAYER
!   SCHEME USED IN ROUTINE SFCDIF, CH AND CM ARE CO-DEPENDENT.  SIMILARLY,
!   THE IMPLICIT NATURE OF THE SFCDIF SCHEME ALSO REQUIRES THAT FOR EITHER
!   OPTION A OR B, CH AND CM MUST BE INITIALIZED EARLIER IN THE DRIVER BEFORE
!   THE START OF THE TIME-STEP LOOP, AS WELL AS BE CARRIED FORWARD FROM
!   TIME STEP TO TIME STEP AS "STATE VARIABLES", BECAUSE THE VALUES OF
!   CH AND CM FROM A PREVIOUS TIME STEP REPRESENT THE FIRST-GUESS VALUES FOR
!   THE CALL TO SFCDIF IN THE PRESENT TIME STEP.
!
!   SOME USERS MAY CHOOSE TO EXECUTE AN ENTIRELY DIFFERENT SCHEME IN PLACE OF
!   ROUTINE SFCDIF HERE, E.G. AN EXPLICIT SCHEME SUCH AS LOUIS (1979) THAT
!   EMPLOYS NO ITERATION AND HAS NO REQUIREMENT TO CARRY CH AND CM FORWARD
!   AS STATE VARIABLES FROM TIME STEP TO TIME STEP.  IN THAT CASE, IN
```

```
!   OPTION A, THE ROUTINE SHOULD BE CALLED HERE IN THE DRIVER AFTER ALL
!   NECESSARY INPUT ARGUMENTS FOR IT ARE DEFINED AT THIS POINT, OR CALLED IN
!   ROUTINE SFLX, AT THE POINT SFCDIF IS CALLED.
!
!      CALL SFCDIF ( Z, Z0, T1V, TH2V, SFCSPD,CZIL, CM, CH )
!
! ---------------------------------------------------------------------|
! INITIALIZE CH, CM (NOTE: initial these before time loop)
!      CH=1.E-4
!      CM=1.E-4
! 1998 May 22 0030 (Julian= 142) typical values initialization
!      CH=  0.0150022404
!      CM=  0.0205970779
! **NOTE: TRYING THESE VALUES AS TEST!
! ---------------------------------------------------------------------|
!=== MAIN CALL TO LAND-SURFACE PHYSICS  <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
      CALL SFLX (T,&
       ICE,DT,Z,NSOIL,SLDPTH,&
       LWDN,SOLDN,SFCPRS,PRCP,SFCTMP,Q2,SFCSPD, &
       TH2,Q2SAT,DQSDT2,&
       SLOPE,SHDFAC,SHDMIN,PTU,ALB,SNOALB, &
       RSMIN,RGL,HS,SNUP,Z0,XLAI,NROOT,&
       PSISAT,BEXP,DKSAT,SMCMAX,QUARTZ,DWSAT, &
       SMCWLT,SMCREF,SMCDRY,F1,KDT,FRZX,FRZFACT,TBOT, &
       NOAH(T)%CMC,NOAH(T)%T1,NOAH(T)%STC,NOAH(T)%SMC,NOAH(T)%SH2O, &
       NOAH(T)%SNOWH,NOAH(T)%SNEQV,ALBEDO,NOAH(T)%CH,NOAH(T)%CM,&
       EVP,ETA,SHTFLX, &
       EC,EDIR,ET,ETT,ESNOW,DRIP,DEW, &
       BETA,ETP,GFLX, &
       FLX1,FLX2,FLX3,&
       SNOMLT,SNCOVR,&
       RUNOFF1,RUNOFF2,RUNOFF3, &
       RC,PC,RCS,RCT,RCQ,RCSOIL, &
       MSTAVRZ,MSTAVTOT,SOILM)
!*******************************************************************
!   CALCULATE UPWARD LONGWAVE RAD USING UPDATED SKIN TEMPERATURE

      T14 = noah(t)%T1 * noah(t)%T1 * noah(t)%T1 * noah(t)%T1
      FUP = 5.67E-8 * T14

!   CALCULATE RESIDUAL OF ALL SURFACE ENERGY BALANCE EQN TERMS.

!      GFLX = -GFLX
!      F = SOLDN*(1.0-ALBEDO) + LWDN
!      RES    = F - SHTFLX - GFLX - ETA - FUP - FLX1 - FLX2 - FLX3
!         ENDIF

!      PRINT*,'  ------------------------------------'
```

```
!      PRINT*,'  State Variables '
!      PRINT*,'  ------------------------------------'
!      WRITE(*,*) NOAH(T)%T1,' T1...Skin temperature (K)'
!      WRITE(*,*)(NOAH(T)%STC(IJ), IJ=1,NSOIL),' STC'
!      WRITE(*,*)(NOAH(T)%SMC(IJ), IJ=1,NSOIL),' SMC'
!      WRITE(*,*)(NOAH(T)%SH2O(IJ), IJ=1,NSOIL),' SH2O'
!      WRITE(*,*) NOAH(T)%CMC,' CMC...Canopy water content (m)'
!      WRITE(*,*) NOAH(T)%SNOWH,' SNOWH...Actual snow depth (m)'
!      WRITE(*,*) NOAH(T)%SNEQV,' SNEQV...Water equiv snow depth (m)'
!      WRITE(*,*) 'CH= ',NOAH(T)%CH,'   CM= ',NOAH(T)%CM
!      PRINT*,'  ------------------------------------'

!=== Collect the output variables into NOAH(T)%RETURN
      noah(t)%swnet = noah(t)%swnet+soldn*(1.0-albedo)
      noah(t)%lwnet = noah(t)%lwnet+(5.67E-8)*(NOAH(T)%T1**4.0)-LWDN
      noah(t)%qle = noah(t)%qle+eta
      noah(t)%qh = noah(t)%qh+shtflx
      noah(t)%qg = noah(t)%qg-gflx

      if (sfctmp .le. t0) then
         noah(t)%snowf = noah(t)%snowf+prcp
         noah(t)%rainf = noah(t)%rainf+0.0
      else
         noah(t)%snowf = noah(t)%snowf+0.0
         noah(t)%rainf = noah(t)%rainf+prcp
      endif
      noah(t)%evap = noah(t)%evap+evp
      noah(t)%qs = noah(t)%qs+runoff1*1000.0
      noah(t)%qsb = noah(t)%qsb+ runoff2*1000.0
      noah(t)%qsm = noah(t)%qsm+ snomlt*1000.0
      NOAH(T)%avgsurft=NOAH(T)%T1
      NOAH(T)%albedo=ALBEDO
      noah(t)%swe = noah(t)%sneqv*1000.0

! NOTE:  Soil temperature for each layer is passed directly to output routines.

      NOAH(T)%soilmoist1 = NOAH(T)%SMC(1)*1000.0*SLDPTH(1)
      NOAH(T)%soilmoist2 = NOAH(T)%SMC(2)*1000.0*SLDPTH(2)
      NOAH(T)%soilmoist3 = NOAH(T)%SMC(3)*1000.0*SLDPTH(3)
      NOAH(T)%soilmoist4 = NOAH(T)%SMC(4)*1000.0*SLDPTH(4)

      NOAH(T)%soilwet = MSTAVTOT
      NOAH(T)%ecanop = noah(t)%ecanop + ec*1000.0
      NOAH(T)%tveg = noah(t)%tveg+ETT*1000.0
      NOAH(T)%esoil = noah(t)%esoil+EDIR*1000.0
      noah(t)%canopint = noah(t)%cmc*1000.0
! ROOT ZONE COLUMN SOIL MOISTURE IN METERS (SOILRZ)
      do k = 1,nroot
```

```
         soilrz = soilrz+(noah(t)%smc(k)*sldpth(k)*1000.0)
      end do
      noah(t)%rootmoist = soilrz

      if(lis%t%tscount == 0 .or. lis%t%tscount ==1 &
           .or.lis%f%rstflag.eq.1) then

         soilmtc(t) = soilmtc(t)+noah(t)%soilmoist1+ &
              noah(t)%soilmoist2+ noah(t)%soilmoist3+&
              noah(t)%soilmoist4
         noah(t)%soilm_prev = soilmtc(t)
         noah(t)%swe_prev = noah(t)%swe
      endif

      noah(t)%count=noah(t)%count+1

      enddo
      return

%//////////////////////////////////////////////////////////////
{\sf CONTENTS:}
 \begin{verbatim}
    end subroutine noah_main


!*** NOAH FUNCTIONS ***************************************************

      FUNCTION DQS (T)

      IMPLICIT NONE

!
!  PURPOSE:  TO CALCULATE VALUES OF VAPOR PRESSURE (E)
!            AND P * DQS/DT (P TIMES CHG IN SAT MXG RATIO WITH RESPECT
!            TO THE CHG IN TEMP) IN SUBSTITUTION TO THE LOOK-UP TABLES.
!
!            FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989.
!                         ADDED BY PABLO J. GRUNMANN, 6/30/97.
!

      REAL DESDT
      REAL DQS
!    REAL ESD
      REAL LW
      REAL T
      REAL ES

!      REAL, PARAMETER:: CP = 1005.
```

```
!      REAL, PARAMETER:: CV = 718.
!      REAL, PARAMETER:: CVV = 1410.
      REAL, PARAMETER:: CPV = 1870.
      REAL, PARAMETER:: RV = 461.5
      REAL, PARAMETER:: CW = 4187.
      REAL, PARAMETER:: EPS = 0.622
      REAL, PARAMETER:: ESO = 611.2
      REAL, PARAMETER:: TO = 273.15
      REAL, PARAMETER:: LVH2O = 2.501000E+6


!     ABOUT THE PARAMETERS:
!
!     EPS ---------- WATER - DRY AIR MOLECULAR MASS RATIO, EPSILON
!
!   VALUES FOR SPECIFIC HEAT CAPACITY AND INDIVIDUAL GAS CONSTANTS
!   IN [JOULES/(KG*KELVIN)] UNITS.
!
!     DRY AIR:
!             CP, CV
!     WATER VAPOR:
!                 CVV = 1410.
!                 CPV = 1870.
!                 RV  =  461.5
!     LIQUID WATER:
!                  CW = 4187.
!
!     ESO = ES(T=273.15 K) = SAT. VAPOR PRESSURE (IN PASCAL) AT T=TO
!      TO = 273.15
!
!     SAT. MIXING  RATIO: QS ~= EPS*ES/P
!     CLAUSIUS-CLAPEYRON: DES/DT = L*ES/(RV*T^2)
!     @QS/@T =  (EPS/P)*DES/DT

          LW = LVH2O - ( CW - CPV ) * ( T - TO )
          ES = ESO*EXP (LW*(1/TO - 1/T)/RV)
          DESDT = LW*ES/(RV*T*T)

!    FOR INSERTION IN DQSDT FUNCTION:
!    DQSDT = DQS/P , WHERE DQS = EPS*DESDT

          DQS = EPS*DESDT

          RETURN
          END

!-----------------------------------------------------------------------
```

```
      FUNCTION DQSDT ( SFCTMP, SFCPRS )

      IMPLICIT NONE

!
!    PURPOSE:  TO RETRIEVE THE APPROPRIATE VALUE OF DQSDT (THE CHANGE
!    =======   OF THE SATURATION MIXING RATIO WITH RESPECT TO THE
!              CHANGE IN TEMPERATURE) FROM:
!
!              FORMULAS INTRODUCED IN FUNCTION DQS
!                                  (MODIFIED BY PABLO GRUNMANN, 7/9/97).
!

      REAL SFCTMP
      REAL SFCPRS
      REAL DQS
      REAL DQSDT

      IF ((SFCTMP .GE. 173.0) .AND. (SFCTMP  .LE.  373.0)) THEN

!  IF THE INPUT SFC AIR TEMP IS BTWN 173 K AND 373 K, USE
!   FUNCTION DQS TO DETERMINE THE SLOPE OF SAT.MIX RATIO FUNCTION

        DQSDT = DQS (SFCTMP) / SFCPRS

      ELSE

!  OTHERWISE, SET DQSDT EQUAL TO ZERO

        DQSDT = 0.0

      END IF

      RETURN
      END

!-----------------------------------------------------------------

      FUNCTION E(T)

      IMPLICIT NONE

!
!  PURPOSE:  TO CALCULATE VALUES OF SAT. VAPOR PRESSURE (E)
!            FORMULAS AND CONSTANTS FROM ROGERS AND YAU, 1989.
!                         ADDED BY PABLO J. GRUNMANN, 7/9/97.
!
```

```
      REAL LW
      REAL T
      REAL E

!      REAL, PARAMETER:: EPS = 0.622
!      REAL, PARAMETER:: CP = 1005.
!      REAL, PARAMETER:: CV = 718.
!      REAL, PARAMETER:: CVV = 1410.
      REAL, PARAMETER:: CPV = 1870.
      REAL, PARAMETER:: RV = 461.5
      REAL, PARAMETER:: CW = 4187.
      REAL, PARAMETER:: ESO = 611.2
      REAL, PARAMETER:: TO = 273.15
      REAL, PARAMETER:: LVH2O = 2.501000E+6

!   ABOUT THE PARAMETERS:
!
!    EPS --- WATER - DRY AIR MOLECULAR MASS RATIO, EPSILON
!
!    VALUES FOR SPECIFIC HEAT CAPACITY AND INDIVIDUAL GAS CONSTANTS
!    IN [JOULES/(KG*KELVIN)] UNITS.
!
!     DRY AIR:
!             CP, CV
!     WATER VAPOR:
!             CVV = 1410.
!             CPV = 1870.
!             RV  =  461.5
!     LIQUID WATER:
!             CW = 4187.
!
!     ESO = ES(TO) = SAT. VAPOR PRESSURE (IN PASCAL) AT T=TO
!      TO = 273.15
!
!     CLAUSIUS-CLAPEYRON: DES/DT = L*ES/(RV*T^2)

          LW = LVH2O - ( CW - CPV ) * ( T - TO )
          E = ESO*EXP (LW*(1/TO - 1/T)/RV)

          RETURN
          END


!CC 1. DRIVER SUBROUTINE ==> SUBROUTINE OBTLWDN CCCCCCCCCCCCCCCCCC

!      SUBROUTINE OBTLWDN(SFCTMP,LWDN)

!                       RADIATION
```

```
!
! The following step (OBTENTION OF LWDN) is used if
! user wants to calculate longwave downward.
!
! OBTENTION OF LWDN <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
!
! COMPUTATION OF LWDN (INCOMING LW RADIATION) FROM TAIR AND Q:
!
!...............LWDN = EMISS*SIGMA*(TAK)^4.
!
! WHERE:   TAK = AIR TEMP IN KELVIN
!        EMISS = 0.7  OR  (IDSO AND JACKSON, 1969):
!
!        EMISS = (1 - 0.261 EXP(-7.77*10^(-4)X(273-TAK)^2)
!
!      NEED STEFAN-BOLTZMANN CONSTANT, SIGMA
!         SIGMA = 5.672 * 10^-8  W M^-2 T^-4
!
!           SIGMA = 5.672E-8
!           TAK = SFCTMP
!           EMISS = 1 - 0.261*EXP((-7.77E-4)*(273-TAK)^2.)
!
!           LWDN = EMISS*SIGMA*TAK^4.
!
!        RETURN
!        END

!CCCC  END OF DRIVER SUBROUTINES CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

\markboth{Left}{Source File: noah\_module.F90,  Date: Fri Feb 11 15:41:05 EST 2005
}

!-----------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
```

```
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------
```

1.74 Fortran: Module Interface noah_module.F90 (Source File: noah_module.F90)

Module for 1-D NOAH land model driver variable specification.

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault added NOAH LSM 2.5 code to LDAS.
14 Nov 2002: Sujay Kumar Optimized version for LIS
```

INTERFACE:

```
module noah_module
```

CONTENTS:

```
  type noahdec

     INTEGER :: ts                 !Timestep (seconds)
     INTEGER :: maxt               !Maximum tiles per grid
     INTEGER :: SIBVEG             !UMD to SIB Vegetation Class Index value
     INTEGER :: NSLAY              !Number of NOAH soil layers (4)
     INTEGER :: COUNT
     INTEGER :: ZOBSOIL(1)  !Zobler Soil Classes (LIS%NCH)

     REAL:: VEGP(7)      !Static vegetation parameter values, dim(NOAH_NVEGP)
     REAL::VEGIP     !Interpolated Green Fraction from monthly parameters
     REAL:: VEGMP1       !Month 1 Greenness Fraction Value
     REAL:: VEGMP2       !Month 2 Greenness Fraction Value
     REAL:: ALBSF1       !Date 1 Snow-Free Albedo Value
     REAL:: ALBSF2       !Date 2 Snow-Free Albedo Value
     REAL:: SOILP(10)    !Static soil parameter values, dim(NOAH_NSOILP)
     REAL:: ALBSF     !Quarterly Snow-Free Albedo dataset
     REAL:: MXSNALB   !Maximum snow albedo dataset
     REAL:: TEMPBOT   !Bottom boundary temperature
!-----------------------------------------------------------------------
! NOAH-State Variables
!-----------------------------------------------------------------------
     REAL :: T1                 !NOAH Skin Temperature (K)
     REAL :: CMC                !NOAH Canopy Water Content
     REAL :: SNOWH              !NOAH Actual Snow depth (m)
     REAL :: SNEQV              !NOAH Water Equivalent Snow Depth (m)
     REAL :: STC(4)    !NOAH Soil Temperaure (4 layers)
```

```
     REAL :: SMC(4)    !NOAH Soil (4 layers)
     REAL :: SH2O(4)   !NOAH Liquid-only soil moisture (4 layers)
     REAL :: CH                 !NOAH Heat/moisture exchange coef.
     REAL :: CM                 !NOAH Momentum exchange coef.
     REAL :: FORCING(10)        ! TILE FORCING..
     REAL :: VEGT              !vegetation type of tile
!-----------------------------------------------------------------------
!  NOAH-Output variables
!-----------------------------------------------------------------------
     REAL :: swnet
     REAL :: lwnet
     REAL :: qle
     REAL :: qh
     REAL :: qg
     REAL :: snowf
     REAL :: rainf
     REAL :: evap
     REAL :: qs
     REAL :: qsb
     REAL :: qsm
     REAL :: avgsurft
     REAL :: albedo
     REAL :: swe
     REAL :: soilmoist1
     REAL :: soilmoist2
     REAL :: soilmoist3
     REAL :: soilmoist4
     REAL :: soilwet
     REAL :: ecanop
     REAL :: canopint
     REAL :: tveg
     REAL :: esoil
     REAL :: rootmoist
     REAL :: soilm_prev
     REAL :: swe_prev
  end type noahdec
```

1.74.1 noah_binout.F90 (Source File: noah_netcdfout.F90)

LIS NOAH data writer: Writes noah output in binary format

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine noah_netcdfout(check, ftn)
```

USES:

```
#if (USE_NETCDF)
  use netcdf
#endif
  use lisdrv_module, only : lis
  use drv_output_mod, only : drv_writevar_netcdf, drv_writevar_netcdf3d
  use noah_varder

  implicit none

  integer :: ftn, dim1
  logical :: check
```

CONTENTS:

```
#if (USE_NETCDF)
  if(.not. check) then
     if((mod(lis%t%yr,4).eq.0.and.mod(lis%t%yr,100).ne.0) &
          .or.(mod(lis%t%yr,400).eq.0)) then
        leap = .true.
     else
        leap = .false.
     endif
     if(lis%t%mo .eq. lis%t%emo) then
        dim1 = ((lis%t%eda-lis%t%da -1)*24+&
             lis%t%ehr+(24-lis%t%hr))/noahdrv%writeintn
     else
        if(leap) then
           dim1 = ((24-lis%t%hr)+&
                (days2(lis%t%mo)-lis%t%da)*24)/noahdrv%writeintn
        else
           dim1 = ((24-lis%t%hr)+&
                (days1(lis%t%mo)-lis%t%da)*24)/noahdrv%writeintn
        endif
     endif
     sldepth(1) = 0.1
     sldepth(2) = 0.3
     sldepth(3) = 0.6
     sldepth(4) = 1.0

     call check_nc(nf90_def_dim(ftn, 'sld',4,soildim))
     call check_nc(nf90_def_var(ftn, "SoilDepth",nf90_float,&
          dimids = soildim, varID= soilid))
     print*, 'dim1 ',dim1
     call check_nc(nf90_def_dim(ftn, 'land',lis%d%glbnch, dimID(1)))
     call check_nc(nf90_def_dim(ftn, 'time',dim1, dimID(2)))
```

```
call check_nc(nf90_def_dim(ftn, 'land',lis%d%glbnch, dimID1(1)))
call check_nc(nf90_def_dim(ftn, 'time',dim1, dimID1(2)))
call check_nc(nf90_def_dim(ftn, 'soil',4,dimID1(3)))

call check_nc(nf90_def_var(ftn, "Swnet", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(1)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(1),"units","W/m2"))

call check_nc(nf90_def_var(ftn, "Lwnet", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(2)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(2),"units","W/m2"))

call check_nc(nf90_def_var(ftn, "Qle", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(3)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(3),"units","W/m2"))

call check_nc(nf90_def_var(ftn, "Qh", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(4)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(4),"units","W/m2"))

call check_nc(nf90_def_var(ftn, "Qg", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(5)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(5),"units","W/m2"))

call check_nc(nf90_def_var(ftn, "Snowf", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(6)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(6),"units","kg/m2s"))

call check_nc(nf90_def_var(ftn, "Rainf", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(7)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(7),"units","kg/m2s"))

call check_nc(nf90_def_var(ftn, "Evap", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(8)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(8),"units","kg/m2s"))


call check_nc(nf90_def_var(ftn, "Qs", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(9)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(9),"units","kg/m2s"))

call check_nc(nf90_def_var(ftn, "Qsb", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(10)))
call check_nc(nf90_put_att(ftn,noahdrv%varid(10),"units","kg/m2s"))

call check_nc(nf90_def_var(ftn, "Qsm", nf90_float, &
     dimids = dimID, varID = noahdrv%varid(11)))
```

```
    call check_nc(nf90_put_att(ftn,noahdrv%varid(11),"units","kg/m2s"))

    call check_nc(nf90_def_var(ftn, "DelSoilMoist", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(12)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(12),"units","kg/m2s"))


    call check_nc(nf90_def_var(ftn, "DelSWE", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(13)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(13),"units","kg/m2s"))

    call check_nc(nf90_def_var(ftn, "AvgSurfT", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(14)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(14),"units","K"))

    call check_nc(nf90_def_var(ftn, "Albedo", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(15)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(15),"units","-"))

    call check_nc(nf90_def_var(ftn, "SWE", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(16)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(16),"units","kg/m2"))

    call check_nc(nf90_def_var(ftn, "SoilMoist", nf90_float, &
         dimids = dimID1, varID = noahdrv%varid(17)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(17),"units","kg/m2"))

    call check_nc(nf90_def_var(ftn, "SoilWet", nf90_float, &
      dimids = dimID, varID = noahdrv%varid(18)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(18),"units","-"))

    call check_nc(nf90_def_var(ftn, "TVeg", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(19)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(19),"units","kg/m2s"))

    call check_nc(nf90_def_var(ftn, "ESoil", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(20)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(20),"units","kg/m2s"))

    call check_nc(nf90_def_var(ftn, "RootMoist", nf90_float, &
         dimids = dimID, varID = noahdrv%varid(21)))
    call check_nc(nf90_put_att(ftn,noahdrv%varid(21),"units","kg/m2"))
    call check_nc(nf90_enddef(ftn))

    call drv_writevar_netcdf(ftn,sldepth,1,soilid)
  endif
  dim1 = noahdrv%numout
!-----------------------------------------------------------------------------
```

```
! General Energy Balance Components
!-----------------------------------------------------------------------------
  noah%swnet = noah%swnet/float(noah%count)
  call drv_writevar_netcdf(ftn,noah%swnet ,dim1,noahdrv%varid(1))
  noah%lwnet = (-1)*noah%lwnet/float(noah%count)
  call drv_writevar_netcdf(ftn,noah%lwnet ,dim1, noahdrv%varid(2))
  noah%qle = noah%qle/float(noah%count)
  call drv_writevar_netcdf(ftn,noah%qle ,dim1, noahdrv%varid(3))
  noah%qh = noah%qh/float(noah%count)
  call drv_writevar_netcdf(ftn,noah%qh ,dim1, noahdrv%varid(4))
  noah%qg = noah%qg/float(noah%count)
  call drv_writevar_netcdf(ftn,noah%qg ,dim1, noahdrv%varid(5))
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   noah%snowf = noah%snowf/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%snowf ,dim1, noahdrv%varid(6))
   noah%rainf = noah%rainf/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%rainf ,dim1, noahdrv%varid(7))
   noah%evap = noah%evap/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%evap ,dim1, noahdrv%varid(8))
   noah%qs = noah%qs/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%qs ,dim1, noahdrv%varid(9))
   noah%qsb = noah%qsb/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%qsb ,dim1, noahdrv%varid(10))
   noah%qsm = noah%qsm/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%qsm ,dim1, noahdrv%varid(11))
   call drv_writevar_netcdf(ftn,  noah%smc(1)*1000.0*0.1+ &
        noah%smc(2)*1000.0*0.3 + &
        noah%smc(3)*1000.0*0.6 + &
        noah%smc(4)*1000.0 -noah%soilm_prev,dim1, noahdrv%varid(12))
   call drv_writevar_netcdf(ftn,noah%sneqv*1000.0-noah%swe_prev,&
        dim1, noahdrv%varid(13))
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call drv_writevar_netcdf(ftn,noah%avgsurft ,dim1, noahdrv%varid(14))
   call drv_writevar_netcdf(ftn,noah%albedo ,dim1, noahdrv%varid(15))
   noah%swe= noah%swe/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%swe ,dim1, noahdrv%varid(16))
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   noah%soilmoist1= noah%soilmoist1/float(noah%count)
   call drv_writevar_netcdf3d(ftn,noah%soilmoist1, dim1, 1,noahdrv%varid(17))
   noah%soilmoist2= noah%soilmoist2/float(noah%count)
   call drv_writevar_netcdf3d(ftn,noah%soilmoist2, dim1, 2, noahdrv%varid(17))
   noah%soilmoist3= noah%soilmoist3/float(noah%count)
```

```
   call drv_writevar_netcdf3d(ftn,noah%soilmoist3, dim1, 3,noahdrv%varid(17))
   noah%soilmoist4= noah%soilmoist4/float(noah%count)
   call drv_writevar_netcdf3d(ftn,noah%soilmoist4, dim1, 4,noahdrv%varid(17))
   noah%soilwet= noah%soilwet/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%soilwet ,dim1, noahdrv%varid(18))
!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
   noah%tveg= noah%tveg/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%tveg ,dim1, noahdrv%varid(19))
   noah%esoil= noah%esoil/float(noah%count)
   call drv_writevar_netcdf(ftn,noah%esoil ,dim1, noahdrv%varid(20))
   noah%rootmoist = noah%rootmoist/float(noah%count)
   call drv_writevar_netcdf(ftn, noah%rootmoist ,dim1, noahdrv%varid(21))
```

1.74.2 noah_output.F90 (Source File: noah_output.F90)

This subroutines sets up methods to write noah output

INTERFACE:

```
 subroutine noah_output
```

USES:

```
   use lisdrv_module, only : lis, tile, glbgindex
   use noah_varder, only : noahdrv
   use spmdMod, only : masterproc,npes
```

CONTENTS:

```
!-------------------------------------------------------------------
! Number of variables to be outputted from Noah LSM
!-------------------------------------------------------------------
  if ( lis%o%wsingle == 1 ) then
     if ( lis%o%wfor == 0 ) then
        num_vars = 30
     else
        num_vars = 38
     endif
!-------------------------------------------------------------------
! Writes each output variable to a separate file
!-------------------------------------------------------------------
     if ( mod(lis%t%gmt,noahdrv%writeintn) == 0 ) then
        if(lis%d%gridDesc(9).eq.0.01) then
           call system('/data1/pool/control/traffic-wait.pl')
        endif
        do i=1,num_vars
```

```
          call noah_singlegather(i,var)
          if ( masterproc ) then
             call noah_singleout(lis, tile, var, i)
          endif
       enddo
       if(lis%d%gridDesc(9).eq.0.01) then
          call system('/data1/pool/control/traffic-go.pl')
       endif
       call noah_totinit()
    endif
  else
!-------------------------------------------------------------------
! Writes bundled output
!-------------------------------------------------------------------
    if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
       if(npes > 1 ) then
          call noah_gather()
       endif
       if(masterproc) then
          call noah_almaout()
       endif
       call noah_totinit()
    endif
  endif
```

1.75 Fortran: Module Interface noahpardef_module.F90 (Source File: noahpardef_module.F90)

This module contains routines that defines MPI derived data types for Noah LSM

REVISION HISTORY:

```
06 Oct 2003; Sujay Kumar  Initial Specification
```

INTERFACE:

```
module noahpardef_module
```

USES:

```
use noah_module
use noahdrv_module
use spmdMod
implicit none
```

ARGUMENTS:

```
#if (defined SPMD)
  integer:: MPI_NOAH_STRUCT  !MPI derived type for noah$_-$module
  integer :: MPI_NOAHDRV_STRUCT !MPI derived type for noahdrv$_-$module
```

1.75.1 def_noahpar_struct (Source File: noahpardef_module.F90)

Routine that defines MPI derived data types for Noah

INTERFACE:

```
subroutine def_noahpar_struct()
```

!ROUTINE : noah_physics.F90
DESCRIPTION: SUB-DRIVER FOR "NOAH/OSU LSM" FAMILY OF PHYSICS SUBROUTINES FOR A SOIL/VEG/SNOWPACK LAND-SURFACE MODEL TO UPDATE SOIL MOISTURE, SOIL ICE, SOIL TEMPERATURE, SKIN TEMPERATURE, SNOWPACK WATER CONTENT, SNOWDEPTH, AND ALL TERMS OF THE SURFACE ENERGY BALANCE AND SURFACE WATER BALANCE (EXCLUDING INPUT ATMOSPHERIC FORCINGS OF DOWNWARD RADIATION AND PRECIP)

REVISION HISTORY:

```
 28 Apr 2002: Kristi Arsenault; Added NOAH LSM 2.5 code to LDAS
 15 May 2002: Urszula Jambor; Changed LOGICAL to LOGICAL*1 to match new
              GRIB libraries
 28 May 2002: Kristi Arsenault; Updated NOAH code to 2.5.1, and
              corrected problem with FX in DEVAP function.
 12 Jun 2002: Kristi Arsenault; Updated NOAH code to 2.5.2
 04 Nov 2002: Kristi Arsenault; Incorporated new TBOT fields
 24 Jun 2003: Kristi Arsenault; Updated Noah LSM to 2.6 version
-----------------------------------------------------------------------
-------  THE FOLLOWING SUBROUTINES ARE IN ALPHABETICAL ORDER  --------

-- 1. PHYSICS SUBROUTINE ==>  SUBROUTINE ALCALC ----------------------

    SUBROUTINE ALCALC (ALB,SNOALB,SHDFAC,SHDMIN,SNCOVR,TSNOW,ALBEDO)

    IMPLICIT NONE

-----------------------------------------------------------------------
CALCULATE ALBEDO INCLUDING SNOW EFFECT (0 -> 1)
  ALB     SNOWFREE ALBEDO
  SNOALB  MAXIMUM (DEEP) SNOW ALBEDO
  SHDFAC    AREAL FRACTIONAL COVERAGE OF GREEN VEGETATION
  SHDMIN    MINIMUM AREAL FRACTIONAL COVERAGE OF GREEN VEGETATION
  SNCOVR  FRACTIONAL SNOW COVER
  ALBEDO  SURFACE ALBEDO INCLUDING SNOW EFFECT
  TSNOW   SNOW SURFACE TEMPERATURE (K)
-----------------------------------------------------------------------
    REAL ALB, SNOALB, SHDFAC, SHDMIN, SNCOVR, ALBEDO, TSNOW

-----------------------------------------------------------------------
SNOALB IS ARGUMENT REPRESENTING MAXIMUM ALBEDO OVER DEEP SNOW,
```

```
AS PASSED INTO SFLX, AND ADAPTED FROM THE SATELLITE-BASED MAXIMUM
SNOW ALBEDO FIELDS PROVIDED BY D. ROBINSON AND G. KUKLA
(1985, JCAM, VOL 24, 402-411)
-----------------------------------------------------------------------
        changed in version 2.6 on June 2nd 2003
         ALBEDO = ALB + (1.0-(SHDFAC-SHDMIN))*SNCOVR*(SNOALB-ALB)
        ALBEDO = ALB + SNCOVR*(SNOALB-ALB)
        IF (ALBEDO .GT. SNOALB) ALBEDO=SNOALB

    BASE FORMULATION (DICKINSON ET AL., 1986, COGLEY ET AL., 1990)
         IF (TSNOW.LE.263.16) THEN
           ALBEDO=SNOALB
         ELSE
           IF (TSNOW.LT.273.16) THEN
             TM=0.1*(TSNOW-263.16)
             ALBEDO=0.5*((0.9-0.2*(TM**3))+(0.8-0.16*(TM**3)))
           ELSE
             ALBEDO=0.67
           ENDIF
         ENDIF

    ISBA FORMULATION (VERSEGHY, 1991; BAKER ET AL., 1990)
         IF (TSNOW.LT.273.16) THEN
           ALBEDO=SNOALB-0.008*DT/86400
         ELSE
           ALBEDO=(SNOALB-0.5)*EXP(-0.24*DT/86400)+0.5
         ENDIF

-----------------------------------------------------------------------
END SUBROUTINE ALCALC
-----------------------------------------------------------------------
    RETURN
    END

CCC 2. PHYSICS SUBROUTINE ==>  SUBROUTINE CANRES  CCCCCCCCCCCCCCCCCCCC

    SUBROUTINE CANRES (SOLAR,CH,SFCTMP,Q2,SFCPRS,SMC,ZSOIL,NSOIL, &
                SMCWLT,SMCREF,RSMIN,RC,PC,NROOT,Q2SAT,DQSDT2, &
                TOPT,RSMAX,RGL,HS,XLAI, &
                RCS,RCT,RCQ,RCSOIL)

    IMPLICIT NONE

-----------------------------------------------------------------------
SUBROUTINE CANRES
-----------------------------------------------------------------------
CALCULATE CANOPY RESISTANCE WHICH DEPENDS ON INCOMING SOLAR RADIATION,
AIR TEMPERATURE, ATMOSPHERIC WATER VAPOR PRESSURE DEFICIT AT THE
```

```
LOWEST MODEL LEVEL, AND SOIL MOISTURE (PREFERABLY UNFROZEN SOIL
MOISTURE RATHER THAN TOTAL)
-----------------------------------------------------------------------
SOURCE:  JARVIS (1976), NOILHAN AND PLANTON (1989, MWR), JACQUEMIN AND
NOILHAN (1990, BLM)
SEE ALSO:  CHEN ET AL (1996, JGR, VOL 101(D3), 7251-7268), EQNS 12-14
AND TABLE 2 OF SEC. 3.1.2
-----------------------------------------------------------------------
INPUT:
  SOLAR   INCOMING SOLAR RADIATION
  CH      SURFACE EXCHANGE COEFFICIENT FOR HEAT AND MOISTURE
  SFCTMP  AIR TEMPERATURE AT 1ST LEVEL ABOVE GROUND
  Q2      AIR HUMIDITY AT 1ST LEVEL ABOVE GROUND
  Q2SAT   SATURATION AIR HUMIDITY AT 1ST LEVEL ABOVE GROUND
  DQSDT2  SLOPE OF SATURATION HUMIDITY FUNCTION WRT TEMP
  SFCPRS  SURFACE PRESSURE
  SMC     VOLUMETRIC SOIL MOISTURE
  ZSOIL   SOIL DEPTH (NEGATIVE SIGN, AS IT IS BELOW GROUND)
  NSOIL   NO. OF SOIL LAYERS
  NROOT   NO. OF SOIL LAYERS IN ROOT ZONE (1.LE.NROOT.LE.NSOIL)
  XLAI    LEAF AREA INDEX
  SMCWLT  WILTING POINT
  SMCREF  REFERENCE SOIL MOISTURE (WHERE SOIL WATER DEFICIT STRESS
            SETS IN)
RSMIN, RSMAX, TOPT, RGL, HS ARE CANOPY STRESS PARAMETERS SET IN
  SURBOUTINE REDPRM
OUTPUT:
  PC  PLANT COEFFICIENT
  RC  CANOPY RESISTANCE
-----------------------------------------------------------------------
    INTEGER NSOLD
    PARAMETER(NSOLD = 20)

    INTEGER K
    INTEGER NROOT
    INTEGER NSOIL

    REAL CH
    REAL CP
    REAL DELTA
    REAL DQSDT2
    REAL FF
    REAL GX
    REAL HS
    REAL P
    REAL PART(NSOLD)
    REAL PC
    REAL Q2
```

```
    REAL Q2SAT
    REAL RC
    REAL RSMIN
    REAL RCQ
    REAL RCS
    REAL RCSOIL
    REAL RCT
    REAL RD
    REAL RGL
    REAL RR
    REAL RSMAX
    REAL SFCPRS
    REAL SFCTMP
    REAL SIGMA
    REAL SLV
    REAL SMC(NSOIL)
    REAL SMCREF
    REAL SMCWLT
    REAL SOLAR
    REAL TOPT
    REAL SLVCP
    REAL ST1
    REAL TAIR4
    REAL XLAI
    REAL ZSOIL(NSOIL)

    PARAMETER(CP = 1004.5)
    PARAMETER(RD = 287.04)
    PARAMETER(SIGMA = 5.67E-8)
    PARAMETER(SLV = 2.501000E6)

-----------------------------------------------------------------------
INITIALIZE CANOPY RESISTANCE MULTIPLIER TERMS.
-----------------------------------------------------------------------
    RCS = 0.0
    RCT = 0.0
    RCQ = 0.0
    RCSOIL = 0.0
    RC = 0.0

-----------------------------------------------------------------------
CONTRIBUTION DUE TO INCOMING SOLAR RADIATION
-----------------------------------------------------------------------
    FF = 0.55*2.0*SOLAR/(RGL*XLAI)
    RCS = (FF + RSMIN/RSMAX) / (1.0 + FF)
    RCS = MAX(RCS,0.0001)

-----------------------------------------------------------------------
```

```
CONTRIBUTION DUE TO AIR TEMPERATURE AT FIRST MODEL LEVEL ABOVE GROUND
RCT EXPRESSION FROM NOILHAN AND PLANTON (1989, MWR).
------------------------------------------------------------------------
    RCT = 1.0 - 0.0016*((TOPT-SFCTMP)**2.0)
    RCT = MAX(RCT,0.0001)

------------------------------------------------------------------------
CONTRIBUTION DUE TO VAPOR PRESSURE DEFICIT AT FIRST MODEL LEVEL.
RCQ EXPRESSION FROM SSIB
------------------------------------------------------------------------
    RCQ = 1.0/(1.0+HS*(Q2SAT-Q2))
    RCQ = MAX(RCQ,0.01)

------------------------------------------------------------------------
CONTRIBUTION DUE TO SOIL MOISTURE AVAILABILITY.
DETERMINE CONTRIBUTION FROM EACH SOIL LAYER, THEN ADD THEM UP.
------------------------------------------------------------------------
    GX = (SMC(1) - SMCWLT) / (SMCREF - SMCWLT)
    IF (GX .GT. 1.) GX = 1.
    IF (GX .LT. 0.) GX = 0.

------------------------------------------------------------------------
USE SOIL DEPTH AS WEIGHTING FACTOR
------------------------------------------------------------------------
    PART(1) = (ZSOIL(1)/ZSOIL(NROOT)) * GX
------------------------------------------------------------------------
USE ROOT DISTRIBUTION AS WEIGHTING FACTOR
     PART(1) = RTDIS(1) * GX
------------------------------------------------------------------------
    DO K = 2,NROOT
      GX = (SMC(K) - SMCWLT) / (SMCREF - SMCWLT)
      IF (GX .GT. 1.) GX = 1.
      IF (GX .LT. 0.) GX = 0.
------------------------------------------------------------------------
USE SOIL DEPTH AS WEIGHTING FACTOR
------------------------------------------------------------------------
      PART(K) = ((ZSOIL(K)-ZSOIL(K-1))/ZSOIL(NROOT)) * GX
------------------------------------------------------------------------
USE ROOT DISTRIBUTION AS WEIGHTING FACTOR
       PART(K) = RTDIS(K) * GX
------------------------------------------------------------------------
    END DO

    DO K = 1,NROOT
      RCSOIL = RCSOIL+PART(K)
    END DO
    RCSOIL = MAX(RCSOIL,0.0001)
```

```
-----------------------------------------------------------------------
DETERMINE CANOPY RESISTANCE DUE TO ALL FACTORS.  CONVERT CANOPY
RESISTANCE (RC) TO PLANT COEFFICIENT (PC) TO BE USED WITH POTENTIAL
EVAP IN DETERMINING ACTUAL EVAP.  PC IS DETERMINED BY:
  PC * LINERIZED PENMAN POTENTIAL EVAP =
  PENMAN-MONTEITH ACTUAL EVAPORATION (CONTAINING RC TERM).
-----------------------------------------------------------------------
    RC = RSMIN/(XLAI*RCS*RCT*RCQ*RCSOIL)

     TAIR4 = SFCTMP**4.
     ST1 = (4.*SIGMA*RD)/CP
     SLVCP = SLV/CP
     RR = ST1*TAIR4/(SFCPRS*CH) + 1.0
    RR = (4.*SIGMA*RD/CP)*(SFCTMP**4.)/(SFCPRS*CH) + 1.0
    DELTA = (SLV/CP)*DQSDT2

    PC = (RR+DELTA)/(RR*(1.+RC*CH)+DELTA)

-----------------------------------------------------------------------
END SUBROUTINE CANRES
-----------------------------------------------------------------------
    RETURN
    END


    FUNCTION CSNOW (DSNOW)

    IMPLICIT NONE

-----------------------------------------------------------------------
FUNCTION CSNOW
-----------------------------------------------------------------------
CALCULATE SNOW TERMAL CONDUCTIVITY
-----------------------------------------------------------------------
    REAL C
    REAL DSNOW
    REAL CSNOW
    REAL UNIT

    PARAMETER(UNIT = 0.11631)

-----------------------------------------------------------------------
CSNOW IN UNITS OF CAL/(CM*HR*C), RETURNED IN W/(M*C)
BASIC VERSION IS DYACHKOVA EQUATION (1960), FOR RANGE 0.1-0.4
-----------------------------------------------------------------------
    C=0.328*10**(2.25*DSNOW)
    CSNOW=UNIT*C
```

```
-----------------------------------------------------------------------
DE VAUX EQUATION (1933), IN RANGE 0.1-0.6
-----------------------------------------------------------------------
     CSNOW=0.0293*(1.+100.*DSNOW**2)

-----------------------------------------------------------------------
E. ANDERSEN FROM FLERCHINGER
-----------------------------------------------------------------------
     CSNOW=0.021+2.51*DSNOW**2

-----------------------------------------------------------------------
END FUNCTION CSNOW
-----------------------------------------------------------------------
    RETURN
    END


    FUNCTION DEVAP (ETP1,SMC,ZSOIL,SHDFAC,SMCMAX,BEXP, &
              DKSAT,DWSAT,SMCDRY,SMCREF,SMCWLT,FXEXP)

    IMPLICIT NONE

-----------------------------------------------------------------------
FUNCTION DEVAP
-----------------------------------------------------------------------
CALCULATE DIRECT SOIL EVAPORATION
-----------------------------------------------------------------------
    REAL BEXP
    REAL DEVAP
    REAL DKSAT
    REAL DWSAT
    REAL ETP1
    REAL FX
    REAL FXEXP
    REAL SHDFAC
    REAL SMC
    REAL SMCDRY
    REAL SMCMAX
    REAL ZSOIL
    REAL SMCREF
    REAL SMCWLT

-----------------------------------------------------------------------
DIRECT EVAP A FUNCTION OF RELATIVE SOIL MOISTURE AVAILABILITY, LINEAR
WHEN FXEXP=1.
-----------------------------------------------------------------------
    FX = ( (SMC - SMCDRY) / (SMCMAX - SMCDRY) )**FXEXP
```

```
  ---------------------------------------------------------------------
  FX > 1 REPRESENTS DEMAND CONTROL
  FX < 1 REPRESENTS FLUX CONTROL
  ---------------------------------------------------------------------
      FX = MAX ( MIN ( FX, 1. ) ,0. )

  ---------------------------------------------------------------------
  ALLOW FOR THE DIRECT-EVAP-REDUCING EFFECT OF SHADE
  ---------------------------------------------------------------------
      DEVAP = FX * ( 1.0 - SHDFAC ) * ETP1

  ---------------------------------------------------------------------
  END FUNCTION DEVAP
  ---------------------------------------------------------------------
      RETURN
      END


CCCC 3. PHYSICS SUBROUTINE ==>  SUBROUTINE HRT  CCCCCCCCCCCCCCCCCCCCC

      SUBROUTINE HRT (RHSTS,STC,SMC,SMCMAX,NSOIL,ZSOIL,YY,ZZ1, &
                  TBOT,ZBOT,PSISAT,SH2O,DT,BEXP, &
                  F1,DF1,QUARTZ,CSOIL,AI,BI,CI)

      IMPLICIT NONE

  ---------------------------------------------------------------------
  SUBROUTINE HRT
  ---------------------------------------------------------------------
  CALCULATE THE RIGHT HAND SIDE OF THE TIME TENDENCY TERM OF THE SOIL
  THERMAL DIFFUSION EQUATION.  ALSO TO COMPUTE ( PREPARE ) THE MATRIX
  COEFFICIENTS FOR THE TRI-DIAGONAL MATRIX OF THE IMPLICIT TIME SCHEME.
  ---------------------------------------------------------------------
      INTEGER NSOLD
      PARAMETER(NSOLD = 20)

      LOGICAL*1 ITAVG

      INTEGER I
      INTEGER K
      INTEGER NSOIL

  ---------------------------------------------------------------------
  DECLARE WORK ARRAYS NEEDED IN TRI-DIAGONAL IMPLICIT SOLVER
  ---------------------------------------------------------------------
      REAL AI(NSOLD)
      REAL BI(NSOLD)
      REAL CI(NSOLD)
```

```
-----------------------------------------------------------------------
DECLARATIONS
-----------------------------------------------------------------------
    REAL BEXP
    REAL CAIR
    REAL CH2O
    REAL CICE
    REAL CSOIL
    REAL DDZ
    REAL DDZ2
    REAL DENOM
    REAL DF1
    REAL DF1N
    REAL DF1K
    REAL DT
    REAL DTSDZ
    REAL DTSDZ2
    REAL F1
    REAL HCPCT
    REAL PSISAT
    REAL QUARTZ
    REAL QTOT
    REAL RHSTS(NSOIL)
    REAL SSOIL
    REAL SICE
    REAL SMC(NSOIL)
    REAL SH2O(NSOIL)
    REAL SMCMAX
    REAL SNKSRC
    REAL STC(NSOIL)
    REAL TO
    REAL TAVG
    REAL TBK
    REAL TBK1
    REAL TBOT
    REAL ZBOT
    REAL TSNSR
    REAL TSURF
    REAL YY
    REAL ZSOIL(NSOIL)
    REAL ZZ1

    PARAMETER(TO = 273.15)

-----------------------------------------------------------------------
SET SPECIFIC HEAT CAPACITIES OF AIR, WATER, ICE, SOIL MINERAL
-----------------------------------------------------------------------
```

```
    PARAMETER(CAIR = 1004.0)
    PARAMETER(CH2O = 4.2E6)
    PARAMETER(CICE = 2.106E6)
NOTE: CSOIL NOW SET IN ROUTINE REDPRM AND PASSED IN
     PARAMETER(CSOIL = 1.26E6)

---------------------------------------------------------------------
INITIALIZE LOGICAL FOR SOIL LAYER TEMPERATURE AVERAGING.
---------------------------------------------------------------------
    ITAVG = .TRUE.
     ITAVG = .FALSE.

---------------------------------------------------------------------
BEGIN SECTION FOR TOP SOIL LAYER
---------------------------------------------------------------------
CALC THE HEAT CAPACITY OF THE TOP SOIL LAYER
---------------------------------------------------------------------
    HCPCT = SH2O(1)*CH2O + (1.0-SMCMAX)*CSOIL + (SMCMAX-SMC(1))*CAIR &
          + ( SMC(1) - SH2O(1) )*CICE

---------------------------------------------------------------------
CALC THE MATRIX COEFFICIENTS AI, BI, AND CI FOR THE TOP LAYER
---------------------------------------------------------------------
    DDZ = 1.0 / ( -0.5 * ZSOIL(2) )
    AI(1) = 0.0
    CI(1) = (DF1 * DDZ) / (ZSOIL(1) * HCPCT)
    BI(1) = -CI(1) + DF1 / (0.5 * ZSOIL(1) * ZSOIL(1)*HCPCT*ZZ1)

---------------------------------------------------------------------
CALCULATE THE VERTICAL SOIL TEMP GRADIENT BTWN THE 1ST AND 2ND SOIL
LAYERS.  THEN CALCULATE THE SUBSURFACE HEAT FLUX. USE THE TEMP
GRADIENT AND SUBSFC HEAT FLUX TO CALC "RIGHT-HAND SIDE TENDENCY
TERMS", OR "RHSTS", FOR TOP SOIL LAYER.
---------------------------------------------------------------------
    DTSDZ = (STC(1) - STC(2)) / (-0.5 * ZSOIL(2))
    SSOIL = DF1 * (STC(1) - YY) / (0.5 * ZSOIL(1) * ZZ1)
    RHSTS(1) = (DF1 * DTSDZ - SSOIL) / (ZSOIL(1) * HCPCT)

---------------------------------------------------------------------
NEXT CAPTURE THE VERTICAL DIFFERENCE OF THE HEAT FLUX AT TOP AND
BOTTOM OF FIRST SOIL LAYER FOR USE IN HEAT FLUX CONSTRAINT APPLIED TO
POTENTIAL SOIL FREEZING/THAWING IN ROUTINE SNKSRC.
---------------------------------------------------------------------
    QTOT = SSOIL - DF1*DTSDZ

---------------------------------------------------------------------
IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP):
SET TEMP "TSURF" AT TOP OF SOIL COLUMN (FOR USE IN FREEZING SOIL
```

```
PHYSICS LATER IN FUNCTION SUBROUTINE SNKSRC).  IF SNOWPACK CONTENT IS
ZERO, THEN TSURF EXPRESSION BELOW GIVES TSURF = SKIN TEMP.  IF
SNOWPACK IS NONZERO (HENCE ARGUMENT ZZ1=1), THEN TSURF EXPRESSION
BELOW YIELDS SOIL COLUMN TOP TEMPERATURE UNDER SNOWPACK.  THEN
CALCULATE TEMPERATURE AT BOTTOM INTERFACE OF 1ST SOIL LAYER FOR USE
LATER IN FUNCTION SUBROUTINE SNKSRC
-----------------------------------------------------------------------
    IF (ITAVG) THEN
      TSURF = (YY + (ZZ1-1) * STC(1)) / ZZ1
      CALL TBND (STC(1),STC(2),ZSOIL,ZBOT,1,NSOIL,TBK)
    ENDIF

-----------------------------------------------------------------------
CALCULATE FROZEN WATER CONTENT IN 1ST SOIL LAYER.
-----------------------------------------------------------------------
    SICE = SMC(1) - SH2O(1)

-----------------------------------------------------------------------
IF FROZEN WATER PRESENT OR ANY OF LAYER-1 MID-POINT OR BOUNDING
INTERFACE TEMPERATURES BELOW FREEZING, THEN CALL SNKSRC TO
COMPUTE HEAT SOURCE/SINK (AND CHANGE IN FROZEN WATER CONTENT)
DUE TO POSSIBLE SOIL WATER PHASE CHANGE
-----------------------------------------------------------------------
    IF ( (SICE   .GT. 0.) .OR. (TSURF .LT. T0) .OR. &
         (STC(1) .LT. T0) .OR. (TBK   .LT. T0) ) THEN

      IF (ITAVG) THEN
        CALL TMPAVG(TAVG,TSURF,STC(1),TBK,ZSOIL,NSOIL,1)
      ELSE
        TAVG = STC(1)
      ENDIF
      TSNSR = SNKSRC (TAVG,SMC(1),SH2O(1), &
       ZSOIL,NSOIL,SMCMAX,PSISAT,BEXP,DT,1,QTOT)

      RHSTS(1) = RHSTS(1) - TSNSR / ( ZSOIL(1) * HCPCT )
    ENDIF

-----------------------------------------------------------------------
THIS ENDS SECTION FOR TOP SOIL LAYER.
-----------------------------------------------------------------------
INITIALIZE DDZ2
-----------------------------------------------------------------------
    DDZ2 = 0.0

-----------------------------------------------------------------------
LOOP THRU THE REMAINING SOIL LAYERS, REPEATING THE ABOVE PROCESS
(EXCEPT SUBSFC OR "GROUND" HEAT FLUX NOT REPEATED IN LOWER LAYERS)
-----------------------------------------------------------------------
```

```
    DF1K = DF1
    DO K = 2,NSOIL

-----------------------------------------------------------------------
CALCULATE HEAT CAPACITY FOR THIS SOIL LAYER.
-----------------------------------------------------------------------
      HCPCT = SH2O(K)*CH2O +(1.0-SMCMAX)*CSOIL +(SMCMAX-SMC(K))*CAIR &
            + ( SMC(K) - SH2O(K) )*CICE

      IF (K .NE. NSOIL) THEN
-----------------------------------------------------------------------
THIS SECTION FOR LAYER 2 OR GREATER, BUT NOT LAST LAYER.
-----------------------------------------------------------------------
CALCULATE THERMAL DIFFUSIVITY FOR THIS LAYER.
-----------------------------------------------------------------------
        CALL TDFCND (DF1N,SMC(K),QUARTZ,SMCMAX,SH2O(K))

-----------------------------------------------------------------------
CALC THE VERTICAL SOIL TEMP GRADIENT THRU THIS LAYER
-----------------------------------------------------------------------
        DENOM = 0.5 * ( ZSOIL(K-1) - ZSOIL(K+1) )
        DTSDZ2 = ( STC(K) - STC(K+1) ) / DENOM

-----------------------------------------------------------------------
CALC THE MATRIX COEF, CI, AFTER CALC'NG ITS PARTIAL PRODUCT
-----------------------------------------------------------------------
        DDZ2 = 2. / (ZSOIL(K-1) - ZSOIL(K+1))
        CI(K) = -DF1N * DDZ2 / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)

-----------------------------------------------------------------------
IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP):  CALCULATE
TEMP AT BOTTOM OF LAYER.
-----------------------------------------------------------------------
        IF (ITAVG) THEN
          CALL TBND (STC(K),STC(K+1),ZSOIL,ZBOT,K,NSOIL,TBK1)
        ENDIF
      ELSE

-----------------------------------------------------------------------
SPECIAL CASE OF BOTTOM SOIL LAYER:  CALCULATE THERMAL DIFFUSIVITY FOR
BOTTOM LAYER.
-----------------------------------------------------------------------
        CALL TDFCND (DF1N,SMC(K),QUARTZ,SMCMAX,SH2O(K))

-----------------------------------------------------------------------
CALC THE VERTICAL SOIL TEMP GRADIENT THRU BOTTOM LAYER.
-----------------------------------------------------------------------
        DENOM = .5 * (ZSOIL(K-1) + ZSOIL(K)) - ZBOT
```

```
        DTSDZ2 = (STC(K)-TBOT) / DENOM

-----------------------------------------------------------------------
SET MATRIX COEF, CI TO ZERO IF BOTTOM LAYER.
-----------------------------------------------------------------------
        CI(K) = 0.

-----------------------------------------------------------------------
IF TEMPERATURE AVERAGING INVOKED (ITAVG=TRUE; ELSE SKIP):  CALCULATE
TEMP AT BOTTOM OF LAST LAYER.
-----------------------------------------------------------------------
        IF (ITAVG) THEN
          CALL TBND (STC(K),TBOT,ZSOIL,ZBOT,K,NSOIL,TBK1)
        ENDIF

      ENDIF
-----------------------------------------------------------------------
THIS ENDS SPECIAL LOOP FOR BOTTOM LAYER.
-----------------------------------------------------------------------
CALCULATE RHSTS FOR THIS LAYER AFTER CALC'NG A PARTIAL PRODUCT.
-----------------------------------------------------------------------
      DENOM = ( ZSOIL(K) - ZSOIL(K-1) ) * HCPCT
      RHSTS(K) = ( DF1N * DTSDZ2 - DF1K * DTSDZ ) / DENOM
      QTOT = -1.0*DENOM*RHSTS(K)
      SICE = SMC(K) - SH2O(K)

      IF ( (SICE .GT. 0.) .OR. (TBK .LT. T0) .OR. &
         (STC(K) .LT. T0) .OR. (TBK1 .LT. T0) ) THEN

        IF (ITAVG) THEN
          CALL TMPAVG(TAVG,TBK,STC(K),TBK1,ZSOIL,NSOIL,K)
        ELSE
          TAVG = STC(K)
        ENDIF
        TSNSR = SNKSRC(TAVG,SMC(K),SH2O(K),ZSOIL,NSOIL, &
                       SMCMAX,PSISAT,BEXP,DT,K,QTOT)
        RHSTS(K) = RHSTS(K) - TSNSR / DENOM
      ENDIF

-----------------------------------------------------------------------
CALC MATRIX COEFS, AI, AND BI FOR THIS LAYER.
-----------------------------------------------------------------------
      AI(K) = - DF1 * DDZ / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)
      BI(K) = -(AI(K) + CI(K))

-----------------------------------------------------------------------
RESET VALUES OF DF1, DTSDZ, DDZ, AND TBK FOR LOOP TO NEXT SOIL LAYER.
-----------------------------------------------------------------------
```

```
        TBK   = TBK1
        DF1K  = DF1N
        DTSDZ = DTSDZ2
        DDZ   = DDZ2
      END DO

  ---------------------------------------------------------------------
  END SUBROUTINE HRT
  ---------------------------------------------------------------------
      RETURN
      END

 CCCC 4. PHYSICS SUBROUTINE ==>  SUBROUTINE HRTICE  CCCCCCCCCCCCCCCCCCCC

      SUBROUTINE HRTICE (RHSTS,STC,NSOIL,ZSOIL,YY,ZZ1,DF1,AI,BI,CI)

      IMPLICIT NONE

  ---------------------------------------------------------------------
  SUBROUTINE HRTICE
  ---------------------------------------------------------------------
  CALCULATE THE RIGHT HAND SIDE OF THE TIME TENDENCY TERM OF THE SOIL
  THERMAL DIFFUSION EQUATION IN THE CASE OF SEA-ICE PACK.  ALSO TO
  COMPUTE (PREPARE) THE MATRIX COEFFICIENTS FOR THE TRI-DIAGONAL MATRIX
  OF THE IMPLICIT TIME SCHEME.
  ---------------------------------------------------------------------
      INTEGER NSOLD
      PARAMETER(NSOLD = 20)

      INTEGER K
      INTEGER NSOIL

      REAL AI(NSOLD)
      REAL BI(NSOLD)
      REAL CI(NSOLD)

      REAL DDZ
      REAL DDZ2
      REAL DENOM
      REAL DF1
      REAL DTSDZ
      REAL DTSDZ2
      REAL HCPCT
      REAL RHSTS(NSOIL)
      REAL SSOIL
      REAL STC(NSOIL)
      REAL TBOT
      REAL YY
```

```
    REAL ZBOT
    REAL ZSOIL(NSOIL)
    REAL ZZ1

    DATA TBOT /271.16/

-----------------------------------------------------------------------
SET A NOMINAL UNIVERSAL VALUE OF THE SEA-ICE SPECIFIC HEAT CAPACITY,
HCPCT = 1880.0*917.0.
-----------------------------------------------------------------------
    PARAMETER(HCPCT = 1.72396E+6)

-----------------------------------------------------------------------
THE INPUT ARGUMENT DF1 IS A UNIVERSALLY CONSTANT VALUE OF SEA-ICE
THERMAL DIFFUSIVITY, SET IN ROUTINE SNOPAC AS DF1 = 2.2.
-----------------------------------------------------------------------
SET ICE PACK DEPTH.  USE TBOT AS ICE PACK LOWER BOUNDARY TEMPERATURE
(THAT OF UNFROZEN SEA WATER AT BOTTOM OF SEA ICE PACK).  ASSUME ICE
PACK IS OF N=NSOIL LAYERS SPANNING A UNIFORM CONSTANT ICE PACK
THICKNESS AS DEFINED BY ZSOIL(NSOIL) IN ROUTINE SFLX.
-----------------------------------------------------------------------
    ZBOT = ZSOIL(NSOIL)

-----------------------------------------------------------------------
CALC THE MATRIX COEFFICIENTS AI, BI, AND CI FOR THE TOP LAYER
-----------------------------------------------------------------------
    DDZ = 1.0 / ( -0.5 * ZSOIL(2) )
    AI(1) = 0.0
    CI(1) = (DF1 * DDZ) / (ZSOIL(1) * HCPCT)
    BI(1) = -CI(1) + DF1/(0.5 * ZSOIL(1) * ZSOIL(1) * HCPCT * ZZ1)

-----------------------------------------------------------------------
CALC THE VERTICAL SOIL TEMP GRADIENT BTWN THE TOP AND 2ND SOIL LAYERS.
RECALC/ADJUST THE SOIL HEAT FLUX.  USE THE GRADIENT AND FLUX TO CALC
RHSTS FOR THE TOP SOIL LAYER.
-----------------------------------------------------------------------
    DTSDZ = ( STC(1) - STC(2) ) / ( -0.5 * ZSOIL(2) )
    SSOIL = DF1 * ( STC(1) - YY ) / ( 0.5 * ZSOIL(1) * ZZ1 )
    RHSTS(1) = ( DF1 * DTSDZ - SSOIL ) / ( ZSOIL(1) * HCPCT )

-----------------------------------------------------------------------
INITIALIZE DDZ2
-----------------------------------------------------------------------
    DDZ2 = 0.0

-----------------------------------------------------------------------
LOOP THRU THE REMAINING SOIL LAYERS, REPEATING THE ABOVE PROCESS
-----------------------------------------------------------------------
```

```
    DO K = 2,NSOIL
      IF (K .NE. NSOIL) THEN

-----------------------------------------------------------------------
CALC THE VERTICAL SOIL TEMP GRADIENT THRU THIS LAYER.
-----------------------------------------------------------------------
        DENOM = 0.5 * ( ZSOIL(K-1) - ZSOIL(K+1) )
        DTSDZ2 = ( STC(K) - STC(K+1) ) / DENOM

-----------------------------------------------------------------------
CALC THE MATRIX COEF, CI, AFTER CALC'NG ITS PARTIAL PRODUCT.
-----------------------------------------------------------------------
        DDZ2 = 2. / (ZSOIL(K-1) - ZSOIL(K+1))
        CI(K) = -DF1 * DDZ2 / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)
      ELSE

-----------------------------------------------------------------------
CALC THE VERTICAL SOIL TEMP GRADIENT THRU THE LOWEST LAYER.
-----------------------------------------------------------------------
        DTSDZ2 = (STC(K)-TBOT)/(.5 * (ZSOIL(K-1) + ZSOIL(K))-ZBOT)

-----------------------------------------------------------------------
SET MATRIX COEF, CI TO ZERO.
-----------------------------------------------------------------------
        CI(K) = 0.
      ENDIF

-----------------------------------------------------------------------
CALC RHSTS FOR THIS LAYER AFTER CALC'NG A PARTIAL PRODUCT.
-----------------------------------------------------------------------
      DENOM = ( ZSOIL(K) - ZSOIL(K-1) ) * HCPCT
      RHSTS(K) = ( DF1 * DTSDZ2 - DF1 * DTSDZ ) / DENOM

-----------------------------------------------------------------------
CALC MATRIX COEFS, AI, AND BI FOR THIS LAYER.
-----------------------------------------------------------------------
      AI(K) = - DF1 * DDZ / ((ZSOIL(K-1) - ZSOIL(K)) * HCPCT)
      BI(K) = -(AI(K) + CI(K))

-----------------------------------------------------------------------
RESET VALUES OF DTSDZ AND DDZ FOR LOOP TO NEXT SOIL LYR.
-----------------------------------------------------------------------
      DTSDZ = DTSDZ2
      DDZ   = DDZ2

    END DO
-----------------------------------------------------------------------
END SUBROUTINE HRTICE
```

```
  ----------------------------------------------------------------------
      RETURN
      END

 CCCC 5. PHYSICS SUBROUTINE ==>  SUBROUTINE HSTEP  CCCCCCCCCCCCCCCCCCCC

      SUBROUTINE HSTEP (STCOUT,STCIN,RHSTS,DT,NSOIL,AI,BI,CI)

      IMPLICIT NONE

  ----------------------------------------------------------------------
  SUBROUTINE HSTEP
  ----------------------------------------------------------------------
  CALCULATE/UPDATE THE SOIL TEMPERATURE FIELD.
  ----------------------------------------------------------------------
      INTEGER NSOLD
      PARAMETER(NSOLD = 20)

      INTEGER K
      INTEGER NSOIL

      REAL AI(NSOLD)
      REAL BI(NSOLD)
      REAL CI(NSOLD)
      REAL CIin(NSOLD)
      REAL DT
      REAL RHSTS(NSOIL)
      REAL RHSTSin(NSOIL)
      REAL STCIN(NSOIL)
      REAL STCOUT(NSOIL)

  ----------------------------------------------------------------------
  CREATE FINITE DIFFERENCE VALUES FOR USE IN ROSR12 ROUTINE
  ----------------------------------------------------------------------
      DO K = 1,NSOIL
        RHSTS(K) = RHSTS(K) * DT
        AI(K) = AI(K) * DT
        BI(K) = 1. + BI(K) * DT
        CI(K) = CI(K) * DT
      END DO

  ----------------------------------------------------------------------
  COPY VALUES FOR INPUT VARIABLES BEFORE CALL TO ROSR12
  ----------------------------------------------------------------------
      DO K = 1,NSOIL
         RHSTSin(K) = RHSTS(K)
      END DO
      DO K = 1,NSOLD
```

```
      CIin(K) = CI(K)
    END DO

 ---------------------------------------------------------------------
 SOLVE THE TRI-DIAGONAL MATRIX EQUATION
 ---------------------------------------------------------------------
    CALL ROSR12(CI,AI,BI,CIin,RHSTSin,RHSTS,NSOIL)

 ---------------------------------------------------------------------
 CALC/UPDATE THE SOIL TEMPS USING MATRIX SOLUTION
 ---------------------------------------------------------------------
    DO K = 1,NSOIL
      STCOUT(K) = STCIN(K) + CI(K)
    END DO

 ---------------------------------------------------------------------
 END SUBROUTINE HSTEP
 ---------------------------------------------------------------------
    RETURN
    END

CCCC 6. PHYSICS SUBROUTINE ==>  SUBROUTINE NOPAC  CCCCCCCCCCCCCCCCCCCC

    SUBROUTINE NOPAC(ETP,ETA,PRCP,SMC,SMCMAX,SMCWLT, &
                SMCREF,SMCDRY,CMC,CMCMAX,NSOIL,DT,SHDFAC, &
                SBETA,Q2,T1,SFCTMP,T24,TH2,FDOWN,F1,SSOIL, &
                STC,EPSCA,BEXP,PC,RCH,RR,CFACTR, &
                SH2O,SLOPE,KDT,FRZFACT,PSISAT,ZSOIL, &
                DKSAT,DWSAT,TBOT,ZBOT,RUNOFF1,RUNOFF2, &
                RUNOFF3,EDIR,EC,ET,ETT,NROOT,ICE,RTDIS, &
                QUARTZ,FXEXP,CSOIL, &
                BETA,DRIP,DEW,FLX1,FLX2,FLX3)

    IMPLICIT NONE

 ---------------------------------------------------------------------
 SUBROUTINE NOPAC
 ---------------------------------------------------------------------
 CALCULATE SOIL MOISTURE AND HEAT FLUX VALUES AND UPDATE SOIL MOISTURE
 CONTENT AND SOIL HEAT CONTENT VALUES FOR THE CASE WHEN NO SNOW PACK IS
 PRESENT.
 ---------------------------------------------------------------------
    INTEGER ICE
    INTEGER NROOT
    INTEGER NSOIL

    REAL BEXP
    REAL BETA
```

```
REAL CFACTR
REAL CMC
REAL CMCMAX
REAL CP
REAL CSOIL
REAL DEW
REAL DF1
REAL DKSAT
REAL DRIP
REAL DT
REAL DWSAT
REAL EC
REAL EDIR
REAL EPSCA
REAL ETA
REAL ETA1
REAL ETP
REAL ETP1
REAL ET(NSOIL)
REAL ETT
REAL FDOWN
REAL F1
REAL FXEXP
REAL FLX1
REAL FLX2
REAL FLX3
REAL FRZFACT
REAL KDT
REAL PC
REAL PRCP
REAL PRCP1
REAL PSISAT
REAL Q2
REAL QUARTZ
REAL RCH
REAL RR
REAL RTDIS(NSOIL)
REAL RUNOFF1
REAL RUNOFF2
REAL RUNOFF3
REAL SSOIL
REAL SBETA
REAL SFCTMP
REAL SHDFAC
REAL SH2O(NSOIL)
REAL SIGMA
REAL SLOPE
REAL SMC(NSOIL)
```

```
    REAL SMCDRY
    REAL SMCMAX
    REAL SMCREF
    REAL SMCWLT
    REAL STC(NSOIL)
    REAL T1
    REAL T24
    REAL TBOT
    REAL TH2
    REAL YY
    REAL YYNUM
    REAL ZBOT
    REAL ZSOIL(NSOIL)
    REAL ZZ1

    PARAMETER(CP = 1004.5)
    PARAMETER(SIGMA = 5.67E-8)

-----------------------------------------------------------------------
EXECUTABLE CODE BEGINS HERE:
CONVERT ETP FROM KG M-2 S-1 TO MS-1 AND INITIALIZE DEW.
-----------------------------------------------------------------------
    PRCP1 = PRCP * 0.001
    ETP1 = ETP * 0.001
    DEW = 0.0

    IF (ETP .GT. 0.0) THEN

-----------------------------------------------------------------------
CONVERT PRCP FROM 'KG M-2 S-1' TO 'M S-1'.
-----------------------------------------------------------------------
         CALL SMFLX (ETA1,SMC,NSOIL,CMC,ETP1,DT,PRCP1,ZSOIL, &
               SH2O,SLOPE,KDT,FRZFACT, &
               SMCMAX,BEXP,PC,SMCWLT,DKSAT,DWSAT, &
               SMCREF,SHDFAC,CMCMAX, &
               SMCDRY,CFACTR,RUNOFF1,RUNOFF2,RUNOFF3, &
               EDIR,EC,ET,ETT,SFCTMP,Q2,NROOT,RTDIS,FXEXP, &
               DRIP)

-----------------------------------------------------------------------
      CONVERT MODELED EVAPOTRANSPIRATION FM  M S-1  TO  KG M-2 S-1
-----------------------------------------------------------------------
      ETA = ETA1 * 1000.0

    ELSE

-----------------------------------------------------------------------
IF ETP < 0, ASSUME DEW FORMS (TRANSFORM ETP1 INTO DEW AND REINITIALIZE
```

```
ETP1 TO ZERO).
-----------------------------------------------------------------------
      DEW = -ETP1
      ETP1 = 0.0

-----------------------------------------------------------------------
CONVERT PRCP FROM 'KG M-2 S-1' TO 'M S-1' AND ADD DEW AMOUNT.
-----------------------------------------------------------------------
      PRCP1 = PRCP1 + DEW

    CALL SMFLX (ETA1,SMC,NSOIL,CMC,ETP1,DT,PRCP1,ZSOIL, &
          SH2O,SLOPE,KDT,FRZFACT, &
          SMCMAX,BEXP,PC,SMCWLT,DKSAT,DWSAT, &
          SMCREF,SHDFAC,CMCMAX, &
          SMCDRY,CFACTR,RUNOFF1,RUNOFF2,RUNOFF3, &
          EDIR,EC,ET,ETT,SFCTMP,Q2,NROOT,RTDIS,FXEXP, &
          DRIP)

-----------------------------------------------------------------------
CONVERT MODELED EVAPOTRANSPIRATION FROM 'M S-1' TO 'KG M-2 S-1'.
-----------------------------------------------------------------------
      ETA = ETA1 * 1000.0

    ENDIF

-----------------------------------------------------------------------
BASED ON ETP AND E VALUES, DETERMINE BETA
-----------------------------------------------------------------------
    IF ( ETP .LE. 0.0 ) THEN
      BETA = 0.0
      IF ( ETP .LT. 0.0 ) THEN
        BETA = 1.0
        ETA = ETP
      ENDIF
    ELSE
      BETA = ETA / ETP
    ENDIF

-----------------------------------------------------------------------
GET SOIL THERMAL DIFFUXIVITY/CONDUCTIVITY FOR TOP SOIL LYR,
CALC. ADJUSTED TOP LYR SOIL TEMP AND ADJUSTED SOIL FLUX, THEN
CALL SHFLX TO COMPUTE/UPDATE SOIL HEAT FLUX AND SOIL TEMPS.
-----------------------------------------------------------------------
    CALL TDFCND (DF1,SMC(1),QUARTZ,SMCMAX,SH2O(1))

-----------------------------------------------------------------------
VEGETATION GREENNESS FRACTION REDUCTION IN SUBSURFACE HEAT FLUX
VIA REDUCTION FACTOR, WHICH IS CONVENIENT TO APPLY HERE TO THERMAL
```

```
DIFFUSIVITY THAT IS LATER USED IN HRT TO COMPUTE SUB SFC HEAT FLUX
(SEE ADDITIONAL COMMENTS ON VEG EFFECT SUB-SFC HEAT FLX IN
ROUTINE SFLX)
-----------------------------------------------------------------------
    DF1 = DF1 * EXP(SBETA*SHDFAC)

-----------------------------------------------------------------------
COMPUTE INTERMEDIATE TERMS PASSED TO ROUTINE HRT (VIA ROUTINE
SHFLX BELOW) FOR USE IN COMPUTING SUBSURFACE HEAT FLUX IN HRT
-----------------------------------------------------------------------
    YYNUM = FDOWN - SIGMA * T24
    YY = SFCTMP + (YYNUM/RCH+TH2-SFCTMP-BETA*EPSCA) / RR
    ZZ1 = DF1 / ( -0.5 * ZSOIL(1) * RCH * RR ) + 1.0

    CALL SHFLX (SSOIL,STC,SMC,SMCMAX,NSOIL,T1,DT,YY,ZZ1,ZSOIL, &
                TBOT,ZBOT,SMCWLT,PSISAT,SH2O,BEXP,F1,DF1,ICE, &
                QUARTZ,CSOIL)

-----------------------------------------------------------------------
SET FLX1 AND FLX3 (SNOPACK PHASE CHANGE HEAT FLUXES) TO ZERO SINCE
THEY ARE NOT USED HERE IN SNOPAC.  FLX2 (FREEZING RAIN HEAT FLUX) WAS
SIMILARLY INITIALIZED IN THE PENMAN ROUTINE.
-----------------------------------------------------------------------
    FLX1 = 0.0
    FLX3 = 0.0

-----------------------------------------------------------------------
END SUBROUTINE NOPAC
-----------------------------------------------------------------------
    RETURN
    END

CCCC 7. PHYSICS SUBROUTINE ==>  SUBROUTINE PENMAN  CCCCCCCCCCCCCCCCCCCC

    SUBROUTINE PENMAN (SFCTMP,SFCPRS,CH,T2V,TH2,PRCP,FDOWN,T24,SSOIL, &
                       Q2,Q2SAT,ETP,RCH,EPSCA,RR,SNOWNG,FRZGRA, &
                       DQSDT2,FLX2)

    IMPLICIT NONE

-----------------------------------------------------------------------
SUBROUTINE PENMAN
-----------------------------------------------------------------------
CALCULATE POTENTIAL EVAPORATION FOR THE CURRENT POINT.  VARIOUS
PARTIAL SUMS/PRODUCTS ARE ALSO CALCULATED AND PASSED BACK TO THE
CALLING ROUTINE FOR LATER USE.
-----------------------------------------------------------------------
    LOGICAL*1 SNOWNG
```

```
      LOGICAL*1 FRZGRA

      REAL A
      REAL BETA
      REAL CH
      REAL CP
      REAL CPH2O
      REAL CPICE
      REAL DELTA
      REAL DQSDT2
      REAL ELCP
      REAL EPSCA
      REAL ETP
      REAL FDOWN
      REAL FLX2
      REAL FNET
      REAL LSUBC
      REAL LSUBF
      REAL PRCP
      REAL Q2
      REAL Q2SAT
      REAL R
      REAL RAD
      REAL RCH
      REAL RHO
      REAL RR
      REAL SSOIL
      REAL SFCPRS
      REAL SFCTMP
      REAL SIGMA
      REAL T24
      REAL T2V
      REAL TH2

      PARAMETER(CP = 1004.6)
      PARAMETER(CPH2O = 4.218E+3)
      PARAMETER(CPICE = 2.106E+3)
      PARAMETER(R = 287.04)
      PARAMETER(ELCP = 2.4888E+3)
      PARAMETER(LSUBF = 3.335E+5)
      PARAMETER(LSUBC = 2.501000E+6)
      PARAMETER(SIGMA = 5.67E-8)

-----------------------------------------------------------------------
EXECUTABLE CODE BEGINS HERE:
-----------------------------------------------------------------------
      FLX2 = 0.0
```

```
  ----------------------------------------------------------------------
  PREPARE PARTIAL QUANTITIES FOR PENMAN EQUATION.
  ----------------------------------------------------------------------
      DELTA = ELCP * DQSDT2
      T24 = SFCTMP * SFCTMP * SFCTMP * SFCTMP
      RR = T24 * 6.48E-8 /(SFCPRS * CH) + 1.0
      RHO = SFCPRS / (R * T2V)
      RCH = RHO * CP * CH

  ----------------------------------------------------------------------
  ADJUST THE PARTIAL SUMS / PRODUCTS WITH THE LATENT HEAT
  EFFECTS CAUSED BY FALLING PRECIPITATION.
  ----------------------------------------------------------------------
      IF (.NOT. SNOWNG) THEN
        IF (PRCP .GT. 0.0) RR = RR + CPH2O*PRCP/RCH
      ELSE
        RR = RR + CPICE*PRCP/RCH
      ENDIF

      FNET = FDOWN - SIGMA*T24 - SSOIL

  ----------------------------------------------------------------------
  INCLUDE THE LATENT HEAT EFFECTS OF FRZNG RAIN CONVERTING TO ICE ON
  IMPACT IN THE CALCULATION OF FLX2 AND FNET.
  ----------------------------------------------------------------------
      IF (FRZGRA) THEN
        FLX2 = -LSUBF * PRCP
        FNET = FNET - FLX2
      ENDIF

  ----------------------------------------------------------------------
  FINISH PENMAN EQUATION CALCULATIONS.
  ----------------------------------------------------------------------
      RAD = FNET/RCH + TH2 - SFCTMP
      A = ELCP * (Q2SAT - Q2)
      EPSCA = (A*RR + RAD*DELTA) / (DELTA + RR)
      ETP = EPSCA * RCH / LSUBC

  ----------------------------------------------------------------------
  END SUBROUTINE PENMAN
  ----------------------------------------------------------------------
      RETURN
      END

CCCC 8. PHYSICS SUBROUTINE ==>  SUBROUTINE ROSR12  CCCCCCCCCCCCCCCCCCCC

      SUBROUTINE ROSR12 (P,A,B,C,D,DELTA,NSOIL)
```

```
    IMPLICIT NONE

-----------------------------------------------------------------------
SUBROUTINE ROSR12
-----------------------------------------------------------------------
INVERT (SOLVE) THE TRI-DIAGONAL MATRIX PROBLEM SHOWN BELOW:
###                                          ### ###  ###   ###  ###
#B(1), C(1),  0  ,  0  ,  0  ,   . . .  ,    0   # #      #   #      #
#A(2), B(2), C(2),  0  ,  0  ,   . . .  ,    0   # #      #   #      #
# 0  , A(3), B(3), C(3),  0  ,   . . .  ,    0   # #      #   # D(3) #
# 0  ,  0  , A(4), B(4), C(4),   . . .  ,    0   # # P(4) #   # D(4) #
# 0  ,  0  ,  0  , A(5), B(5),   . . .  ,    0   # # P(5) #   # D(5) #
# .                                          .   # #  .   # = #   .  #
# .                                          .   # #  .   #   #   .  #
# .                                          .   # #  .   #   #   .  #
# 0  , . . . , 0 , A(M-2), B(M-2), C(M-2),   0   # #P(M-2)#   #D(M-2)#
# 0  , . . . , 0 ,   0   , A(M-1), B(M-1), C(M-1)# #P(M-1)#   #D(M-1)#
# 0  , . . . , 0 ,   0   ,   0   ,  A(M) ,  B(M) # # P(M) #   # D(M) #
###                                          ### ###  ###   ###  ###
-----------------------------------------------------------------------
    INTEGER K
    INTEGER KK
    INTEGER NSOIL

    REAL A(NSOIL)
    REAL B(NSOIL)
    REAL C(NSOIL)
    REAL D(NSOIL)
    REAL DELTA(NSOIL)
    REAL P(NSOIL)

-----------------------------------------------------------------------
INITIALIZE EQN COEF C FOR THE LOWEST SOIL LAYER
-----------------------------------------------------------------------
    C(NSOIL) = 0.0

-----------------------------------------------------------------------
SOLVE THE COEFS FOR THE 1ST SOIL LAYER
-----------------------------------------------------------------------
    P(1) = -C(1) / B(1)
    DELTA(1) = D(1) / B(1)

-----------------------------------------------------------------------
SOLVE THE COEFS FOR SOIL LAYERS 2 THRU NSOIL
-----------------------------------------------------------------------
    DO K = 2,NSOIL
      P(K) = -C(K) * ( 1.0 / (B(K) + A (K) * P(K-1)) )
      DELTA(K) = (D(K)-A(K)*DELTA(K-1))*(1.0/(B(K)+A(K)*P(K-1)))
```

```
      END DO

  ---------------------------------------------------------------------
  SET P TO DELTA FOR LOWEST SOIL LAYER
  ---------------------------------------------------------------------
      P(NSOIL) = DELTA(NSOIL)

  ---------------------------------------------------------------------
  ADJUST P FOR SOIL LAYERS 2 THRU NSOIL
  ---------------------------------------------------------------------
      DO K = 2,NSOIL
         KK = NSOIL - K + 1
         P(KK) = P(KK) * P(KK+1) + DELTA(KK)
      END DO

  ---------------------------------------------------------------------
  END SUBROUTINE ROSR12
  ---------------------------------------------------------------------
      RETURN
      END

CCCC 9. PHYSICS SUBROUTINE ==>  SUBROUTINE SFCDIF  CCCCCCCCCCCCCCCCCCCC

      SUBROUTINE SFCDIF (ZLM,Z0,THZ0,THLM,SFCSPD,CZIL,AKMS,AKHS)

      IMPLICIT NONE

  ---------------------------------------------------------------------
  SUBROUTINE SFCDIF
  ---------------------------------------------------------------------
  CALCULATE SURFACE LAYER EXCHANGE COEFFICIENTS VIA ITERATIVE PROCESS.
  SEE CHEN ET AL (1997, BLM)
  ---------------------------------------------------------------------

      REAL WWST, WWST2, G, VKRM, EXCM, BETA, BTG, ELFC, WOLD, WNEW
      REAL PIHF, EPSU2, EPSUST, EPSIT, EPSA, ZTMIN, ZTMAX, HPBL, SQVISC
      REAL RIC, RRIC, FHNEU, RFC, RFAC, ZZ, PSLMU, PSLMS, PSLHU, PSLHS
      REAL XX, PSPMU, YY, PSPMS, PSPHU, PSPHS, ZLM, Z0, THZ0, THLM
      REAL SFCSPD, CZIL, AKMS, AKHS, ZILFC, ZU, ZT, RDZ, CXCH
      REAL DTHV, DU2, BTGH, WSTAR2, USTAR, ZSLU, ZSLT, RLOGU, RLOGT
      REAL RLMO, ZETALT, ZETALU, ZETAU, ZETAT, XLU4, XLT4, XU4, XT4
      REAL XLU, XLT, XU, XT, PSMZ, SIMM, PSHZ, SIMH, USTARK, RLMN, RLMA
CC    ......REAL ZTFC

      INTEGER ITRMX, ILECH, ITR

      PARAMETER &
        (WWST=1.2,WWST2=WWST*WWST,G=9.8,VKRM=0.40,EXCM=0.001 &
```

```
      ,BETA=1./270.,BTG=BETA*G,ELFC=VKRM*BTG &
      ,WOLD=.15,WNEW=1.-WOLD,ITRMX=05,PIHF=3.14159265/2.)
-----------------------------------------------------------------------
    PARAMETER &
      (EPSU2=1.E-4,EPSUST=0.07,EPSIT=1.E-4,EPSA=1.E-8 &
      ,ZTMIN=-5.,ZTMAX=1.,HPBL=1000.0 &
      ,SQVISC=258.2)
-----------------------------------------------------------------------
    PARAMETER &
      (RIC=0.183,RRIC=1.0/RIC,FHNEU=0.8,RFC=0.191 &
      ,RFAC=RIC/(FHNEU*RFC*RFC))

-----------------------------------------------------------------------
NOTE: THE TWO CODE BLOCKS BELOW DEFINE FUNCTIONS
-----------------------------------------------------------------------
LECH'S SURFACE FUNCTIONS
-----------------------------------------------------------------------
    PSLMU(ZZ)=-0.96*log(1.0-4.5*ZZ)
    PSLMS(ZZ)=ZZ*RRIC-2.076*(1.-1./(ZZ+1.))
    PSLHU(ZZ)=-0.96*log(1.0-4.5*ZZ)
    PSLHS(ZZ)=ZZ*RFAC-2.076*(1.-1./(ZZ+1.))

-----------------------------------------------------------------------
PAULSON'S SURFACE FUNCTIONS
-----------------------------------------------------------------------
    PSPMU(XX)=-2.*log((XX+1.)*0.5)-log((XX*XX+1.)*0.5)+2.*ATAN(XX) &
              -PIHF
    PSPMS(YY)=5.*YY
    PSPHU(XX)=-2.*log((XX*XX+1.)*0.5)
    PSPHS(YY)=5.*YY

-----------------------------------------------------------------------
THIS ROUTINE SFCDIF CAN HANDLE BOTH OVER OPEN WATER (SEA, OCEAN) AND
OVER SOLID SURFACE (LAND, SEA-ICE).
-----------------------------------------------------------------------
    ILECH=0

-----------------------------------------------------------------------
    ZTFC: RATIO OF ZOH/ZOM  LESS OR EQUAL THAN 1
    C......ZTFC=0.1
    CZIL: CONSTANT C IN Zilitinkevich, S. S.1995,:NOTE ABOUT ZT
-----------------------------------------------------------------------
    ZILFC=-CZIL*VKRM*SQVISC

-----------------------------------------------------------------------
    ZU=Z0
    C.......ZT=Z0*ZTFC
    RDZ=1./ZLM
```

```
      CXCH=EXCM*RDZ
      DTHV=THLM-THZO
      DU2=MAX(SFCSPD*SFCSPD,EPSU2)

  ---------------------------------------------------------------------
  BELJARS CORRECTION OF USTAR
  ---------------------------------------------------------------------
      BTGH=BTG*HPBL
cc    If statements to avoid TANGENT LINEAR problems near zero
      IF (BTGH*AKHS*DTHV .NE. 0.0) THEN
         WSTAR2=WWST2*ABS(BTGH*AKHS*DTHV)**(2./3.)
      ELSE
         WSTAR2=0.0
      ENDIF
      USTAR=MAX(SQRT(AKMS*SQRT(DU2+WSTAR2)),EPSUST)

  ---------------------------------------------------------------------
  ZILITINKEVITCH APPROACH FOR ZT
  ---------------------------------------------------------------------
      ZT=EXP(ZILFC*SQRT(USTAR*Z0))*Z0

  ---------------------------------------------------------------------
      ZSLU=ZLM+ZU
      ZSLT=ZLM+ZT
      PRINT*,'ZSLT=',ZSLT
      PRINT*,'ZLM=',ZLM
      PRINT*,'ZT=',ZT

      RLOGU=log(ZSLU/ZU)
      RLOGT=log(ZSLT/ZT)

      RLMO=ELFC*AKHS*DTHV/USTAR**3
      PRINT*,'RLMO=',RLMO
      PRINT*,'ELFC=',ELFC
      PRINT*,'AKHS=',AKHS
      PRINT*,'DTHV=',DTHV
      PRINT*,'USTAR=',USTAR

      DO ITR=1,ITRMX
  ---------------------------------------------------------------------
  1./MONIN-OBUKKHOV LENGTH-SCALE
  ---------------------------------------------------------------------
         ZETALT=MAX(ZSLT*RLMO,ZTMIN)
         RLMO=ZETALT/ZSLT
         ZETALU=ZSLU*RLMO
         ZETAU=ZU*RLMO
         ZETAT=ZT*RLMO
```

```
         IF(ILECH.EQ.0) THEN
            IF(RLMO.LT.0.)THEN
               XLU4=1.-16.*ZETALU
               XLT4=1.-16.*ZETALT
               XU4 =1.-16.*ZETAU
               XT4 =1.-16.*ZETAT

               XLU=SQRT(SQRT(XLU4))
               XLT=SQRT(SQRT(XLT4))
               XU =SQRT(SQRT(XU4))
               XT =SQRT(SQRT(XT4))

               PSMZ=PSPMU(XU)
     PRINT*,'-----------1------------'
     PRINT*,'PSMZ=',PSMZ
     PRINT*,'PSPMU(ZETAU)=',PSPMU(ZETAU)
     PRINT*,'XU=',XU
     PRINT*,'------------------------'
               SIMM=PSPMU(XLU)-PSMZ+RLOGU
               PSHZ=PSPHU(XT)
               SIMH=PSPHU(XLT)-PSHZ+RLOGT
            ELSE
               ZETALU=MIN(ZETALU,ZTMAX)
               ZETALT=MIN(ZETALT,ZTMAX)
               PSMZ=PSPMS(ZETAU)
     PRINT*,'-----------2------------'
     PRINT*,'PSMZ=',PSMZ
     PRINT*,'PSPMS(ZETAU)=',PSPMS(ZETAU)
     PRINT*,'ZETAU=',ZETAU
     PRINT*,'------------------------'
               SIMM=PSPMS(ZETALU)-PSMZ+RLOGU
               PSHZ=PSPHS(ZETAT)
               SIMH=PSPHS(ZETALT)-PSHZ+RLOGT
            ENDIF
         ELSE
-----------------------------------------------------------------------
LECH'S FUNCTIONS
-----------------------------------------------------------------------
            IF(RLMO.LT.0.)THEN
               PSMZ=PSLMU(ZETAU)
     PRINT*,'-----------3------------'
     PRINT*,'PSMZ=',PSMZ
     PRINT*,'PSLMU(ZETAU)=',PSLMU(ZETAU)
     PRINT*,'ZETAU=',ZETAU
     PRINT*,'------------------------'
               SIMM=PSLMU(ZETALU)-PSMZ+RLOGU
               PSHZ=PSLHU(ZETAT)
               SIMH=PSLHU(ZETALT)-PSHZ+RLOGT
```

```
            ELSE
               ZETALU=MIN(ZETALU,ZTMAX)
               ZETALT=MIN(ZETALT,ZTMAX)

               PSMZ=PSLMS(ZETAU)
     PRINT*,'-----------4------------'
     PRINT*,'PSMZ=',PSMZ
     PRINT*,'PSLMS(ZETAU)=',PSLMS(ZETAU)
     PRINT*,'ZETAU=',ZETAU
     PRINT*,'------------------------'
               SIMM=PSLMS(ZETALU)-PSMZ+RLOGU
               PSHZ=PSLHS(ZETAT)
               SIMH=PSLHS(ZETALT)-PSHZ+RLOGT
            ENDIF
         ENDIF
 ---------------------------------------------------------------------
 BELJAARS CORRECTION FOR USTAR
 ---------------------------------------------------------------------
         USTAR=MAX(SQRT(AKMS*SQRT(DU2+WSTAR2)),EPSUST)

 ---------------------------------------------------------------------
 ZILITINKEVITCH FIX FOR ZT
 ---------------------------------------------------------------------
         ZT=EXP(ZILFC*SQRT(USTAR*Z0))*Z0

         ZSLT=ZLM+ZT
         RLOGT=log(ZSLT/ZT)
---------------------------------------------------------------------
         USTARK=USTAR*VKRM
         AKMS=MAX(USTARK/SIMM,CXCH)
         AKHS=MAX(USTARK/SIMH,CXCH)
---------------------------------------------------------------------
 IF STATEMENTS TO AVOID TANGENT LINEAR PROBLEMS NEAR ZERO
---------------------------------------------------------------------
         IF (BTGH*AKHS*DTHV .NE. 0.0) THEN
            WSTAR2=WWST2*ABS(BTGH*AKHS*DTHV)**(2./3.)
         ELSE
            WSTAR2=0.0
         ENDIF
         RLMN=ELFC*AKHS*DTHV/USTAR**3
---------------------------------------------------------------------
         RLMA=RLMO*WOLD+RLMN*WNEW
---------------------------------------------------------------------
     IF(ABS((RLMN-RLMO)/RLMA).LT.EPSIT)    GO TO 110
---------------------------------------------------------------------
         RLMO=RLMA
---------------------------------------------------------------------
     END DO
```

```
      PRINT*,'----------------------------'
      PRINT*,'SFCDIF OUTPUT !  ! ! ! ! ! ! ! !  !   !    !'
      PRINT*,'ZLM=',ZLM
      PRINT*,'Z0=',Z0
      PRINT*,'THZ0=',THZ0
      PRINT*,'THLM=',THLM
      PRINT*,'SFCSPD=',SFCSPD
      PRINT*,'CZIL=',CZIL
      PRINT*,'AKMS=',AKMS
      PRINT*,'AKHS=',AKHS
      PRINT*,'----------------------------'

  ---------------------------------------------------------------------
  END SUBROUTINE SFCDIF
  ---------------------------------------------------------------------
      RETURN
      END

 CCCC 10. PHYSICS SUBROUTINE ==>  SUBROUTINE SFLX  CCCCCCCCCCCCCCCCCCCCC
```

INTERFACE:

```
      SUBROUTINE SFLX (ntl, &
      ICE,DT,ZLVL,NSOIL,SLDPTH, &
      LWDN,SOLDN,SFCPRS,PRCP,SFCTMP,Q2,SFCSPD, &
      TH2,Q2SAT,DQSDT2, &
      SLOPE,SHDFAC,SHDMIN,PTU,ALB,SNOALB, &
      RSMIN,RGL,HS,SNUP,Z0,XLAI,NROOT, &
      PSISAT,BEXP,DKSAT,SMCMAX,QUARTZ,DWSAT, &
      SMCWLT,SMCREF,SMCDRY,F1,KDT,FRZX,FRZFACT,TBOT, &
      CMC,T1,STC,SMC,SH2O,SNOWH,SNEQV,ALBEDO,CH,CM, &
      EVP,ETA,SHEAT, &
      EC,EDIR,ET,ETT,ESNOW,DRIP,DEW, &
      BETA,ETP,SSOIL, &
      FLX1,FLX2,FLX3, &
      SNOMLT,SNCOVR, &
      RUNOFF1,RUNOFF2,RUNOFF3, &
      RC,PC,RCS,RCT,RCQ,RCSOIL, &
      SOILW,SOILT,SOILM)
```

1.75.2 noahrst.F90 (Source File: noahrst.F90)

This program reads restart files for Noah. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
    1  Oct 1999: Jared Entin; Initial code
    15 Oct 1999: Paul Houser; Significant F90 Revision
    05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA
                 initial conditions if necessary.  This is controlled with
                 local variable NOAAIC.  Normally set to 0 in this subroutine
                 but set to 1 if want to use Dag's NOAA IC's.  Changed output
                 directory structure, and commented out if-then check so that
                 directory is always made.
    28 Apr 2002: Kristi Arsenault; Added NOAH LSM into LDAS
    28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
                  and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
   RESTART FILE FORMAT(fortran sequential binary):
    YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
    TILE(NCH)%COL        !Grid Col of Tile
    TILE(NCH)%ROW        !Grid Row of Tile
    TILE(NCH)%FGRD       !Fraction of Grid covered by tile
    TILE(NCH)%VEGT       !Vegetation Type of Tile
    NOAH(NCH)%STATES     !Model States in Tile Space
```

INTERFACE:

```
 #include "misc.h"
 subroutine noahrst
```

USES:

```
   use lisdrv_module, only : lis, grid, tile
   use noah_varder, only : noahdrv
   USE noah_varder      ! NOAH tile variables
   use time_manager
   use tile_spmdMod
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Read Active Archive File
!-----------------------------------------------------------------------
  print*,'DBG: noahrst -- in noahrst',' (',iam,')'
  if(masterproc) then
     IF(LIS%O%STARTCODE.EQ.1)THEN
        allocate(tmptile(lis%d%nch))
        OPEN(40,FILE=noahdrv%NOAH_RFILE,FORM='unformatted')

        call timemgr_read_restart(40)
        !call timemgr_restart()
        WRITE(*,*)'NOAH Restart File Used: ',noahdrv%NOAH_RFILE
        READ(40) VCLASS,NC,NR,NCH  !Time, veg class, no. tiles
!-----------------------------------------------------------------------
!   Check for Vegetation Class Conflict
```

```
!-----------------------------------------------------------------------
        IF(VCLASS.NE.LIS%P%VCLASS)THEN
           WRITE(*,*)noahdrv%NOAH_RFILE,' Vegetation class conflict'
           call endrun
        ENDIF
!-----------------------------------------------------------------------
!   Check for Grid Space Conflict
!-----------------------------------------------------------------------
        IF(NC.NE.LIS%D%LNC.OR.NR.NE.LIS%D%LNR)THEN
           WRITE(*,*)noahdrv%NOAH_RFILE,'Grid space mismatch - NOAH HALTED'
           call endrun
        ENDIF
!-----------------------------------------------------------------------
! Transfer Restart tile space to LIS tile space
!-----------------------------------------------------------------------
        IF(NCH.NE.LIS%D%NCH)THEN
           WRITE(*,*)'Restart Tile Space Mismatch, Halting..'
           call endrun
        endif
        READ(40) noah%T1         !NOAH Skin Temperature (K)
        READ(40) noah%CMC        !NOAH Canopy Water Content
        READ(40) noah%SNOWH      !NOAH Actual Snow Depth (m)
        READ(40) noah%SNEQV      !NOAH Water Equivalent Snow Depth (m)
        DO L=1,4
           READ(40) TMPTILE !NOAH Soil Layer Temp (4 layers)
           noah%STC(L)=TMPTILE
        ENDDO
        DO L=1,4
           READ(40) TMPTILE !NOAH Total soil moist. (4 layers)
           noah%SMC(L)=TMPTILE
        ENDDO
        DO L=1,4
           READ(40) TMPTILE !NOAH Liquid-only soil moist. (4 layers)
           noah%SH2O(L)=TMPTILE
        ENDDO
        READ(40) noah%CH         !NOAH Sfc Exchange Coef. for Heat/Moisture
        READ(40) noah%CM         !NOAH Sfc Exchange Coef. for Momentum
        close(40)
        deallocate(tmptile)
     endif
  endif
#if ( defined SPMD )
  if ( ( lis%o%startcode == 1 ) .and. ( npes >  1 ) ) then
     call noah_scatter()
  endif
#endif
```

1.75.3 noah_scatter.F90 (Source File: noah_scatter.F90)

Distributes noah tiles on to compute nodes

REVISION HISTORY:

```
Apr 2003 ; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine noah_scatter()
```

USES:

```
use tile_spmdMod
use noah_varder
use noahpardef_module
```

CONTENTS:

```
#if (defined SPMD)
  call MPI_SCATTERV(noah,di_array,displs, &
       MPI_NOAH_STRUCT,noah,di_array(iam),MPI_NOAH_STRUCT, &
       0,MPI_COMM_WORLD,ierr)
#endif
```

1.75.4 set_mxsnalb (Source File: noah_setmxsnalb.F90)

This subroutine retrieves NOAH parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah_setmxalb
```

USES:

```
use lisdrv_module, only : tile,lis
use noah_varder      ! NOAH tile variables
use lis_openfileMod
use lis_indices_module
```

CONTENTS:

```
!-----------------------------------------------------------------------
! The MAX SNOW ALBEDO field is opened and read in here:
!-----------------------------------------------------------------------
  allocate(tmpalb(lis_nc_data,lis_nr_data))
  call lis_set_filename(noahdrv%noah_mxsnal)
  print *, 'MSG: setnoahp -- Opening GLDAS MAXSNALB File', &
       trim(noahdrv%noah_mxsnal), ' (',iam,')'
  call lis_open_file(12,file=noahdrv%noah_mxsnal, &
       access='direct',form="unformatted",recl=4, script='getmaxsnalb.pl')

  call lis_read_file(12,tmpalb)
  close(12)

  do i=1,lis%d%nch
     if(tmpalb(tile(i)%col, tile(i)%row-lis_tnroffset).ne.-9999.00) then
        noah(i)%mxsnalb = tmpalb(tile(i)%col, tile(i)%row-lis_tnroffset)
     endif
  enddo

  deallocate(tmpalb)
  call absoft_release_cache()
```

1.75.5 noahsetsoils (Source File: noah_setsoils.F90)

This subroutine retrieves NOAH parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah_setsoils
```

USES:

```
  use lisdrv_module, only : grid,tile,lis
  use noah_varder      ! NOAH tile variables
  use lis_openfileMod
  use lis_indices_module
```

CONTENTS:

```
  print*,'MSG: setnoahp -- reading soil and clay files',' (',iam,')'
!-----------------------------------------------------------------------
```

```
! Open soil files (sand, clay, and porosity).
!-----------------------------------------------------------------------

  allocate(sand1(lis_nc_data,lis_nr_data))
  allocate(clay1(lis_nc_data,lis_nr_data))

!-----------------------------------------------------------------------
!     Read soil properties and convert to Zobler soil classes.
!     Note that the 3 files each contain 3 global records, one
!     for each layer depth (0-2, 2-150, 150-350 cm).  At this
!     time only the top layer data are used to map soil
!     parameters.  Since NOAH has 4 layers, the third layer of
!     porosity is temporarily used for layer 4 as well.
!-----------------------------------------------------------------------
  call readsand(lis%d%soil, sand1)
  call readclay(lis%d%soil, clay1)

  print*, 'MSG: setnoahp -- read sand and clay files',' (',iam,')'
!-----------------------------------------------------------------------
! Determine Zobler-equivalent soil classes derived from
! percentages of sand and clay, used in NOAH.
!-----------------------------------------------------------------------
  allocate(soiltyp(lis_nc_data, lis_nr_data))
  CALL SOILTYPE(lis_nc_data, lis_nr_data,SAND1,CLAY1,SOILTYP)
  deallocate(sand1)
  deallocate(clay1)
  call absoft_release_cache()
!-----------------------------------------------------------------------
! Read in the NOAH Soil Parameter File
!-----------------------------------------------------------------------
  open(unit=18,file=noahdrv%noah_sfile,status='old', &
       access='sequential')

  do i=1,noahdrv%noah_nsoilp
     read(18,*)(basicset(jj,i),jj=1,noahdrv%noah_zst)
  enddo
  close(18)
  print*, 'msg: setnoahp -- read sfile ', trim(noahdrv%noah_sfile), &
              ' (', iam, ')'
!-----------------------------------------------------------------------
! Convert grid space to tile space for soil type values.
!-----------------------------------------------------------------------
  allocate(placesltyp(lis%d%nch))
  do i=1,lis%d%nch
     placesltyp(i) = soiltyp(tile(i)%col, tile(i)%row-lis_tnroffset)
     noah(i)%zobsoil = placesltyp(i)
  end do
```

```
  if(lis%o%wparam.eq.1) then
     allocate(stype(lis%d%lnc,lis%d%lnr))
     stype = -9999.0
     if(lis%d%gridDesc(9) .ne. 0.01) then
        do i=1,lis%d%nch
           if(grid(i)%lat.ge.lis%d%gridDesc(4).and. &
                grid(i)%lat.le.lis%d%gridDesc(7).and. &
                grid(i)%lon.ge.lis%d%gridDesc(5).and. &
                grid(i)%lon.le.lis%d%gridDesc(8)) then
              rindex = tile(i)%row - nint((lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
                   /lis%d%gridDesc(9))
              cindex = tile(i)%col - nint((lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
                   /lis%d%gridDesc(10))
              stype(cindex,rindex) = noah(i)%zobsoil(1)*1.0
           endif
        enddo
     elseif(lis%d%gridDesc(9) .eq. 0.01) then
        do i=1,lis%d%nch
           stype(tile(i)%col,tile(i)%row) = noah(i)%zobsoil(1)*1.0
        enddo
     endif

     open(32,file="soiltype.bin",form='unformatted')
     write(32) stype
     close(32)
     deallocate(stype)
  endif

  deallocate(soiltyp)
  call absoft_release_cache()
!-----------------------------------------------------------------------
! Assign SOIL Parameters to each tile based on the
! type of Zobler soil class present in that tile.
!-----------------------------------------------------------------------
  do i=1,lis%d%nch            !tile loop
     k=placesltyp(i)           !soil type
     do j=1,noahdrv%noah_nsoilp   !soil parameter loop
        noah(i)%soilp(j)=basicset(k,j)
     enddo !j
  enddo !i
  deallocate(placesltyp)
  call absoft_release_cache()
  return
```

1.75.6 setnoahp.F90 (Source File: noah_settbot.F90)

This subroutine retrieves NOAH parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah_settbot
```

USES:

```
use lisdrv_module, only : tile,lis
use noah_varder      ! NOAH tile variables
use lis_openfileMod
use lis_indices_module
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Read in bottom temperature fields and adjust for elevation differnce
! with either Eta (NLDAS) or GDAS (GLDAS) correction datasets.
!-----------------------------------------------------------------------
  allocate(placetbot(lis_nc_data,lis_nr_data))

  call lis_set_filename(noahdrv%noah_tbot)
  print *, 'Opening GLDAS TBOT File ', noahdrv%NOAH_TBOT

  call lis_open_file(12, file=noahdrv%noah_tbot,   &
                     access='direct',status='old', &
                     form="unformatted", recl=4, script='gettbot.pl')
  call lis_read_file(12,placetbot)

  close(12)

  print*, 'MSG: setnoahp -- Read TBOT file',' (',iam,')'
  do i = 1, lis%d%nch
     if(placetbot(tile(i)%col,tile(i)%row-lis_tnroffset).ne.-9999.00) then
        noah(i)%tempbot = placetbot(tile(i)%col, tile(i)%row-lis_tnroffset)
     endif
  enddo
  deallocate(placetbot)
end subroutine noah_settbot

\markboth{Left}{Source File: noah\_setup.F90,  Date: Fri Feb 11 15:41:05 EST 2005
}
```

```
!-----------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------
```

1.75.7 noah_setup.F90 (Source File: noah_setup.F90)

Complete the setup routines for noah

REVISION HISTORY:

```
4 Nov. 1999: Paul Houser; Initial Code
28 Apr. 2002: K. Arsenault; Modified to NOAH LSM 2.5 code to LDAS
```

INTERFACE:

```
subroutine noah_setup()
```

USES:

```
use lisdrv_module, only: lis,tile
use noah_varder
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endif
     call noah_setvegparms()
     call noah_settbot()
```

```
     call noah_setmxalb()
     call noah_setsoils()
     call noah_gfrac()
     call noah_alb()
     call noah_coldstart()
     do t=1,lis%d%nch
        noah(t)%swnet = 0
        noah(t)%lwnet = 0
        noah(t)%qle = 0
        noah(t)%qh = 0
        noah(t)%qg = 0
        noah(t)%snowf = 0
        noah(t)%rainf = 0
        noah(t)%evap = 0
        noah(t)%qs = 0
        noah(t)%qsb = 0
        noah(t)%qsm = 0
        noah(t)%swe = 0
        noah(t)%soilmoist1 = 0
        noah(t)%soilmoist2 = 0
        noah(t)%soilmoist3 = 0
        noah(t)%soilmoist4 = 0
        noah(t)%soilwet = 0
        noah(t)%tveg = 0
        noah(t)%esoil = 0
        noah(t)%rootmoist =0
        noah(t)%soilm_prev = 0
        noah(t)%swe_prev = 0
        noah(t)%ecanop = 0
        noah(t)%canopint = 0
        noah(t)%count = 0
     enddo
#if ( ! defined OPENDAP )
      endif
      if ( npes > 1 ) then
         call noah_scatter
      endif
#endif
```

1.75.8 noah_setvegparms.F90 (Source File: noah_setvegparms.F90)

This subroutine retrieves NOAH parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault;  Added NOAH LSM, Initial Code
13 Oct 2003: Sujay Kumar; Domain independent modifications
```

INTERFACE:

```
subroutine noah_setvegparms
```

USES:

```
use lisdrv_module, only : tile,lis
use noah_varder      ! NOAH tile variables
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Convert UMD Classes to SIB Classes for Each Tile
!-----------------------------------------------------------------------
  print*,'MSG: setnoahp -- Calling MAPVEGC to convert UMD to SIB', &
       ' (', iam,')'
  print*,'DBG: setnoahp -- nch',lis%d%nch,' (',iam,')'

  do n=1,lis%d%nch
     call mapvegc(tile(n)%vegt)
     noah(n)%vegt = tile(n)%vegt
  enddo
  print*,'DBG: setnoahp -- left MAPVEGC',' (',iam,')'
!-----------------------------------------------------------------------
! Get Vegetation Parameters for NOAH Model in Tile Space
! Read in the NOAH Static Vegetation Parameter Files
!-----------------------------------------------------------------------
  open(unit=11,file=noahdrv%noah_vfile,status='old')

  do j=1,noahdrv%noah_nvegp
     read(11,*)(value(i,j),i=1,lis%p%nt)
  enddo
  close(11)
!-----------------------------------------------------------------------
! Assign STATIC vegetation parameters to each tile based on the
! type of vegetation present in that tile.
! These parameters will be stored in one long array--structured
! as follows: Tile 1, all the parameters (1 through numparam)
! then Tile 2, all the parameters.
! Then Tile 3, all the parameters etc.
!-----------------------------------------------------------------------

  do i=1,lis%d%nch
     do j=1,noahdrv%noah_nvegp
        noah(i)%vegp(j)=value(tile(i)%vegt,j)
     enddo
  enddo
```

```
end subroutine noah_setvegparms

\markboth{Left}{Source File: noah\_singlegather.F90,  Date: Fri Feb 11 15:41:06 EST 2005
}

!-----------------------------------------------------------------------
! NASA Goddard Space Flight Center Land Information System (LIS) V3.0
! Released May 2004
!
! See SOFTWARE DISTRIBUTION POLICY for software distribution policies
!
! The LIS source code and documentation are in the public domain,
! available without fee for educational, research, non-commercial and
! commercial purposes.  Users may distribute the binary or source
! code to third parties provided this statement appears on all copies and
! that no charge is made for such copies.
!
! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
! SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
! IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
!
! See COPYRIGHT.TXT for copyright details.
!
!-----------------------------------------------------------------------
#include "misc.h"
```

1.75.9 noah_singlegather.F90 (Source File: noah_singlegather.F90)

Gather single variable for output

REVISION HISTORY:

```
   Apr 2003 ; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine noah_singlegather(index, var)
%/////////////////////////////////////////////////////////////

\mbox{}\hrulefill\



\bigskip{\em USES:}
\begin{verbatim}   use lisdrv_module, only : lis
   use tile_spmdMod
```

```
use noah_varder
use noahpardef_module
IMPLICIT NONE
```

ARGUMENTS:

```
integer :: index              ! Index of Noah variable
real    :: var(lis%d%glbnch) ! Noah variable being gathered
```

CONTENTS:

```
do t = 1, di_array(iam)
   select case (index)
   case(1)
      var_temp(t) = noah(t)%swnet/float(noah(t)%count)
   case(2)
      var_temp(t) = (-1)*noah(t)%lwnet/float(noah(t)%count)
   case(3)
      var_temp(t) = noah(t)%qle/float(noah(t)%count)
   case(4)
      var_temp(t) = noah(t)%qh/float(noah(t)%count)
   case(5)
      var_temp(t) = noah(t)%qg/float(noah(t)%count)
   case(6)
      var_temp(t) = noah(t)%snowf/float(noah(t)%count)
   case(7)
      var_temp(t) = noah(t)%rainf/float(noah(t)%count)
   case(8)
      var_temp(t) = noah(t)%evap/float(noah(t)%count)
   case(9)
      var_temp(t) = noah(t)%qs/float(noah(t)%count)
   case(10)
      var_temp(t) = noah(t)%qsb/float(noah(t)%count)
   case(11)
      var_temp(t) = noah(t)%qsm/float(noah(t)%count)
   case(12)
      var_temp(t) = (noah(t)%smc(1)*1000.0*0.1+                        &
                     noah(t)%smc(2)*1000.0*0.3 +                       &
                     noah(t)%smc(3)*1000.0*0.6 +                       &
                     noah(t)%smc(4)*1000.0*1.0 - noah(t)%soilm_prev) / &
                    float(noah(t)%count)
   case(13)
      var_temp(t) = (noah(t)%sneqv*1000.0-noah(t)%swe_prev) / &
                    float(noah(t)%count)
   case(14)
      var_temp(t) = noah(t)%avgsurft
   case(15)
      var_temp(t) = noah(t)%albedo
   case(16)
```

```
      var_temp(t) = noah(t)%swe
   case(17)
      var_temp(t) = noah(t)%stc(1)
   case(18)
      var_temp(t) = noah(t)%stc(2)
   case(19)
      var_temp(t) = noah(t)%stc(3)
   case(20)
      var_temp(t) = noah(t)%stc(4)
   case(21)
      var_temp(t) = noah(t)%soilmoist1
   case(22)
      var_temp(t) = noah(t)%soilmoist2
   case(23)
      var_temp(t) = noah(t)%soilmoist3
   case(24)
      var_temp(t) = noah(t)%soilmoist4
   case(25)
      var_temp(t) = noah(t)%soilwet
   case(26)
      var_temp(t) = noah(t)%ecanop/float(noah(t)%count)
   case(27)
      var_temp(t) = noah(t)%tveg/float(noah(t)%count)
   case(28)
      var_temp(t) = noah(t)%esoil/float(noah(t)%count)
   case(29)
      var_temp(t) = noah(t)%rootmoist
   case(30)
      var_temp(t) = noah(t)%canopint
   case(31)
      if(lis%o%wfor.eq.1) then
         var_temp(t) = sqrt(noah(t)%forcing(5)*noah(t)%forcing(5)+ &
              noah(t)%forcing(6)*noah(t)%forcing(6))
      endif
   case(32)
      if(lis%o%wfor.eq.1) then
         if(noah(t)%forcing(1) < 273.15) then
            var_temp(t) = 0.0
         else
            var_temp(t) = noah(t)%forcing(8)
         endif
      endif
   case(33)
      if(lis%o%wfor.eq.1) then
         if(noah(t)%forcing(1) < 273.15) then
            var_temp(t) = noah(t)%forcing(8)
         else
            var_temp(t) = 0.0
```

```
            endif
         endif
      case(34)
         if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(1)
         endif
      case(35)
         if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(2)
         endif
      case(36)
         if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(7)
         endif
      case(37)
         if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(3)
         endif
      case(38)
         if(lis%o%wfor.eq.1) then
            var_temp(t) = noah(t)%forcing(4)
         endif
      end select
   enddo
#if ( defined SPMD )
   call MPI_GATHERV(var_temp(1:di_array(iam)), &
        di_array(iam), &
        MPI_REAL,var,di_array,displs,MPI_REAL, &
        0,MPI_COMM_WORLD, ierr)
#else
    var = var_temp
#endif
```

1.75.10 noah_singleout.F90 (Source File: noah_singleout.F90)

Write output file for a single noah variable

REVISION HISTORY:

```
14 Jun 2002 Sujay Kumar; Initial Specification
24 Aug 2004 James Geiger; Added missing variables
```

INTERFACE:

```
subroutine noah_singleout (ld,tile,var_array, index)
```

USES:

```
   use lis_module      ! LDAS non-model-specific 1-D variables
   use tile_module      ! LDAS non-model-specific tile variables
   use noah_varder, only : noahdrv
   use drv_output_mod, only : t2gr

   implicit none
```

ARGUMENTS:

```
   type (lisdec) :: ld    !data structure for lis domain specific variables
   type (tiledec) :: tile(ld%d%glbnch) !tile array for the modeled domain
   real           :: var_array(ld%d%glbnch) !array of variable that is being output
   integer        :: index  !Index of the output varible in the ALMA list.
```

CONTENTS:

```
!-----------------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------------
  IF(MOD(LD%T%GMT, noahdrv%WRITEINTN).EQ.0)THEN
     noahdrv%NUMOUT=noahdrv%NUMOUT+1
!-----------------------------------------------------------------------------
! Generate directory structure and file names for NOAH output
!-----------------------------------------------------------------------------
     length = len(trim(vname1(index)))
     WRITE(UNIT=temp1, FMT='(A10)') VNAME1(index)
     READ(UNIT=temp1,FMT='(10A1)') (FVARNAME(I), I=1,length)
     WRITE(UNIT=temp1,FMT='(I4,I2,I2)')LD%T%YR,LD%T%MO,LD%T%DA
     READ(UNIT=temp1,FMT='(8A1)') FTIME
     DO I=1,8
        IF(FTIME(I).EQ.(' '))FTIME(I)='0'
     ENDDO
     WRITE(UNIT=temp1,FMT='(I4)')LD%T%YR
     READ(UNIT=temp1,FMT='(8A1)')FTIMEC
     DO I=1,4
        IF(FTIMEC(I).EQ.(' '))FTIMEC(I)='0'
     ENDDO

#if 0
     WRITE(UNIT=temp1,FMT='(A7,I3,A1)') '/LDAS.E',LD%O%EXPCODE,'.'
     READ(UNIT=temp1,FMT='(80A1)') (FNAME(I),I=1,11)
     DO I=1,11
        IF(FNAME(I).EQ.(' '))FNAME(I)='0'
     ENDDO
#endif

!     idisk = mod(index, ld%o%odirn)
!     if ( idisk == 0 ) then
!        idisk = ld%o%odirn
```

```
!     endif
     idisk = 1
!     WRITE(UNIT=temp1,FMT='(A40)') LD%O%ODIR_ARRAY(idisk)
     WRITE(UNIT=temp1,FMT='(A40)') LD%O%ODIR
     READ(UNIT=temp1,FMT='(40A1)') (FBASE(I),I=1,40)
     C=0
     DO I=1,40
        IF(FBASE(I).EQ.(' ').AND.C.EQ.0)C=I-1
     ENDDO

     WRITE(UNIT=temp1,FMT='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
          LD%O%EXPCODE,'/NOAH/', &
          LD%T%YR,'/',LD%T%YR,LD%T%MO,LD%T%DA
     READ(UNIT=temp1,FMT='(80A1)') (FYRMODIR(I),I=1,26)
     DO I=1,26
        IF(FYRMODIR(I).EQ.(' '))FYRMODIR(I)='0'
     ENDDO

     WRITE(UNIT=temp1,FMT='(A9)')'mkdir -p '
     READ(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9)

     WRITE(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
          (FYRMODIR(I),I=1,26)
     READ(UNIT=temp1,FMT='(A80)')MKFYRMO
!-----------------------------------------------------------------------------
!  Make the directories for the NOAH output data files
!-----------------------------------------------------------------------------
     CALL SYSTEM(MKFYRMO)
!-----------------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------------
     IF(LD%O%WOUT.EQ.1) THEN
        WRITE(UNIT=FBINNAME, FMT='(I4,I2,I2,I2)') LD%T%YR,LD%T%MO, &
             LD%T%DA,LD%T%HR
        READ(UNIT=FBINNAME,FMT='(10A1)') FTIMEB
        DO I=1,10
           IF(FTIMEB(I).EQ.(' '))FTIMEB(I)='0'
        ENDDO
!        WRITE(UNIT=FBINNAME,FMT='(A9)') '.NOAHgbin'
!        READ(UNIT=FBINNAME,FMT='(80A1)') (FSUBGB(I),I=1,9)
        if ( ld%d%gridDesc(9) .eq. 0.01) then
           write(unit=fbinname,fmt='(A5)') '.gd4r'
           read(unit=fbinname,fmt='(80A1)') (fsubgb(i),i=1,5)
        else
           write(unit=fbinname,fmt='(A5)') '.ls4r'
           read(unit=fbinname,fmt='(80A1)') (fsubgb(i),i=1,5)
        endif
#if 0
```

```
        WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
             (FYRMODIR(I),I=1,26), &
             (FNAME(I),I=1,11),(FTIMEB(I),I=1,10), &
             (FVARNAME(I), I=1,length),(FSUBGB(I),I=1,9)
        READ(UNIT=FBINNAME,FMT='(A80)')FILENGB
#endif
        WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
             (FYRMODIR(I),I=1,26), '/',&
             (FTIMEB(I),I=1,10), &
             (FVARNAME(I), I=1,length),(FSUBGB(I),I=1,5)
        READ(UNIT=FBINNAME,FMT='(A80)')FILENGB
!-----------------------------------------------------------------------------
! Open statistical output file
!-----------------------------------------------------------------------------
        IF(noahdrv%NOAHopen.EQ.0)THEN
           FILE='Noahstats.dat'
           CALL OPENFILE(NAME,LD%O%ODIR,LD%O%EXPCODE,FILE)
           IF(LD%O%STARTCODE.EQ.1)THEN
              OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='UNKNOWN', &
                   POSITION='APPEND')
           ELSE
              OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='REPLACE')
           ENDIF
           noahdrv%NOAHopen=1
        ENDIF

        WRITE(65,996)'       Statistical Summary of NOAH Output for:  ', &
             LD%T%MO,'/',LD%T%DA,'/',LD%T%YR,LD%T%HR,':',LD%T%MN,':',LD%T%SS
996     FORMAT(A47,I2,A1,I2,A1,I4,1X,I2,A1,I2,A1,I2)
        WRITE(65,*)
       WRITE(65,997)
997    FORMAT(T27,'Mean',T41,'StDev',T56,'Min',T70,'Max')
    ENDIF
!-----------------------------------------------------------------------------
! Write output in HDF and binary (if WBIN=1) format
!-----------------------------------------------------------------------------
    if ( ld%o%wout == 1 ) then
       if ( ld%d%gridDesc(9) .eq. 0.01 ) then ! special 1km regional domain
           allocate(g2tmp(ld%d%lnc,ld%d%lnr))
           g2tmp = ld%d%UDEF
           do i = 1, ld%d%glbnch
             g2tmp(tile(i)%col, tile(i)%row) = var_array(i)
           enddo

              open(58,file=filengb,form='unformatted',access='direct', &
                   recl=ld%d%lnc * ld%d%lnr * 4)
              write(58, rec=1) g2tmp
!           else !*** multi disk output
```

```
!              open(58,file=filengb,form='unformatted',access='direct', &
!                   recl=ld%d%lnc*4)
!              do i = 1, ld%d%lnr
!                 nrec = (ld%d%ir-1) * ld%d%lnr * 50 + ( i - 1 ) * 50 + ld%d%ic
!                 write(58,rec=nrec) g2tmp(:,i)
!              enddo
!           endif
           deallocate(g2tmp)
       else                          ! all other domains
          open(58,file=filengb,form='unformatted')
          allocate(gtmp(ld%d%glbngrid))
          call t2gr(var_array,gtmp,ld%d%glbngrid,ld%d%glbnch,tile)
          write(58) gtmp
          deallocate(gtmp)
       endif
       call stats(var_array,ld%d%udef,ld%d%glbnch, vmean, &
                  vstdev, vmin, vmax);
       write(65,999) vname(index),vmean, vstdev, vmin, vmax
       close(58)
    endif

998 FORMAT(1X,A18,4E14.3)
999 FORMAT(1X,A18,4F14.3)
 endif
```

1.75.11 noah_soiltype.F90 (Source File: noah_soil_typ.F90)

This subroutine uses the percentages of sand and clay derived from the global soils dataset of Reynolds, Jackson, and Rawls [1999], to convert to Zobler soil class values to be used in NOAH LSM v2.5 in LDAS. (Original code by Matt Rodell, 3/7/01)
28 Apr 2002, K Arsenault: Added NOAH LSM to LDAS

INTERFACE:

```
 subroutine soiltype(nc,nr,sand,clay,soiltyp)
```

CONTENTS:

```
  do j=1,nr
     do i=1,nc
        if (clay(i,j) .lt. 0.00) then
           soiltyp(i,j) = -99
        else
           cl = clay(i,j)
           sa = sand(i,j)
        endif
!-----------------------------------------------------------------
```

```
!     identify texture class.
!-------------------------------------------------------------------
        if (cl .lt. 0.23) then
           if (sa .lt. 0.50) then
              soiltyp(i,j) = 8          ! loam
           else
              if (sa .lt. 0.75) then
                 soiltyp(i,j) = 4        ! sandy loam
              else
                 soiltyp(i,j) = 1        ! loamy sand
              end if
           end if
        else
           if (cl .lt. 0.28) then
              if (sa .lt. 0.45) then
                 soiltyp(i,j) = 8        ! loam
              else
                 soiltyp(i,j) = 7        ! sandy clay loam
              endif
           else
              if (cl .lt. 0.37) then
                 if (sa .lt. 0.2) then
                    soiltyp(i,j) = 2      ! silty clay loam
                 else
                    if (sa .lt. 0.43) then
                       soiltyp(i,j) = 6    ! clay loam
                    else
                       soiltyp(i,j) = 7    ! sandy clay loam
                    end if
                 end if
              else
                 if (cl .lt. 0.41) then
                    if (sa .lt. 0.2) then
                       soiltyp(i,j) = 2   ! silty clay loam
                    else
                       if (sa .lt. 0.43) then
                          soiltyp(i,j) = 6    ! clay loam
                       else
                          soiltyp(i,j) = 5    ! sandy clay
                       end if
                    end if
                 else
                   if (sa .lt. 0.43) then
                      soiltyp(i,j) = 3      ! light clay
                   else
                      soiltyp(i,j) = 5      ! sandy clay
                   end if
                end if
```

```
            end if
         end if
      end if
   end do
end do
```

1.75.12 noah_totinit.F90 (Source File: noah_totinit.F90)

Initialize NOAH output arrays

REVISION HISTORY:

```
14 Jun 2002  Sujay Kumar  Initial Specification
```

INTERFACE:

```
subroutine noah_totinit()
```

USES:

```
use noah_varder      ! NOAH LSM module
use tile_spmdMod
use lisdrv_module, only : lis
```

CONTENTS:

```
do t = 1, di_array(iam)
   if(mod(lis%t%gmt,noahdrv%writeintn).eq.0)then
      noah(t)%soilm_prev=noah(t)%smc(1)*1000.0*0.1+ &
           noah(t)%smc(2)*1000.0*0.3 + &
           noah(t)%smc(3)*1000.0*0.6 + &
           noah(t)%smc(4)*1000.0
      noah(t)%swe_prev =  noah(t)%sneqv*1000.0
   endif
enddo
do t = 1, di_array(iam)
   noah(t)%swnet = 0
   noah(t)%lwnet = 0
   noah(t)%qle = 0
   noah(t)%qh = 0
   noah(t)%qg = 0
   noah(t)%snowf = 0
   noah(t)%rainf = 0
   noah(t)%evap = 0
   noah(t)%qs = 0
   noah(t)%qsb = 0
   noah(t)%qsm = 0
   noah(t)%ecanop = 0
```

```
     noah(t)%tveg = 0
     noah(t)%esoil = 0
     noah(t)%count = 0
  enddo
```

1.75.13 noah_varder.F90 (Source File: noah_varder.F90)

Module for 1-D NOAH land model driver variable initialization

REVISION HISTORY:

```
   Apr 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
 module noah_varder
```

USES:

```
   use noah_module
   use tile_spmdMod
   use noahpardef_module
   use noahdrv_module
```

1.75.14 noah_varder_ini (Source File: noah_varder.F90)

Reads in runtime noah parameters, allocates memory for variables

INTERFACE:

```
   subroutine noah_varder_ini(nch)
```

USES:

```
 #if ( defined OPENDAP )
     use opendap_module
 #endif
```

CONTENTS:

```
    if(masterproc) then
       call readnoahcrd(noahdrv)
    endif
    call def_noahpar_struct
#if (defined SPMD)
    call MPI_BCAST(noahdrv, 1, MPI_NOAHDRV_STRUCT, 0, &
         MPI_COMM_WORLD, ierr)
#endif
```

```
#if ( defined OPENDAP )
    noahdrv%noah_albfile = trim(opendap_data_prefix)//'/'// &
                           trim(adjustl(ciam))//'/'//noahdrv%noah_albfile
    noahdrv%noah_mgfile  = trim(opendap_data_prefix)//'/'// &
                           trim(adjustl(ciam))//'/'//noahdrv%noah_mgfile
    noahdrv%noah_mxsnal  = trim(opendap_data_prefix)//'/'// &
                           trim(adjustl(ciam))//'/'//noahdrv%noah_mxsnal
    noahdrv%noah_tbot    = trim(opendap_data_prefix)//'/'// &
                           trim(adjustl(ciam))//'/'//noahdrv%noah_tbot
#endif
    if(masterproc) then
       allocate(noah(nch))
    else
       allocate(noah(di_array(iam)))
    endif
  end subroutine noah_varder_ini
```

1.75.15 noah_writerst.F90 (Source File: noah_writerst.F90)

This program writes restart files for NOAH. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
    1  Oct 1999: Jared Entin; Initial code
    15 Oct 1999: Paul Houser; Significant F90 Revision
    05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA
                 initial conditions if necessary.  This is controlled with
                 local variable NOAAIC.  Normally set to 0 in this subroutine
                 but set to 1 if want to use Dag's NOAA IC's.  Changed output
                 directory structure, and commented out if-then check so that
                 directory is always made.
    28 Apr 2002: Kristi Arsenault; Added NOAH LSM into LDAS
    28 May 2002: Kristi Arsenault; For STARTCODE=4, corrected SNEQV values
                  and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
    14 Jun 2003: Sujay Kumar , Separated the write restart from the original
                  code
    24 Aug 2004: James Geiger, Added support for GrADS-DODS server and
                               corrected MPI base parallel restart writing
   RESTART FILE FORMAT(fortran sequential binary):
    YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
    TILE(NCH)%COL        !Grid Col of Tile
    TILE(NCH)%ROW        !Grid Row of Tile
    TILE(NCH)%FGRD       !Fraction of Grid covered by tile
    TILE(NCH)%VEGT       !Vegetation Type of Tile
```

```
    NOAH(NCH)%STATES     !Model States in Tile Space

 #include "misc.h"
```

INTERFACE:

```
 subroutine noah_writerst()
   !uses:
   use lisdrv_module, only : lis,tile
   use time_manager
   use noah_varder      ! NOAH tile variables
```

CONTENTS:

```
   if ( ( lis%t%gmt == (24-noahdrv%writeintn) ) .or. &
        lis%t%endtime == 1 ) then
!
! All processes of a multi-process run call this subroutine
!
! If you are not writing bundled output, then noah_gather has not been
! called.  Gather the Noah data structure, unless you are using a GrADS-DODS
! server (OPENDAP).  In this case, the variables will be gathered one at a time.
!
#if ( ! defined OPENDAP )
     if ( lis%o%wsingle == 1 .and. npes > 1 ) then
        call noah_gather()
     endif

! If you are using a GrADS-DODS server then every process
! should continue, else only the master process.
     if ( masterproc ) then
#endif

!-----------------------------------------------------------------------
! Restart Writing (2 files are written = active and archive)
!-----------------------------------------------------------------------


! Eventhough all processes may continue to this point, only the master
! process should perform any writing.
        if ( masterproc ) then
           !Active archive restart
           open(40,file=noahdrv%noah_rfile,form='unformatted')
        endif

        call noah_dump_restart(40)

        if ( masterproc ) then
```

```
            close(40)
         endif

         if ( masterproc ) then
            write(*,*)'Noah Active restart written: ',noahdrv%noah_rfile
            write(unit=temp,fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
                lis%t%da,lis%t%hr
            read(unit=temp,fmt='(10a1)')ftime
            do i=1,10
              if(ftime(i).eq.(' '))ftime(i)='0'
            enddo
            write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2,a6,i3,a1)') &
                '/EXP',lis%o%expcode,'/NOAH/',lis%t%yr, &
                '/',lis%t%yr,lis%t%mo, &
                lis%t%da,'/LIS.E',lis%o%expcode,'.'
            read(unit=temp,fmt='(80a1)') (fname(i),i=1,37)
            do i=1,37
              if(fname(i).eq.(' '))fname(i)='0'
            enddo
            write(unit=temp,fmt='(a9)')'mkdir -p '
            read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)
            write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)') &
                '/EXP',lis%o%expcode,'/NOAH/', &
                lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
            read(unit=temp,fmt='(80a1)') (fyrmodir(i),i=1,26)
            do i=1,26
              if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
            enddo

            write(unit=temp,fmt='(a8)')'.Noahrst'
            read(unit=temp,fmt='(80a1)') (fsubs(i),i=1,8)

            write(unit=temp,fmt='(a40)') lis%o%odir
            read(unit=temp,fmt='(80a1)') (fbase(i),i=1,80)
            c=0
            do i=1,80
              if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
            enddo
            write(unit=temp,fmt='(80a1)')(fbase(i),i=1,c),(fname(i),i=1,37), &
                            (ftime(i),i=1,10),(fsubs(i),i=1,8)
            read(unit=temp,fmt='(a80)')filen

            write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
             (fyrmodir(i),i=1,26)
            read(unit=temp,fmt='(a80)')mkfyrmo

!-----------------------------------------------------------------------
! Archive File Name Generation Complete
```

```
! Make the directories for the NOAH restart file
!-----------------------------------------------------------------------
           call system(mkfyrmo)
!-----------------------------------------------------------------------
! Archive File Name Generation Complete
!-----------------------------------------------------------------------
           open(40,file=filen,status='unknown',form='unformatted')
        endif

        call noah_dump_restart(40)

        if ( masterproc ) then
           close(40)
           write(*,*)'noah archive restart written: ',filen
        endif

#if ( ! defined OPENDAP )
      endif
#endif
   endif

   return
```

1.75.16 noah_dump_restart (Source File: noah_writerst.F90)

This routine gathers the necessary restart variables and performs the actual write statements to create the restart files.

REVISION HISTORY:

```
24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
subroutine noah_dump_restart(ftn)
```

USES:

```
use lisdrv_module, only : lis
use noah_varder
use tile_spmdMod
use time_manager
```

1.75.17 noah_gather_restart (Source File: noah_writerst.F90)

This routine is a wrapper for the mpi_gatherv subroutine. It performs any necessary data gathering for the noah_writerst routine.

REVISION HISTORY:

```
    24 Aug 2004: James Geiger, Initial Specification
```

INTERFACE:

```
 subroutine noah_gather_restart(tmptilen)

    use spmdMod
    use tile_spmdMod

    implicit none

    real, dimension(*), intent(inout) :: tmptilen
    integer :: ierr

 #if ( defined SPMD )

    call MPI_GATHERV(tmptilen(1:di_array(iam)), di_array(iam),       &
                     MPI_REAL, tmptilen, di_array, displs, MPI_REAL, &
                     0, MPI_COMM_WORLD, ierr)
 #endif

 end subroutine noah_gather_restart
\markboth{Left}{Source File: noah\_writestats.F90,  Date: Fri Feb 11 15:41:06 EST 2005
}


  ------------------------------------------------------------------------
   NASA Goddard Space Flight Center Land Information System (LIS) V3.0
   Released May 2004
   See SOFTWARE DISTRIBUTION POLICY for software distribution policies
   The LIS source code and documentation are in the public domain,
   available without fee for educational, research, non-commercial and
   commercial purposes.  Users may distribute the binary or source
   code to third parties provided this statement appears on all copies and
   that no charge is made for such copies.
   NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE
   SOFTWARE FOR ANY PURPOSE.  IT IS PROVIDED AS IS WITHOUT EXPRESS OR
   IMPLIED WARRANTY.  NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE
   LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE.
   See COPYRIGHT.TXT for copyright details.
  ------------------------------------------------------------------------
%//////////////////////////////////////////////////////////////
```

```
\mbox{}\hrulefill\


  \subsubsection{noah\_writevars.F90 (Source File: noah\_writestats.F90)}



    LIS NOAH data writer: Writes noah output

\bigskip{\sf REVISION HISTORY:}
\begin{verbatim}   02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
 subroutine noah_writestats(ftn_stats)
```

USES:

```
   use lisdrv_module, only : lis
   use noah_varder

   implicit none

   integer :: ftn_stats,t
```

CONTENTS:

```
  do t=1,lis%d%glbnch
     if(noah(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = noah(t)%forcing(8)
     else
        rainf(t) = noah(t)%forcing(8)
        snowf(t) = 0.0
     endif
  enddo
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
   call stats(noah%swnet,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SWnet(W/m2)', &
        vmean,vstdev,vmin,vmax
   call stats(noah%lwnet,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'LWnet(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%qle,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
```

```
   write(ftn_stats,999) 'Qle(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%qh,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'Qh(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%qg,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'Qg(W/m2)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   call stats(noah%snowf,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Snowf(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%rainf,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Rainf(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%evap,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Evap(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%qs,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Qs(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%qsb,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Qsb(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats((noah%smc(1)*1000.0*0.1+ &
        noah%smc(2)*1000.0*0.3 + &
        noah%smc(3)*1000.0*0.6 + &
        noah%smc(4)*1000.0 -noah%soilm_prev)/float(noah%count), &
        lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
   write(ftn_stats,999) 'DelSoilMoist(kg/m2s)', &
        vmean,vstdev,vmin,vmax
   call stats( (noah%sneqv*1000.0-noah%swe_prev)/float(noah%count), &
        lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
   write(ftn_stats,999) 'DelSWE(kg/m2s)', &
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call stats(noah%avgsurft,lis%d%udef,lis%d%glbnch,vmean, &
```

```
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'AvgSurfT(K)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%albedo,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Albedo(-)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%swe,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'SWE(kg/m2)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   call stats(noah%soilmoist1,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist1(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%soilmoist2,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist2(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%soilmoist3,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist3(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%soilmoist4,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist4(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%stc(1),lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilTemp1 (K)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%stc(2),lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilTemp2 (K)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%stc(3),lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilTemp3 (K)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%stc(4),lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilTemp4 (K)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%soilwet,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
```

```
   write(ftn_stats,998) 'SoilWet(-)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
   call stats(noah%ecanop,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'ECanop(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%tveg,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'TVeg(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%esoil,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'ESoil(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%rootmoist,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'RootMoist(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(noah%canopint,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'CanopInt(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   if(lis%o%wfor.eq.1) then
      call stats(sqrt(noah%forcing(5)*noah%forcing(5)+ &
           noah%forcing(6)*noah%forcing(6)), &
           lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,999) 'Wind(m/s)', &
           vmean,vstdev,vmin,vmax
      call stats(rainf, &
           lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,998) 'Rainfforc(kg/m2s)', &
           vmean,vstdev,vmin,vmax
      call stats(snowf, &
           lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,998) 'Snowfforc(kg/m2s)', &
           vmean,vstdev,vmin,vmax
      call stats(noah%forcing(1),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
           vmin, vmax)
      write(ftn_stats,999) 'Tair(K)', &
           vmean,vstdev,vmin,vmax
      call stats(noah%forcing(2),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
           vmin, vmax)
      write(ftn_stats,999) 'Qair(kg/kg)', &
           vmean,vstdev,vmin,vmax
      call stats(noah%forcing(7),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
```

```
            vmin, vmax)
      write(ftn_stats,999) 'PSurf(Pa)', &
            vmean,vstdev,vmin,vmax
      call stats(noah%forcing(3),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'SWdown (W/m2)', &
            vmean,vstdev,vmin,vmax
      call stats(noah%forcing(4),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'LWdown(W/m2)', &
            vmean,vstdev,vmin,vmax
   endif
998    FORMAT(1X,A18,4E14.3)
999    FORMAT(1X,A18,4F14.3)
```

1.75.18 readkpds.F90 (Source File: readkpds.F90)

Reads the kpds array from the grib table

REVISION HISTORY:

```
23 Oct 2003; Sujay Kumar; Initial Version
```

INTERFACE:

```
subroutine readkpds(ftn, kpds)
```

INTERFACE:

```
  use noah_varder, only : noahdrv
```

1.75.19 readnoahcrd.F90 (Source File: readnoahcrd.F90)

Routine to read Noah specific parameters from the card file.

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readnoahcrd(noahdrv)
```

USES:

```
use noahdrv_module
```

CONTENTS:

```
  open(11,file='lis.crd',form='formatted',status='old')
  read(unit=11,NML=noah)
  print*,'Running NOAH LSM:'
  print*,'NOAH Active Restart File: ', noahdrv%NOAH_RFILE
  noahdrv%noah_gfractime = 0.0
  noahdrv%noah_albtime = 0
  noahdrv%noah_albdchk = 0
  noahdrv%noah_gfracdchk = 0
  noahdrv%NOAHOPEN=0
  noahdrv%NOAH_ZST     = 9

  close(11)
```

1.75.20 sh2o_init.F90 (Source File: sh2o_init.F90)

To do a 'cold start' initialization of soil liquid water SH2O for NOAH LSM, though also adaptable for other land-surface models, using either GDAS or Eta forcing data.

REVISION HISTORY:

```
    NCEP;  Original code developed by NCEP for Eta model
             subroutines to initialize soil liquid water, SH2O
    04 Nov 2002: Kristi Arsenault; Modified code to be used with noahrst.f
                                   to initialize NOAH with NCEP forcing data
```

INTERFACE:

```
 subroutine sh2oinit(smc,stc,smcmax,psis,beta,sh2o)

   implicit none
```

ARGUMENTS:

```
   REAL STC      ! NOAH Soil Layer Temperature (K)
   REAL SMC      ! NOAH Soil Layer Total Moisture (liq+frzn)
   REAL SH2O     ! NOAH Soil Layer Liquid Moisture

   REAL  PSIS                    ! Saturated soil potential
   REAL  BETA                    ! B-parameter
   REAL  SMCMAX                  ! Max soil moisture content (porosity)
   REAL  BX
```

CONTENTS:

```
! ----------------------------------------------------------------------
! COLD START:  determine liquid soil water content (SH2O)
! NSOIL number of soil layers
! ----------------------------------------------------------------------
```

```
!  SH2O <= SMC for T < 273.149K (-0.001C)
  IF (STC .LT. 273.149) THEN
! ---------------------------------------------------------------------
! first guess following explicit solution for Flerchinger Eqn from Koren
! et al, JGR, 1999, Eqn 17 (KCOUNT=0 in FUNCTION FRH2O).
! ---------------------------------------------------------------------
     BX = BETA
     IF ( BETA .GT. BLIM )  BX = BLIM
     FK=(((HLICE/(GRAV*(-PSIS)))* &
          ((STC-T0)/STC))**(-1/BX))*SMCMAX
     IF (FK .LT. 0.02) FK = 0.02
     SH2O = MIN ( FK, SMC )
! ---------------------------------------------------------------------
! now use iterative solution for liquid soil water content using
! FUNCTION FRH2O with the initial guess for SH2O from above explicit
! first guess.
! ---------------------------------------------------------------------
     SH2O = FRH2O(STC,SMC,SH2O,SMCMAX,BETA,PSIS)

  ELSE
! ---------------------------------------------------------------------
!  SH2O = SMC for T => 273.149K (-0.001C)
     SH2O=SMC
! ---------------------------------------------------------------------
  ENDIF

  RETURN
```

1.75.21 noah_atmdrv.F90 (Source File: mos_atmdrv.F90)

Transfer forcing from grid to tile space.

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial Code
28 Jan 2002: Jon Gottschalck; Added option for different number of forcing variables
```

INTERFACE:

```
subroutine mos_f2t(t, forcing)
```

USES:

```
use lisdrv_module , only : lis,tile     ! LDAS non-model-specific 1-D variables
use spmdMod
use tile_spmdMod
use mos_varder
```

CONTENTS:

```
      do f=1,lis%f%nforce
        mos(t)%forcing(f)=forcing(f)
      enddo
      return
  end subroutine mos_f2t

\markboth{Left}{Source File: mos\_binout.F90,  Date: Fri Feb 11 15:41:06 EST 2005
}

!-------------------------------------------------------------------------
!     NASA GSFC Land Information Systems LIS 2.3                         !
!-------------------------------------------------------------------------
```

1.75.22 mos_binout.F90 (Source File: mos_binout.F90)

LIS MOS data writer: Writes mos output in binary format

REVISION HISTORY:

```
   02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
 subroutine mos_binout(ftn)
```

USES:

```
   use lisdrv_module, only : lis
   use drv_output_mod, only : drv_writevar_bin
   use mos_varder

   implicit none

   integer :: ftn
```

CONTENTS:

```
  do t=1,lis%d%glbnch
     if(mos(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = mos(t)%forcing(8)
     else
        rainf(t) = mos(t)%forcing(8)
        snowf(t) = 0.0
     endif
  enddo
```

```
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
   mos%swnet = mos%swnet/float(mos%count)
   call drv_writevar_bin(ftn,mos%swnet) !Net shortwave radiation (surface) (W/m2)
   mos%lwnet = (-1)*mos%lwnet/float(mos%count)
   call drv_writevar_bin(ftn,mos%lwnet)!Net longwave radiation (surface) (W/m2)
   mos%qle = mos%qle/float(mos%count)
   call drv_writevar_bin(ftn,mos%qle) !Latent Heat Flux (W/m2)
   mos%qh = mos%qh/float(mos%count)
   call drv_writevar_bin(ftn,mos%qh) !Sensible Heat Flux (W/m2)
   mos%qg = mos%qg/float(mos%count)
   call drv_writevar_bin(ftn,mos%qg)
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   mos%snowf = mos%snowf/float(mos%count)
   call drv_writevar_bin(ftn,mos%snowf)
   mos%rainf = mos%rainf/float(mos%count)
   call drv_writevar_bin(ftn,mos%rainf)
   mos%evap = mos%evap/float(mos%count)
   call drv_writevar_bin(ftn,mos%evap)
   mos%qs = mos%qs/float(mos%count)
   call drv_writevar_bin(ftn,mos%qs)
   mos%qsb = mos%qsb/float(mos%count)
   call drv_writevar_bin(ftn,mos%qsb)
   mos%qsm = mos%qsm/float(mos%count)
   call drv_writevar_bin(ftn,mos%qsm)
   call drv_writevar_bin(ftn,(mos%water1 + &
        mos%water2 + &
        mos%water3  &
        -mos%soilm_prev)/float(mos%count))
   call drv_writevar_bin(ftn,(mos%snow-mos%swe_prev)/float(mos%count))
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call drv_writevar_bin(ftn,mos%avgsurft)
   call drv_writevar_bin(ftn,mos%soT)
   call drv_writevar_bin(ftn,mos%albedo)
   mos%swe= mos%swe/float(mos%count)
   call drv_writevar_bin(ftn,mos%swe)
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   mos%soilmoist1= mos%soilmoist1/float(mos%count)
   call drv_writevar_bin(ftn,mos%soilmoist1)
   mos%soilmoist2= mos%soilmoist2/float(mos%count)
   call drv_writevar_bin(ftn,mos%soilmoist2)
```

```
   mos%soilmoist3= mos%soilmoist3/float(mos%count)
   call drv_writevar_bin(ftn,mos%soilmoist3)
   mos%soilwet= mos%soilwet/float(mos%count)
   call drv_writevar_bin(ftn,mos%soilwet)
!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
   mos%ecanop = mos%ecanop/float(mos%count)
   call drv_writevar_bin(ftn,mos%ecanop)
   mos%tveg= mos%tveg/float(mos%count)
   call drv_writevar_bin(ftn,mos%tveg)
   mos%esoil= mos%esoil/float(mos%count)
   call drv_writevar_bin(ftn,mos%esoil)
   mos%rootmoist = mos%rootmoist/float(mos%count)
   call drv_writevar_bin(ftn, mos%rootmoist)
   mos%canopint = mos%canopint/float(mos%count)
   call drv_writevar_bin(ftn, mos%canopint)
   mos%acond = mos%acond
   call drv_writevar_bin(ftn, mos%acond)
   if(lis%o%wfor.eq.1) then
      call drv_writevar_bin(ftn, sqrt(mos%forcing(5)*mos%forcing(5)+ &
           mos%forcing(6)*mos%forcing(6)))
      call drv_writevar_bin(ftn,rainf)
      call drv_writevar_bin(ftn,snowf)
      call drv_writevar_bin(ftn,mos%forcing(1))
      call drv_writevar_bin(ftn,mos%forcing(2))
      call drv_writevar_bin(ftn,mos%forcing(7))
      call drv_writevar_bin(ftn,mos%forcing(3))
      call drv_writevar_bin(ftn,mos%forcing(4))
   endif
```

1.75.23 mos_coldstart.F90 (Source File: mos_coldstart.F90)

Routine for mosaic initialization from cold start

INTERFACE:

```
 subroutine mos_coldstart()
```

USES:

```
    use lisdrv_module, only : lis
    use mos_varder
    use time_manager
    use spmdMod, only : iam
```

CONTENTS:

```
if ( lis%o%startcode == 2 ) then
   print*,'MSG: mos_coldstart -- cold-starting mosaic', &
          '...using ics from card file',' (', iam, ')'

   print*,'DBG: mos_coldstart -- nch',lis%d%nch, &
        ' (', iam, ')'
   do t=1,lis%d%nch
      mos(t)%ct=mosdrv%mos_it
      mos(t)%qa=0.0
      mos(t)%ics=0.0
      mos(t)%snow=0.0
      mos(t)%SoT=mosdrv%mos_it
      do l=1,3
         mos(t)%SoWET(l)=mosdrv%mos_ism
      enddo
   enddo
   lis%t%yr=lis%t%syr
   lis%t%mo=lis%t%smo
   lis%t%da=lis%t%sda
   lis%t%hr=lis%t%shr
   lis%t%mn=lis%t%smn
   lis%t%ss=lis%t%sss

   call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr,&
                  lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
   write(*,*)'MSG: mos_coldstart -- Using lis.crd start time ',&
             lis%t%time, ' (', iam, ')'
endif
```

1.76 Fortran: Module Interface mosdrv_module.F90 (Source File: mosdrv_module.F90)

Module for runtime specific Mosaic variables

REVISION HISTORY:

```
15 Oct 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
MODULE mosdrv_module
```

1.76.1 noah_dynsetup.F90 (Source File: mos_dynsetup.F90)

Updates the time dependent MOS variables

REVISION HISTORY:

```
4 Nov. 1999: Jon Gottschalck; Initial Code
```

INTERFACE:

```
subroutine mos_dynsetup()
```

USES:

```
use lisdrv_module, only : lis, tile
use mos_varder
use spmdMod, only : masterproc, npes
use mospardef_module
```

CONTENTS:

```
  integer :: t, n,ier
#if ( ! defined OPENDAP )
  if ( npes > 1 ) then
     call mos_gather
  endif
#endif
  if (masterproc) then
     call mosdynp(lis%d, lis%t, lis%p)
  endif
#if ( ! defined OPENDAP )

!  call MPI_BCAST(mosdrv%noah_gflag,1,MPI_INTEGER,0, &
!       MPI_COMM_WORLD,ier)
!  call MPI_BCAST(mosdrv%noah_aflag,1,MPI_INTEGER,0, &
!       MPI_COMM_WORLD,ier)
!  if( npes > 1 .and. ( noahdrv%noah_gflag==1 .or. &
!      noahdrv%noah_aflag ==1 ) ) then
      call mos_scatter
!  endif
#endif
```

1.76.2 mos_gather.F90 (Source File: mos_gather.F90)

Gathers mosaic tiles

REVISION HISTORY:

```
Aug 2003 ; Jon Gottschalck, Initial Code
```

INTERFACE:

```
subroutine mos_gather()
```

USES:

```
use tile_spmdMod
use mos_varder
use mospardef_module
```

1.77 Fortran: Module Interface mos_module.F90: (Source File: mos_module.F90)

Module for 1-D MOSAIC land model driver variable specification.

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial code
11 Feb 2002: Jon Gottschalck; Added AVHRR derived variables
```

INTERFACE:

```
module mos_module
```

CONTENTS:

```
  type mosdec

      INTEGER :: ts                 !Timestep (seconds)
      INTEGER :: NSLAY               !Number of MOSAIC soil layers
      INTEGER :: COUNT                !MOSAIC Output Counter
      INTEGER :: VEGT

      REAL :: VEGP(24)       !Static vegetation parameter values, dim(MOS_NVEGP)
!      REAL :: VEGMP(6,12)    !Monthly vegetation parameter values dim(MOS_NMVEGP,12)
      REAL :: VEGIP(6)      !Interpolated from monthly parameters (MOS_NMVEGP)
      REAL :: SOILP(10)      !Static soil parameter values, dim(NOS_NSOILP)

      REAL :: LAI           !AVHRR Derived LAI
      REAL :: GREEN         !AVHRR Derived GREENNESS
      REAL :: DSAI          !AVHRR Derived DSAI

!=== LDAS-MOSAIC States ================================================
      REAL :: CT                     !MOSAIC Canopy/Soil Temperature
      REAL :: QA                     !MOSAIC Canopy Humidity
      REAL :: ICS                    !MOSAIC Interception Canopy Storage
      REAL :: SNOW                   !MOSAIC Snow Depth
      REAL :: SoT                    !MOSAIC Deep Soil Temperaure
      REAL :: SoWet(3)      !MOSAIC Soil Wetness (3 layers)
!=== Analysis and Bias Correction Variables ==========================
      REAL :: DTCANAL                !MOSAIC Change in Temperature based on Analysis
      real :: lai1, lai2
      real :: sai1, sai2
!=== LDAS-MOSAIC OUTPUT States ============================================
```

```
      real :: swnet
      real :: lwnet
      real :: qle
      real :: qh
      real :: qg
      real :: snowf
      real :: rainf
      real :: evap
      real :: qs
      real :: qsb
      real :: qsm
      real :: avgsurft
      real :: albedo
      real :: swe
      real :: soilmoist1
      real :: soilmoist2
      real :: soilmoist3
      real :: soilwet
      real :: ecanop
      real :: tveg
      real :: esoil
      real :: rootmoist
      real :: canopint
      real :: acond
      real :: soilm_prev
      real :: swe_prev
      real :: water1
      real :: water2
      real :: water3
      real :: forcing(10)
!      REAL :: AC
!      REAL :: CC
      REAL :: LAT, LON
      end type mosdec
```

1.77.1 mos_output.F90 (Source File: mos_output.F90)

This subroutines sets up methods to write mosaic output

INTERFACE:

```
subroutine mos_output
```

USES:

```
   use lisdrv_module, only : lis, tile, glbgindex
   use mos_varder, only : mosdrv
   use spmdMod, only : masterproc,npes
```

CONTENTS:

```
  if(lis%o%wsingle ==1) then
    print*, "Mosaic not currently able to write output for each variable in a separate file"
    print*, "Please set lis%o%wsingle = 2"
    stop
!-----------------------------------------------------------------
! Writes each output variable to a separate file
!-----------------------------------------------------------------
!!!     if(mod(lis%t%gmt,mosdrv%writeintm).eq.0)then
!!!        do i=1,32
!!!           call noah_singlegather(i,var)
!!!           if(masterproc) then
!!!              call noah_singleout(lis, tile, glbgindex, var, i)
!!!           endif
!!!        enddo
!!!        call noah_totinit()
!!!     endif
  else
!-----------------------------------------------------------------
! Writes bundled output
!-----------------------------------------------------------------
     if(mod(lis%t%gmt,mosdrv%writeintm).eq.0)then
        if(npes > 1 ) then
           call mos_gather()
        endif
        if(masterproc) then
           call mos_almaout()
        endif
        call mos_totinit()
     endif
  endif
```

1.78 Fortran: Module Interface mospardef_module.F90 (Source File: mospardef_module.F90)

This module contains routines that defines MPI derived data types for Mosaic LSM

REVISION HISTORY:

```
   15 Oct 2003; Sujay Kumar  Initial Specification
 #include <misc.h>
```

INTERFACE:

```
       module mospardef_module
```

USES:

```
use mos_module
use mosdrv_module
use spmdMod
```

Routine that defines MPI derived data types for Mosaic

INTERFACE:

```
subroutine def_mospar_struct()
```

1.78.1 mosrst.F90 (Source File: mosrst.F90)

This program reads restart files for Mosaic. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.
REVISION HISTORY: 1 Oct 1999: Jared Entin; Initial code 15 Oct 1999: Paul Houser; Significant F90 Revision 19 Jan 2001: Brian Cosgrove; Added CLOSE statement 15 Mar 2001: Jon Gottschalck; Added option 4 (initialize Mosaic with GDAS forcing) 17 Apr 2001: Jon Gottschalck; Modified option 4 to work with GEOS forcing 05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA initial conditions if necessary. This is controlled with local variable NOAAIC. Normally set to 0 in this subroutine but set to 1 if want to use Dag's NOAA IC's. Changed output directory structure, and commented out if-then check so that directory is always made. 18 Sep 2001: Brian Cosgrove; Changed use of Dag's ICs so that third mosaic soil layer is set to layer 2's soil moisture since layer 3 doesn't have a wilting point 04 Feb 2002: Jon Gottschalck; Added section to set all Koster tiles of veg type 9 (land ice) to have tons of snow so it never melts since the soil/veg parameters are set for bare soil (Koster – personal communication). 05 Feb 2002: Brian Cosgrove; Added a few diagnostic vars to the NOAAIC=1 option 20 Nov 2002: Jon Radakovich; Updated for PSAS temperature assimilation and BC so the forecast bias is included in the restart file. Added conditionals based on startcode=5 (Using spun-up restart so model time comes from card file) and startcode=6 (Restarting a bias correction run). 12 Dec. 2002: Brian Cosgrove; Fixed usage of Wiltpoint variable. Before, Wiltpoint1 and Wiltpoint2 were used in calculation of root zone soil moisture availability...now, only Wiltpoint2 is used since wiltpoint1 is not the correct wilting point needed for the calculation. 14 Jan 2003: Urszula Jambor; Added deallocation statements near end of routine and changed pointer variables to allocatable. 23 Jan 2003: Urszula Jambor; Switch index order of GEOS forcing array. Snow is 12, soil wetness is 13.
==========
RESTART FILE FORMAT(fortran sequential binary): VCLASS,NC,NR,NCH !Veg class,no. columns, no. rows, no.tiles MOS(NCH)==========

INTERFACE:

```
subroutine MOSrst
```

USES:

```
use lisdrv_module, only : lis, grid, tile
use mos_varder, only : mosdrv
```

```
   USE mos_varder      ! MOSAIC tile variables
   use time_manager
   use tile_spmdMod
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Read Active Archive File
!-----------------------------------------------------------------------
  print*,'DBG: mosrst -- in mosrst',' (',iam,')'
  if(masterproc) then
     IF(LIS%O%STARTCODE.EQ.1)THEN
        allocate(tmptile(lis%d%nch))
        OPEN(40,FILE=mosdrv%MOS_RFILE,FORM='unformatted')

        call timemgr_read_restart(40)
!        call timemgr_restart()
!        call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
!        call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
!        call updatetime(lis%t) !Updates LIS variables.
        WRITE(*,*)'MOSAIC Restart File Used: ',mosdrv%MOS_RFILE
        READ(40) VCLASS,NC,NR,NCH  !Veg class, no. columns, no. rows, no. tiles
!-----------------------------------------------------------------------
!   Check for Vegetation Class Conflict
!-----------------------------------------------------------------------
        IF(VCLASS.NE.LIS%P%VCLASS)THEN
           WRITE(*,*)mosdrv%MOS_RFILE,' Vegetation class conflict'
           call endrun
        ENDIF
!-----------------------------------------------------------------------
!   Check for Grid Space Conflict
!-----------------------------------------------------------------------
        IF(NC.NE.LIS%D%LNC.OR.NR.NE.LIS%D%LNR)THEN
           WRITE(*,*)mosdrv%MOS_RFILE,'Grid space mismatch - MOSAIC HALTED'
           call endrun
        ENDIF
!-----------------------------------------------------------------------
! Transfer Restart tile space to LIS tile space
!-----------------------------------------------------------------------
        IF(NCH.NE.LIS%D%NCH)THEN
           WRITE(*,*)'Restart Tile Space Mismatch, Halting..'
           call endrun
        endif
        read(40) mos%ct         !MOSAIC Canopy/Soil Temperature
read(40) mos%qa         !MOSAIC Canopy Humidity
        read(40) mos%ics        !MOSAIC Interception Canopy Storage
        read(40) mos%snow       !MOSAIC Snow Depth
read(40) mos%SoT        !MOSAIC Deep Soil Temperaure
        do l=1,3
```

```
  read(40) tmptile  !MOSAIC Soil Wetness (3 layers)
  mos%SoWET(l)=tmptile
        enddo
        close(40)
        deallocate(tmptile)
     endif
  endif
```

1.78.2 mos_scatter.F90 (Source File: mos_scatter.F90)

Distributes mosaic tiles on to compute nodes

REVISION HISTORY:

```
Aug 2003 ; Jon Gottschalck, Initial Code
```

INTERFACE:

```
subroutine mos_scatter()
```

USES:

```
 use tile_spmdMod
 use mos_varder
 use mospardef_module
```

1.78.3 mos_setup.F90 (Source File: mos_setup.F90)

Complete the setup routines for mosaic

REVISION HISTORY:

```
4 Nov. 1999: Paul Houser; Initial Code
3 Jun  2003: Jon Gottschalck; Modified code
```

INTERFACE:

```
subroutine mos_setup()
```

USES:

```
      use lisdrv_module, only : lis, tile
      use mos_varder
      use sibalb_module
      use spmdMod, only : masterproc, npes
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endif
      call setmosp()
      call mapsib2umd()
      call mos_coldstart()

      do t=1,lis%d%nch
         mos(t)%swnet = 0
         mos(t)%lwnet = 0
         mos(t)%qle = 0
         mos(t)%qh = 0
         mos(t)%qg = 0
         mos(t)%rainf = 0
         mos(t)%snowf = 0
         mos(t)%evap = 0
         mos(t)%qs = 0
         mos(t)%qsm = 0
         mos(t)%qsb = 0
         mos(t)%swe = 0
         mos(t)%soilmoist1 = 0
         mos(t)%soilmoist2 = 0
         mos(t)%soilmoist3 = 0
         mos(t)%soilwet = 0
         mos(t)%ecanop = 0
         mos(t)%tveg = 0
         mos(t)%esoil = 0
         mos(t)%rootmoist = 0
         mos(t)%canopint = 0
         mos(t)%soilm_prev = 0
         mos(t)%swe_prev = 0
         mos(t)%count = 0
         mos(t)%dtcanal = 0
      enddo

#if ( ! defined OPENDAP )
   endif
   if ( npes > 1 ) then
      call mos_scatter
   endif
#endif
```

1.78.4 mos_totinit.F90 (Source File: mos_totinit.F90)

Initialize Mosaic output arrays

REVISION HISTORY:

```
1 Aug 2003  Sujay Kumar  Initial Specification
```

INTERFACE:

```
subroutine mos_totinit()
```

USES:

```
use mos_varder      ! Mosaic LSM module
use tile_spmdMod
use lisdrv_module, only : lis
```

CONTENTS:

```
do t = 1, di_array(iam)
   if(mod(lis%t%gmt,mosdrv%writeintm).eq.0)then
      mos(t)%soilm_prev=mos(t)%water1 + &
           mos(t)%water2 + &
           mos(t)%water3
      mos(t)%swe_prev =  mos(t)%snow
   endif
enddo
do t = 1, di_array(iam)

  mos(t)%swnet = 0
  mos(t)%lwnet = 0
  mos(t)%qle = 0
  mos(t)%qh = 0
  mos(t)%qg = 0
  mos(t)%rainf = 0
  mos(t)%snowf = 0
  mos(t)%evap = 0
  mos(t)%qs = 0
  mos(t)%qsm = 0
  mos(t)%qsb = 0
  mos(t)%swe = 0
  mos(t)%soilmoist1 = 0
  mos(t)%soilmoist2 = 0
  mos(t)%soilmoist3 = 0
  mos(t)%soilwet = 0
  mos(t)%ecanop = 0
  mos(t)%tveg = 0
  mos(t)%esoil = 0
  mos(t)%rootmoist = 0
  mos(t)%canopint = 0
  mos(t)%count = 0
enddo
```

1.78.5 mos_varder.F90 (Source File: mos_varder.F90)

Module for 1-D MOSAIC land model driver variable initialization

REVISION HISTORY:

```
Jun 2003; Jon Gottschalck, Initial Code
```

INTERFACE:

```
module mos_varder
```

USES:

```
use mos_module
use mosdrv_module
use mospardef_module
use tile_spmdMod
```

1.78.6 mos_varder_ini (Source File: mos_varder.F90)

Reads in runtime mos parameters, allocates memory for variables

INTERFACE:

```
subroutine mos_varder_ini(nch)
```

USES:

```
#if ( defined OPENDAP )
  use opendap_module
#endif
```

CONTENTS:

```
  if(masterproc) then
     call readmoscrd(mosdrv)
  endif
  call def_mospar_struct
#if (defined SPMD)
  call MPI_BCAST(mosdrv,1,MPI_MOSDRV_STRUCT, 0, &
       MPI_COMM_WORLD, ierr)
#endif
  if(masterproc) then
    allocate(mos(nch))
  else
    allocate(mos(di_array(iam)))
  endif
  end subroutine mos_varder_ini
```

1.78.7 mos_writerst.F90 (Source File: mos_writerst.F90)

This program writes restart files for Mosaic. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.
REVISION HISTORY: 1 Oct 1999: Jared Entin; Initial code 15 Oct 1999: Paul Houser; Significant F90 Revision 19 Jan 2001: Brian Cosgrove; Added CLOSE statement 15 Mar 2001: Jon Gottschalck; Added option 4 (initialize Mosaic with GDAS forcing) 17 Apr 2001: Jon Gottschalck; Modified option 4 to work with GEOS forcing 05 Sep 2001: Brian Cosgrove; Modified code to use Dag Lohmann's NOAA initial conditions if necessary. This is controlled with local variable NOAAIC. Normally set to 0 in this subroutine but set to 1 if want to use Dag's NOAA IC's. Changed output directory structure, and commented out if-then check so that directory is always made. 18 Sep 2001: Brian Cosgrove; Changed use of Dag's ICs so that third mosaic soil layer is set to layer 2's soil moisture since layer 3 doesn't have a wilting point 04 Feb 2002: Jon Gottschalck; Added section to set all Koster tiles of veg type 9 (land ice) to have tons of snow so it never melts since the soil/veg parameters are set for bare soil (Koster – personal communication). 05 Feb 2002: Brian Cosgrove; Added a few diagnostic vars to the NOAAIC=1 option 20 Nov 2002: Jon Radakovich; Updated for PSAS temperature assimilation and BC so the forecast bias is included in the restart file. Added conditionals based on startcode=5 (Using spun-up restart so model time comes from card file) and startcode=6 (Restarting a bias correction run). 12 Dec. 2002: Brian Cosgrove; Fixed usage of Wiltpoint variable. Before, Wiltpoint1 and Wiltpoint2 were used in calculation of root zone soil moisture availability...now, only Wiltpoint2 is used since wiltpoint1 is not the correct wilting point needed for the calculation. 14 Jan 2003: Urszula Jambor; Added deallocation statements near end of routine and changed pointer variables to allocatable. 23 Jan 2003: Urszula Jambor; Switch index order of GEOS forcing array. Snow is 12, soil wetness is 13. RESTART FILE FORMAT(fortran sequential binary): VCLASS,NC,NR,NCH !Veg class,no. columns, no. rows, no. tiles MOS(NCH)

INTERFACE:

```
 subroutine mos_writerst()
   !uses:
   use lisdrv_module, only : lis,tile
   USE mos_varder      ! Mosaic tile variables
   use time_manager
   use tile_spmdMod
```

CONTENTS:

```
  if(masterproc) then
!-----------------------------------------------------------------------
! Restart Writing (2 files are written = active and archive)
!-----------------------------------------------------------------------
     if((lis%t%gmt.eq.(24-mosdrv%writeintm)) &
          .or.lis%t%endtime.eq.1)then
        allocate(tmptilen(lis%d%nch))
        open(40,file=mosdrv%mos_rfile,form='unformatted') !Active archive restart
        call timemgr_write_restart(40)
        write(40) lis%p%vclass,lis%d%lnc,lis%d%lnr,lis%d%nch  !Veg class, no tiles
        write(40) mos%ct         !MOSAIC Canopy/Soil Temperature
```

```
write(40) mos%qa         !MOSAIC Canopy Humidity
write(40) mos%ics        !MOSAIC Interception Canopy Storage
write(40) mos%snow       !MOSAIC Snow Depth
write(40) mos%SoT        !MOSAIC Deep Soil Temperaure
        do l=1,3
          do t=1,lis%d%nch
            tmptilen(t)=mos(t)%SoWET(l)
          enddo
          write(40) tmptilen  !Mosaic Soil Wetness (3 layers)
        enddo
        close(40)
        write(*,*)'Mosaic Active restart written: ',mosdrv%mos_rfile
        write(unit=temp,fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
            lis%t%da,lis%t%hr
        read(unit=temp,fmt='(10a1)')ftime
        do i=1,10
          if(ftime(i).eq.(' '))ftime(i)='0'
        enddo
        write(unit=temp,fmt='(a4,i3,a5,i4,a1,i4,i2,i2,a6,i3,a1)') &
            '/EXP',lis%o%expcode,'/MOS/',lis%t%yr, &
            '/',lis%t%yr,lis%t%mo, &
            lis%t%da,'/LIS.E',lis%o%expcode,'.'
        read(unit=temp,fmt='(80a1)') (fname(i),i=1,35)
        do i=1,35
          if(fname(i).eq.(' '))fname(i)='0'
        enddo
        write(unit=temp,fmt='(a9)')'mkdir -p '
        read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)
        write(unit=temp,fmt='(a4,i3,a5,i4,a1,i4,i2,i2)') &
            '/EXP',lis%o%expcode,'/MOS/', &
            lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
        read(unit=temp,fmt='(80a1)') (fyrmodir(i),i=1,25)
        do i=1,25
          if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
        enddo

        write(unit=temp,fmt='(a7)')'.MOSrst'
        read(unit=temp,fmt='(80a1)') (fsubs(i),i=1,7)

        write(unit=temp,fmt='(a40)') lis%o%odir
        read(unit=temp,fmt='(80a1)') (fbase(i),i=1,80)
        c=0
        do i=1,80
          if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
        enddo
        write(unit=temp,fmt='(80a1)')(fbase(i),i=1,c),(fname(i),i=1,35), &
                        (ftime(i),i=1,10),(fsubs(i),i=1,7)
        read(unit=temp,fmt='(a80)')filen
```

```
        write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
         (fyrmodir(i),i=1,25)
        read(unit=temp,fmt='(a80)')mkfyrmo

!-----------------------------------------------------------------------
! Archive File Name Generation Complete
! Make the directories for the NOAH restart file
!-----------------------------------------------------------------------
        CALL SYSTEM(MKFYRMO)
!-----------------------------------------------------------------------
! Archive File Name Generation Complete
!-----------------------------------------------------------------------
        open(40,file=filen,status='unknown',form='unformatted')
        write(40) lis%p%vclass,lis%d%lnc,lis%d%lnr,lis%d%nch  !veg class, no tiles
        call timemgr_write_restart(40)
        write(40) mos%ct         !MOSAIC Canopy/Soil Temperature
write(40) mos%qa         !MOSAIC Canopy Humidity
write(40) mos%ics        !MOSAIC Interception Canopy Storage
write(40) mos%snow       !MOSAIC Snow Depth
write(40) mos%SoT        !MOSAIC Deep Soil Temperaure
        do l=1,3
  do t=1,lis%d%nch
   tmptilen(t)=mos(t)%SoWET(l)
  enddo
  write(40) tmptilen !Mosaic Soil Wetness (3 layers)
        enddo
        close(40)

        write(*,*)'mosaic archive restart written: ',filen
        deallocate(tmptilen)
      endif
   endif
   return
```

1.78.8 mos_writevars.F90 (Source File: mos_writestats.F90)

LIS MOS data writer: Writes mos output

REVISION HISTORY:

```
02 Dec 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
subroutine mos_writestats(ftn_stats)
```

USES:

```
   use lisdrv_module, only : lis
   use mos_varder

   implicit none

   integer :: ftn_stats,t
```

CONTENTS:

```
  do t=1,lis%d%glbnch
     if(mos(t)%forcing(1) < 273.15) then
        rainf(t) = 0.0
        snowf(t) = mos(t)%forcing(8)
     else
        rainf(t) = mos(t)%forcing(8)
        snowf(t) = 0.0
     endif
  enddo
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
   call stats(mos%swnet,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SWnet(W/m2)', &
        vmean,vstdev,vmin,vmax
   call stats(mos%lwnet,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'LWnet(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%qle,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'Qle(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%qh,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'Qh(W/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%qg,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'Qg(W/m2)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------------
   call stats(mos%snowf,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Snowf(kg/m2s)',&
```

```
        vmean,vstdev,vmin,vmax
   call stats(mos%rainf,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Rainf(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%evap,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Evap(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%qs,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Qs(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%qsb,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Qsb(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats((mos%water1+ &
        mos%water2+ &
        mos%water3  &
        -mos%soilm_prev)/float(mos%count), &
        lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
   write(ftn_stats,999) 'DelSoilMoist(kg/m2s)', &
        vmean,vstdev,vmin,vmax
   call stats((mos%snow-mos%swe_prev)/float(mos%count), &
        lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
   write(ftn_stats,999) 'DelSWE(kg/m2s)', &
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
   call stats(mos%avgsurft,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'AvgSurfT(K)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%albedo,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Albedo(-)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%swe,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'SWE(kg/m2)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
   call stats(mos%soilmoist1,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
```

```
   write(ftn_stats,999) 'SoilMoist1(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%soilmoist2,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist2(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%soilmoist3,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,999) 'SoilMoist3(kg/m2)',&
        vmean,vstdev,vmin,vmax
   write(ftn_stats,998) 'SoilWet(-)',&
        vmean,vstdev,vmin,vmax
!-----------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------
   call stats(mos%ecanop,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'ECanop(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%tveg,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'TVeg(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%esoil,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'ESoil(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%rootmoist,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'RootMoist(kg/m2)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%canopint,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Canopint(kg/m2s)',&
        vmean,vstdev,vmin,vmax
   call stats(mos%acond,lis%d%udef,lis%d%glbnch,vmean, &
        vstdev,vmin, vmax)
   write(ftn_stats,998) 'Acond(m/s)',&
        vmean,vstdev,vmin,vmax
   if(lis%o%wfor.eq.1) then
      call stats(sqrt(mos%forcing(5)*mos%forcing(5)+ &
           mos%forcing(6)*mos%forcing(6)), &
           lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,999) 'Wind(m/s)', &
           vmean,vstdev,vmin,vmax
      call stats(rainf, &
           lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,998) 'Rainf(kg/m2s)', &
```

```
            vmean,vstdev,vmin,vmax
      call stats(snowf, &
            lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, vmax)
      write(ftn_stats,998) 'Snowf(kg/m2s)', &
            vmean,vstdev,vmin,vmax
      call stats(mos%forcing(1),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'Tair(K)', &
            vmean,vstdev,vmin,vmax
      call stats(mos%forcing(2),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'Qair(kg/kg)', &
            vmean,vstdev,vmin,vmax
      call stats(mos%forcing(7),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'PSurf(Pa)', &
            vmean,vstdev,vmin,vmax
      call stats(mos%forcing(3),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'SWdown (W/m2)', &
            vmean,vstdev,vmin,vmax
      call stats(mos%forcing(4),lis%d%udef,lis%d%glbnch,vmean,vstdev, &
            vmin, vmax)
      write(ftn_stats,999) 'LWdown(W/m2)', &
            vmean,vstdev,vmin,vmax
   endif
998    FORMAT(1X,A18,4E14.3)
999    FORMAT(1X,A18,4F14.3)
```

!ROUTINE : readmoscrd.F90
Routine to read Mosaic specific parameters from the card file.

REVISION HISTORY:

```
15 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readmoscrd(mosdrv)
```

USES:

```
use mosdrv_module
```

1.78.9 setnoahp.F90 (Source File: setmosp.F90)

This subroutine retrieves MOSAIC parameters - Significant F90 revisions below this subroutine will be requiret in the future.

REVISION HISTORY:

```
15 Oct 1999: Paul Houser; Initial Code
31 Jul 2001: Matt Rodell; Updated for new soil parameter definition.
14 Feb 2002: Jon Gottschalck; Added allocated space for AVHRR LAI/DSAI
07 Mar 2002: Brian Cosgrove; Corrected declaration of TEX1 var from real to int
14 Jan 2003: Urszula Jambor; Added conditional to check if need exists
             to allocate for AVHRR LAI/DSAI variables.
```

INTERFACE:

```
SUBROUTINE SETMOSP()
```

USES:

```
  use lisdrv_module, only : lis, tile, gindex
  use mos_varder      ! NOAH tile variables
  use lis_indices_module
#if ( defined OPENDAP )
  use opendap_module
#endif
```

1.78.10 hyssib_albedo.F90: (Source File: hyssib_albedo.F90)

This subroutine takes quarterly surface albedo (snow-free) data and day to interpolate and determine the actual value of the albedo for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the dates of January 31, April 30, July 31, and October 31.

REVISION HISTORY:

```
28 Apr 2002: K. Arsenault, Added NOAH LSM to LDAS, initial code
21 Apr 2004: David Mocko, Conversion from NOAH to HY-SSiB
```

INTERFACE:

```
      subroutine hyssib_albedo
```

USES:

```
      use time_manager
      use hyssib_varder           ! HY-SSiB tile variables
      use time_manager
      use lisdrv_module, only : grid,tile,lis
#if ( defined OPENDAP )
      use opendap_module
#endif
```

CONTENTS:

```
      zeroi=0
      hyssibdrv%hyssib_aflag = 0
!-----------------------------------------------------------------------
! Determine Dates of the quarters in terms of Year (e.g., 1999.3)
!-----------------------------------------------------------------------
      time=lis%t%time
      yr=lis%t%yr
!-----------------------------------------------------------------------
!  January 31
!-----------------------------------------------------------------------
      janda=31
      janmo=01
      call date2time(jan31,doy1,gmt1,yr,janmo,janda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  April 30
!-----------------------------------------------------------------------
      aprda=30
      aprmo=04
      call date2time(apr30,doy1,gmt1,yr,aprmo,aprda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  July 31
!-----------------------------------------------------------------------
      julda=31
      julmo=07
      call date2time(jul31,doy1,gmt1,yr,julmo,julda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
!  October 31
!-----------------------------------------------------------------------
      octda=31
      octmo=10
      call date2time(oct31,doy1,gmt1,yr,octmo,octda,zeroi,zeroi,zeroi)
!-----------------------------------------------------------------------
! Determine which two quarterly albedo files book-end model time.
!-----------------------------------------------------------------------
      if ((time.ge.jan31).and.(time.le.apr30)) then
         qq1="01"
         qq2="02"
         qdif = apr30-jan31
         timdif = time-jan31
         albflag = 1
      elseif ((time.ge.apr30).and.(time.le.jul31)) then
         qq1="02"
         qq2="03"
         qdif = jul31-apr30
         timdif = time-apr30
         albflag = 2
      elseif ((time.ge.jul31).and.(time.le.oct31)) then
         qq1="03"
```

```
         qq2="04"
         qdif = oct31-jul31
         timdif = time-jul31
         albflag = 3
      elseif (time.ge.oct31) then
         qq1="04"
         qq2="01"
         qdif = (jan31+1.0)-oct31
         timdif = time-oct31
         albflag = 4
      elseif (time.lt.jan31) then
         qq1="04"
         qq2="01"
         oct31=oct31-1.0
         qdif = jan31-oct31
         timdif = time-oct31
         albflag = 5
      endif

      if (hyssibdrv%hyssib_albtime.ne.albflag) then
         hyssibdrv%hyssib_albtime = albflag
         hyssibdrv%hyssib_aflag = 1
!-----------------------------------------------------------------------
!  Open the needed two quarterly snow-free albedo files
!-----------------------------------------------------------------------

#if ( defined OPENDAP )
         print*,'MSG: hyssib_alb -- Retrieving ALBEDO file ',&
                trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ1// &
                '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',' (',iam,')'
         call system("opendap_scripts/getalbedo.pl "//ciam//" "// &
                     trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ1// &
                     '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa'     &
                     //" "//cparm_slat//" "//cparm_nlat           &
                     //" "//cparm_wlon//" "//cparm_elon//" "//QQ1)
         print*,'MSG: hyssib_alb -- Retrieving ALBEDO file ', &
                trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ2// &
                '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',' (',iam,')'
         call system("opendap_scripts/getalbedo.pl "//ciam//" "// &
                     trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ2// &
                     '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa'     &
                     //" "//cparm_slat//" "//cparm_nlat           &
                     //" "//cparm_wlon//" "//cparm_elon//" "//QQ2)
#endif
         print*,'MSG: hyssib_alb -- Retrieving ALBEDO file ', &
                trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ1// &
                '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',' (',iam,')'
         OPEN(10,FILE=trim(HYSSIBDRV%HYSSIB_ALBFILE)//'albedo_'//QQ1// &
```

```
              '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',&
              STATUS='OLD',FORM='UNFORMATTED')
         print*,'MSG: hyssib_alb -- Retrieving ALBEDO file ', &
                trim(hyssibdrv%hyssib_albfile)//'albedo_'//QQ2// &
                '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',' (',iam,')'
         OPEN(11,FILE=trim(HYSSIBDRV%HYSSIB_ALBFILE)//'albedo_'//QQ2// &
              '_'//trim(hyssibdrv%hyssib_galbres)//'.bfsa',&
              STATUS='OLD',FORM='UNFORMATTED')

         read(10) value1
         read(11) value2
         close(10)
         close(11)
!-----------------------------------------------------------------------
! Assign quarterly albedo fractions to each tile.
!-----------------------------------------------------------------------
         do i=1,lis%d%nch
            if ((value1(tile(i)%col,tile(i)%row-tnroffset).ne.-9999.000) &
               .and.(value2(tile(i)%col,tile(i)%row-tnroffset)          &
               .ne.-9999.000)) then
               hyssib(i)%albsf1= value1(tile(i)%col,tile(i)%row-tnroffset)
               hyssib(i)%albsf2= value2(tile(i)%col,tile(i)%row-tnroffset)
            endif
         enddo
      endif                      ! End albflag selection
!-----------------------------------------------------------------------
! Assign albedo fractions to each tile and interpolate daily.
!-----------------------------------------------------------------------
      if (hyssibdrv%hyssib_albdchk.ne.lis%t%da) then
         hyssibdrv%hyssib_aflag = 1
         do i=1,lis%d%nch
            if (hyssib(i)%albsf1.ne.-9999.000) then
               valdif(i) = hyssib(i)%albsf2 - hyssib(i)%albsf1
               hyssib(i)%albsf = (timdif*valdif(i)/qdif)+hyssib(i)%albsf1
            endif
         enddo
         hyssibdrv%hyssib_albdchk=lis%t%da

         if (lis%o%wparam.eq.1) then
            allocate(about(lis%d%lnc,lis%d%lnr))
            do i=1,lis%d%nch
               if (grid(i)%lat*1000.ge.lis%d%gridDesc(4).and. &
                   grid(i)%lat*1000.le.lis%d%gridDesc(7).and. &
                   grid(i)%lon*1000.ge.lis%d%gridDesc(5).and. &
                   grid(i)%lon*1000.le.lis%d%gridDesc(8)) then
                  rindex = tile(i)%row - (lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
                           /lis%d%gridDesc(9)
                  cindex = tile(i)%col - (lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
```

```
                        /lis%d%gridDesc(10)
                 about(cindex,rindex) = hyssib(i)%albsf
              endif
           enddo
           open(32,file="albout.bin",form='unformatted')
           write(32) albout
           close(32)
           deallocate(albout)
        endif
     endif                      ! End daily interpolation
     return
```

1.78.11 hyssib_almaout.F90 (Source File: hyssib_almaout.F90)

LIS HY-SSiB data writer: Binary and stat files in ALMA convention

REVISION HISTORY:

```
 4 Nov 1999: Jon Radakovich, Initial Code
28 Apr 2002: Kristi Arsenault, Added SSIB LSM to LDAS
15 Jun 2003: Sujay Kumar, ALMA version
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_almaout()
```

USES:

```
use lisdrv_module, only : lis
use hyssib_varder         ! HY-SSiB-specific variables

implicit none
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
      IF (MOD(LIS%T%GMT,hyssibdrv%WRITEINT).EQ.0) THEN
         hyssibdrv%NUMOUT = hyssibdrv%NUMOUT+1
         WRITE(UNIT=temp1,FMT='(I4,I2,I2)') LIS%T%YR,LIS%T%MO,LIS%T%DA
         READ(UNIT=temp1,FMT='(8A1)') FTIME
         DO I=1,8
            IF (FTIME(I).EQ.(' ')) FTIME(I)='0'
         ENDDO
         WRITE(UNIT=temp1,FMT='(I4)') LIS%T%YR
```

```
         READ(UNIT=temp1,FMT='(8A1)') FTIMEC
         DO I=1,4
            IF (FTIMEC(I).EQ.(' ')) FTIMEC(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A8,I3,A1)') '/LIS.EXP',LIS%O%EXPCODE,'.'
         READ(UNIT=temp1,FMT='(80A1)') (FNAME(I),I=1,12)
         DO I=1,12
            IF (FNAME(I).EQ.(' ')) FNAME(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A40)') LIS%O%ODIR
         READ(UNIT=temp1,FMT='(40A1)') (FBASE(I),I=1,40)
         C=0
         DO I=1,40
            IF (FBASE(I).EQ.(' ').AND.C.EQ.0) C=I-1
         ENDDO

         WRITE(UNIT=temp1,FMT='(A4,I3,A8,I4,A1,I4,I2,I2)') '/EXP', &
               LIS%O%EXPCODE,'/HYSSIB/', &
               LIS%T%YR,'/',LIS%T%YR,LIS%T%MO,LIS%T%DA
         READ(UNIT=temp1,FMT='(80A1)') (FYRMODIR(I),I=1,28)
         DO I=1,28
            IF (FYRMODIR(I).EQ.(' ')) FYRMODIR(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A9)')'mkdir -p '
         READ(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9)

         WRITE(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
                                       (FYRMODIR(I),I=1,28)
         READ(UNIT=temp1,FMT='(A80)') MKFYRMO
         CALL SYSTEM(MKFYRMO)
!-----------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------
         IF (LIS%O%WOUT.EQ.1) THEN
            WRITE(UNIT=FBINNAME, FMT='(I4,I2,I2,I2)') LIS%T%YR,LIS%T%MO, &
                                                      LIS%T%DA,LIS%T%HR
            READ(UNIT=FBINNAME,FMT='(10A1)') FTIMEB
            DO I=1,10
               IF (FTIMEB(I).EQ.(' ')) FTIMEB(I)='0'
            ENDDO
            WRITE(UNIT=FBINNAME,FMT='(A11)') '.HYSSIBgbin'
            READ(UNIT=FBINNAME,FMT='(80A1)') (FSUBGB(I),I=1,11)

            WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
                       (FYRMODIR(I),I=1,28), &
```

```
                      (FNAME(I),I=1,12),(FTIMEB(I),I=1,10), &
                      (FSUBGB(I),I=1,11)
            READ(UNIT=FBINNAME,FMT='(A80)') FILENGB
!-----------------------------------------------------------------------
! Open statistical output file
!-----------------------------------------------------------------------
            IF (hyssibdrv%HYSSIBopen.EQ.0) THEN
               FILE='HYSSIBstats.dat'
               CALL OPENFILE(NAME,LIS%O%ODIR,LIS%O%EXPCODE,FILE)
               IF (LIS%O%STARTCODE.EQ.1) THEN
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='UNKNOWN', &
                          POSITION='APPEND')
               ELSE
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='REPLACE')
               ENDIF
               hyssibdrv%HYSSIBopen=1
            ENDIF

            WRITE(65,996)'     Statistical Summary of HY-SSiB Output for:  ', &
                  LIS%T%MO,'/',LIS%T%DA,'/',LIS%T%YR,LIS%T%HR,':',LIS%T%MN,':', &
                  LIS%T%SS
 996        FORMAT(A47,I2,A1,I2,A1,I4,1X,I2,A1,I2,A1,I2)
            WRITE(65,*)
            WRITE(65,997)
 997        FORMAT(T27,'Mean',T41,'StDev',T56,'Min',T70,'Max')
         endif
         if (lis%o%wout.eq.1) then
            open(58,file=filengb,form='unformatted')
         endif
         if (lis%o%wout.eq.1) then
            call hyssib_binout(58)
         endif
         call hyssib_writestats(65)
         hyssib%count=0           !reset counters

         write(65,*)
         write(65,*)
      endif
```

1.78.12 hyssib_atmdrv.f (Source File: hyssib_atmdrv.F90)

Transfer forcing from grid to tile space

REVISION HISTORY:

```
15 Oct 1999: Paul Houser, Initial Code
```

```
28 Jan 2002: Jon Gottschalck, Added option for different number of
                              forcing variables
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
SUBROUTINE hyssib_f2t(t, forcing)
```

USES:

```
use lisdrv_module, only : lis   ! LIS non-model-specific 1-D variables
use spmdMod
use tile_spmdMod
use hyssib_varder
```

1.78.13 hyssib_gridout.F90 (Source File: hyssib_binout.F90)

LIS HY-SSiB data writer: Writes HY-SSiB output in grid space

REVISION HISTORY:

```
02 Dec 2003, Sujay Kumar, Initial Version
   Feb 2004, David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_binout(ftn)
```

USES:

```
use lisdrv_module, only : lis
use drv_output_mod, only : drv_writevar_bin
use hyssib_varder

implicit none
```

ARGUMENTS:

```
integer :: ftn
```

CONTENTS:

```
do t=1,lis%d%glbnch
   if (hyssib(t)%forcing(1).lt.273.15) then
      rainf(t) = 0.0
      snowf(t) = hyssib(t)%forcing(8)
   else
      rainf(t) = hyssib(t)%forcing(8)
      snowf(t) = 0.0
   endif
enddo
```

```
!-----------------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------------
  hyssib%swnet = hyssib%swnet/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%swnet)

  hyssib%lwnet = (-1)*hyssib%lwnet/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%lwnet)

  hyssib%qle = hyssib%qle/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%qle)

  hyssib%qh = hyssib%qh/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%qh)

  hyssib%qg = hyssib%qg/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%qg)

  hyssib%qf = hyssib%qf/float(hyssib%count)
  call drv_writevar_bin(ftn, hyssib%qf)

  hyssib%qv = hyssib%qv/float(hyssib%count)
  call drv_writevar_bin(ftn, hyssib%qv)

  hyssib%qtau = hyssib%qtau/float(hyssib%count)
  call drv_writevar_bin(ftn, hyssib%qtau)

  hyssib%qa = hyssib%qa/float(hyssib%count)
  call drv_writevar_bin(ftn, hyssib%qa)

  call drv_writevar_bin(ftn, hyssib%delsurfheat)

  call drv_writevar_bin(ftn, hyssib%delcoldcont)

  !---------------------------------------------------------------------------
  ! General Water Balance Components
  !---------------------------------------------------------------------------
  hyssib%snowf = hyssib%snowf/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%snowf)

  hyssib%rainf = hyssib%rainf/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%rainf)

  hyssib%evap = hyssib%evap/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%evap)

  hyssib%qs = hyssib%qs/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%qs)
```

```
hyssib%qrec = hyssib%qrec/float(hyssib%count)
call drv_writevar_bin(ftn, hyssib%qrec)

hyssib%qsb = hyssib%qsb/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%qsb)

hyssib%qsm = hyssib%qsm/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%qsm)

hyssib%qfz = hyssib%qfz/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%qfz)

hyssib%qst = hyssib%qst/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%qst)

call drv_writevar_bin(ftn,hyssib%delsoilmoist)

call drv_writevar_bin(ftn,hyssib%delswe)

call drv_writevar_bin(ftn,hyssib%delintercept)

!-----------------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------------
if (hyssibdrv%STATEVAR_AVG.eq.1) then
   do t = 1,lis%d%glbnch
      if (hyssib(t)%snowtcount.gt.0) then
         hyssib(t)%snowt = hyssib(t)%snowt/float(hyssib(t)%snowtcount)
      else
         hyssib(t)%snowt = lis%d%udef
      endif
      if (hyssib(t)%albedocount.gt.0) then
         hyssib(t)%albedo = hyssib(t)%albedo/float(hyssib(t)%albedocount)
      else
         hyssib(t)%albedo = lis%d%udef
      endif
      if (hyssib(t)%sliqfraccount.gt.0) then
         hyssib(t)%sliqfrac = hyssib(t)%sliqfrac/float(hyssib(t)%sliqfraccount)
      else
         hyssib(t)%sliqfrac = lis%d%udef
      endif
   enddo
   hyssib%vegtc = hyssib%vegtc/float(hyssib%count)
   hyssib%baresoilt = hyssib%baresoilt/float(hyssib%count)
   hyssib%avgsurft = hyssib%avgsurft/float(hyssib%count)
   hyssib%radteff = hyssib%radteff/float(hyssib%count)
   hyssib%swe = hyssib%swe/float(hyssib%count)
```

```
   hyssib%sweveg = hyssib%sweveg/float(hyssib%count)
endif

call drv_writevar_bin(ftn,hyssib%snowt)

call drv_writevar_bin(ftn,hyssib%vegtc)

call drv_writevar_bin(ftn,hyssib%baresoilt)

call drv_writevar_bin(ftn,hyssib%avgsurft)

call drv_writevar_bin(ftn, hyssib%radteff)

call drv_writevar_bin(ftn,hyssib%albedo)

call drv_writevar_bin(ftn,hyssib%swe)

call drv_writevar_bin(ftn, hyssib%sweveg)

!-----------------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------------
if (hyssibdrv%STATEVAR_AVG.eq.1) then
   hyssib%soilmoist1 = hyssib%soilmoist1/float(hyssib%count)
   hyssib%soilmoist2 = hyssib%soilmoist2/float(hyssib%count)
   hyssib%soilmoist3 = hyssib%soilmoist3/float(hyssib%count)
   hyssib%soiltemp = hyssib%soiltemp/float(hyssib%count)
   hyssib%soilwet = hyssib%soilwet/float(hyssib%count)
endif

call drv_writevar_bin(ftn,hyssib%soilmoist1)

call drv_writevar_bin(ftn,hyssib%soilmoist2)

call drv_writevar_bin(ftn,hyssib%soilmoist3)

call drv_writevar_bin(ftn,hyssib%soiltemp)

call drv_writevar_bin(ftn,hyssib%soilwet)

!-----------------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------------
hyssib%potevap = hyssib%potevap/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%potevap)

hyssib%ecanop = hyssib%ecanop/float(hyssib%count)
call drv_writevar_bin(ftn,hyssib%ecanop)
```

```
  hyssib%tveg = hyssib%tveg/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%tveg)

  hyssib%esoil = hyssib%esoil/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%esoil)

  hyssib%rootmoist = hyssib%rootmoist/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%rootmoist)

  hyssib%subsnow = hyssib%subsnow/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%subsnow)

  hyssib%subsurf = hyssib%subsurf/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%subsurf)

  hyssib%acond = hyssib%acond/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%acond)

  hyssib%snowfrac = hyssib%snowfrac/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%snowfrac)
!-----------------------------------------------------------------------
! Cold Season Processes
!-----------------------------------------------------------------------
  hyssib%snowdepth = hyssib%snowdepth/float(hyssib%count)
  call drv_writevar_bin(ftn,hyssib%snowdepth)

  call drv_writevar_bin(ftn,hyssib%sliqfrac)

  if (lis%o%wfor.eq.1) then
     call drv_writevar_bin(ftn,sqrt(hyssib%forcing(5)*hyssib%forcing(5)+ &
          hyssib%forcing(6)*hyssib%forcing(6)))

     call drv_writevar_bin(ftn,rainf)

     call drv_writevar_bin(ftn,snowf)

     call drv_writevar_bin(ftn,hyssib%forcing(1))

     call drv_writevar_bin(ftn,hyssib%forcing(2))

     call drv_writevar_bin(ftn,hyssib%forcing(7))

     call drv_writevar_bin(ftn,hyssib%forcing(3))

     call drv_writevar_bin(ftn,hyssib%forcing(4))

  endif
```

1.78.14 hyssib_coldstart.F90 (Source File: hyssib_coldstart.F90)

Routine for HY-SSiB initialization from cold start

REVISION HISTORY:

```
     Dec 2003: Luis-Gustavo Goncalves, Initial version
  29 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
      subroutine hyssib_coldstart()
```

USES:

```
      use lisdrv_module, only : lis, grid, tile
      use hyssib_varder      ! HY-SSIB tile variables
      use time_manager
      use tile_spmdMod
```

CONTENTS:

```
     if (lis%o%startcode.eq.2) then
        print*,'MSG: hyssib_coldstart -- cold-starting hyssib', &
               '...using ics from card file',' (', iam, ')'
        print*,'DBG: hyssib_coldstart -- nch',lis%d%nch,' (', iam, ')'
        do t=1,lis%d%nch
           HYSSIB(T)%TC          =  hyssibdrv%HYSSIB_IT
           HYSSIB(T)%TG          =  hyssibdrv%HYSSIB_IT
           HYSSIB(T)%TSN         =  hyssibdrv%HYSSIB_IT
           HYSSIB(T)%TD          =  hyssibdrv%HYSSIB_IT
           HYSSIB(T)%WWW(1)      =  hyssibdrv%HYSSIB_ISM
           HYSSIB(T)%WWW(2)      =  hyssibdrv%HYSSIB_ISM
           HYSSIB(T)%WWW(3)      =  hyssibdrv%HYSSIB_ISM
           HYSSIB(T)%CAPAC(1)    =  0.0
           HYSSIB(T)%CAPAC(2)    =  0.0
           HYSSIB(T)%SNOW(1)     =  0.0
           HYSSIB(T)%SNOW(2)     =  0.0
           HYSSIB(T)%SGFG        =  0.0
           HYSSIB(T)%SDENS       =  0.0
        enddo                  ! end of tile loop for card options

        lis%t%yr=lis%t%syr
        lis%t%mo=lis%t%smo
        lis%t%da=lis%t%sda
        lis%t%hr=lis%t%shr
```

```
      lis%t%mn=lis%t%smn
      lis%t%ss=lis%t%sss

      call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr, &
                     lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
      write(*,*) 'MSG: hyssib_coldstart -- Using lis.crd start time ', &
                 lis%t%time, ' (', iam, ')'
   endif

   return
```

1.79 Fortran: Module Interface hyssibdrv_module.f: (Source File: hyssibdrv_module.F90)

Module for runtime specific HY-SSiB variables

REVISION HISTORY:

```
14 Oct 2003: Sujay Kumar, Initial Version
23 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
MODULE hyssibdrv_module
```

CONTENTS:

```
integer :: hyssibopen          !Keeps track of opening files
integer :: numout            !Counts number of output times for HY-SSiB
integer :: hyssib_nvegp        !Number of static vegetation parameter
integer :: hyssib_nvegip       !Number of monthly vegetation parameter
integer :: hyssib_flgres       !Restart flag
integer :: hyssib_nsoilp       !Number of static soil parameters
integer :: hyssib_zst          !Number of Zobler soil classes
integer :: hyssib_gflag        !Time flag to update gfrac files
integer :: hyssib_albtime      !Time flag to update albedo files
integer :: hyssib_aflag        !Time flag to update albedo files
integer :: hyssib_albdchk      !Day check to interpolate alb values
integer :: hyssib_gfracdchk    !Day check to interpolate gfrac value
integer :: STATEVAR_AVG        !HY-SSiB Instantaneous (=0) or Time-Averaged (=1) output
integer :: varid(46)
CHARACTER*40 :: HYSSIB_RFILE   !HY-SSiB Active Restart File
CHARACTER*40 :: HYSSIB_AFILE   !HY-SSiB Albedo and Radiation Parameter File
CHARACTER*40 :: HYSSIB_VFILE   !HY-SSiB Static Vegetation Parameter File
CHARACTER*40 :: HYSSIB_SFILE   !HY-SSiB Soil Parameter File
CHARACTER*40 :: HYSSIB_MGFILE  !HY-SSiB Monthly Veg. Green Frac.
CHARACTER*40 :: HYSSIB_ALBFILE !HY-SSiB Quart. Snow-free albedo
CHARACTER*40 :: HYSSIB_GALBRES !HY-SSiB Quart. Snow-free albedo resolution
```

```
CHARACTER*50 :: HYSSIB_MXSNAL  !HY-SSiB LIS max snow albedo
CHARACTER*50 :: HYSSIB_TBOT    !HY-SSiB LIS Bottom Temp
CHARACTER*50 :: HYSSIB_TOPOSTD !HY-SSiB LIS Standard Dev. of Topography
character*80 :: ofile
REAL*8 :: HYSSIB_GFRACTIME     !Time flag to update gfrac files
REAL :: HYSSIB_ISM             !HY-SSiB Initial Soil Moisture (m3/m3)
REAL :: HYSSIB_IT              !HY-SSiB Initial Soil Temperature (K)
REAL :: WRITEINT              !HY-SSiB Output Interval (hours)
end type hyssibdrvdec
```

1.79.1 hyssib_dynsetup.F90 (Source File: hyssib_dynsetup.F90)

Updates the time dependent HY-SSiB variables

REVISION HISTORY:

```
15 Apr 2002: Sujay Kumar, Initial Specification
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_dynsetup()
```

USES:

```
use lisdrv_module, only: lis,tile
use hyssib_varder
use spmdMod, only : masterproc, npes
use hyssibpardef_module
```

1.79.2 hyssib_gather.F90 (Source File: hyssib_gather.F90)

Gathers HY-SSiB tiles

REVISION HISTORY:

```
Apr 2003: Sujay Kumar, Initial Code
Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_gather()
```

USES:

```
use tile_spmdMod
use hyssib_varder
use hyssibpardef_module
```

1.79.3 hyssib_gfrac.F90 (Source File: hyssib_gfrac.F90)

This subroutine takes vegetation greenness fraction data and the date to interpolate and determine the actual value of the greenness fraction for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the 16th of the given month, at 00Z.
Gustavo will take advantage of this routine to read the other time varying parameters and interpolate them to the appropriate time frame

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added SSiB LSM to LDAS, initial code
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
      SUBROUTINE HYSSIB_GFRAC(LD,LT,TILE)
```

USES:

```
      USE lis_module      ! LIS non-model-specific 1-D variables
      USE hyssib_varder   ! HY-SSIB tile variables
      use time_manager
      use tile_module
 #if ( defined OPENDAP )
      use opendap_module
 #endif
```

1.79.4 hyssib_main.f (Source File: hyssib_main.F90)

********** *** This code is the offline HY-SSiB Model (Hydrology with Simple SiB) Authors: Y.C. Sud and David M. Mocko, NASA Goddard Space Flight Center Code Maintenance by: David Mocko ¡mocko@climate.gsfc.nasa.gov¿
Referred Papers:
SMP1: Sud, Y.C., and D.M. Mocko, 1999: New snow-physics to complement SSiB. Part I: Design and evaluation with ISLSCP Initiative I datasets. J. Meteor. Soc. Japan, Vol. 77, No. 1B, 335-348.
SMP2: Mocko, D.M., Walker, G.K., and Y.C. Sud, 1999: New snow-physics to complement SSiB. Part II: Effects on soil moisture initialization and simulated surface fluxes, precipitation and hydrology of GEOS II GCM. J. Meteor. Soc. Japan, Vol. 77, No. 1B, 349-366.
SMP3: Mocko, D.M., and Y.C. Sud, 2001: Refinements to SSiB with an emphasis on snow-physics: Evaluation and validation using GSWP and Valdai data. Earth Interactions, Vol. 5, (5-001), 31 pp.
Other papers in comments:
Sellers et al (two sets of authors), 1996: J. Climate, Vol. 6, No. 4, A revised land surface parameterization (SiB2) for atmospheric GCMs Part I - Model formulation. p. 676-705. Part II - The generation of global fields of terrestrial biophysical parameters from satellite data. p. 706-737.

GEWEX/GSWP, 1995: International GEWEX Project Office. Version 1.0, 1 Dec 1995. 46 pp.
Douville et al, 1995: A new snow parameterization for the Meteo-France climate model. Parts I and II. Clim. Dyn., Vol. 12, 21-52. Mahrt and Sun, 1995: The subgrid velocity scale in the bulk aerodynamic relationship for spatially averaged scalar fluxes. Mon. Wea. Rev., Vol. 123, No. 10, 3032-3041.
Xue et al, 1991: A simplified biosphere model for global climate studies. J. Climate, Vol. 4, No. 3, 345-364.
Sellers et al, 1986: A simple biosphere model (SiB) for use within general circulation models. J. Atmos Sci., Vol. 43, No. 6, 505-531.
Milly and Eagleson, 1982: Parameterization of moisture and heat fluxes across the land surface for use in atmospheric general circulation models. Dept. of Engineering, Massachusetts Inst. of Technology, Rep. 279, 159 pp.
Deardorff, 1972: Parameterization of planetary boundary-layer for use in general circulation models. Mon. Wea. Rev., Vol. 100, No. 2, 93-106.

REVISION HISTORY:

```
21 Apr 2004: David Mocko, Initial Code
30 Apr 2004: David Mocko, Revisions to fix MPI version
```

INTERFACE:

```
      SUBROUTINE HYSSIB_MAIN()
```

USES:

```
      USE lisdrv_module, only : lis,grid,tile
      USE hyssib_varder      ! HY-SSiB tile variables
      use tile_spmdMod
```

CONTENTS:

```
!      if (first) then
!         print *,'setting HY-SSiB constants for the first time'
! Set constants
         grav = 9.81
         pie = 3.1415926
         cpair = 1004.16
         gasr = 287.04
         hlat = 2499000.
         snomel = 3.336242E+08
         clai = 840.0
         cw = 4200000.
         cs = 2106000.
         dkfac = 25.0
         epsfac = 0.622
         stefan = 5.67000E-08
         thres = 5.0E-03
         sskin = 4.0E-03
         tf = 273.15
```

```
         tf_snow = 273.15
         tf_drain = 272.15
         delsig = 0.012

         OPEN(UNIT=11,file=hyssibdrv%HYSSIB_AFILE,status='old',        &
                      form='unformatted')
         read(11) cedfu, cedir, cedfu1, cedir1, cedfu2, cedir2,        &
                  cledfu,cledir, xmiu, cether, xmiw
         CLOSE(11)
!         first = .false.
!      endif

!=== Convert LIS Timestep varname to HY-SSiB timestep varname (DT) (sec)
      do t = 1,di_array(iam)
         DTT = lis%t%TS

!=== RESET SOME LOCAL ACCUMULATIVE VARS
         RNOFFS = 0.
         SNM = 0.
         SIBSU = 0.
         BEDO = 0
         SOILDIF = 0
         SOILDRA = 0
         RNOFFT = 0
         RNOFFB = 0
         ITER = 1
!===END RESET SOME LOCAL ACCUMULATIVE VARS

!========static parameters
         chil(1)     = HYSSIB(T)%VEGP(1)
         topt(1)     = HYSSIB(T)%VEGP(2)
         tll(1)      = HYSSIB(T)%VEGP(3)
         tu(1)       = HYSSIB(T)%VEGP(4)
         defac(1)    = HYSSIB(T)%VEGP(5)
         ph1(1)      = HYSSIB(T)%VEGP(6)
         ph2(1)      = HYSSIB(T)%VEGP(7)
         rootd(1,1)  = HYSSIB(T)%VEGP(8)
         rootd(1,2)  = HYSSIB(T)%VEGP(9)
         rstpar(1,1) = HYSSIB(T)%VEGP(10)
         rstpar(1,2) = HYSSIB(T)%VEGP(11)
         rstpar(1,3) = HYSSIB(T)%VEGP(12)
         phsat(1)    = HYSSIB(T)%VEGP(13)
         poros(1)    = HYSSIB(T)%VEGP(14)
         bee(1)      = HYSSIB(T)%VEGP(15)
         satco(1)    = HYSSIB(T)%VEGP(16)
         slope(1)    = HYSSIB(T)%VEGP(17)
         zdepth(1,1) = HYSSIB(T)%VEGP(18)
         zdepth(1,2) = HYSSIB(T)%VEGP(19)
```

```
         zdepth(1,3) = HYSSIB(T)%VEGP(20)
         latco(1)    = grid(tile(T)%index)%lat
         lonco(1)    = grid(tile(T)%index)%lon
         wiltp(1)    = (-exp(ph2(1)) / phsat(1)) ** (-1 / bee(1))
         psilow(1)   = phsat(1) * (wiltp(1) ** (-bee(1)))
         topostd(1)  = HYSSIB(T)%tempstd
         tdd(1)      = HYSSIB(T)%tempbot
         visalb(1)   = HYSSIB(T)%albsf
         niralb(1)   = HYSSIB(T)%albsf
         zs(1)       = 0.0

!========monthly parameters
 z0(1)       = HYSSIB(T)%VEGIP(1)
 z1(1)       = HYSSIB(T)%VEGIP(2)
 z2(1)       = HYSSIB(T)%VEGIP(3)
 dd(1)       = HYSSIB(T)%VEGIP(4)
 vcover(1,1) = HYSSIB(T)%VEGIP(5)
 vcover(1,2) = HYSSIB(T)%VEGIP(6)
 zlt(1,1)    = HYSSIB(T)%VEGIP(7)
 zlt(1,2)    = HYSSIB(T)%VEGIP(8)
 green(1)    = HYSSIB(T)%VEGIP(9)
 rbc(1)      = HYSSIB(T)%VEGIP(10)
 rdc(1)      = HYSSIB(T)%VEGIP(11)

!======other parameters (state)
         tc(1)       = HYSSIB(T)%TC
         tg(1)       = HYSSIB(T)%TG
         tsn(1)      = HYSSIB(T)%TSN
         td(1)       = HYSSIB(T)%TD
         gw1(1)      = HYSSIB(T)%WWW(1)
         gw2(1)      = HYSSIB(T)%WWW(2)
         gw3(1)      = HYSSIB(T)%WWW(3)
         capac1(1)   = HYSSIB(T)%CAPAC(1)
         capac2(1)   = HYSSIB(T)%CAPAC(2)
         sn1(1)      = HYSSIB(T)%SNOW(1)
         sn2(1)      = HYSSIB(T)%SNOW(2)
         sgfg(1)     = HYSSIB(T)%SGFG
         sdens(1)    = HYSSIB(T)%SDENS

!======more parameters (state)
         ityp(1)     = int(HYSSIB(T)%VEGT)
         month       = lis%t%mo
         nymd        = int((mod(lis%t%yr,100) * 10000) +               &
                             (lis%t%mo * 100) + lis%t%da)
         nhms        = int((lis%t%hr * 10000) + (lis%t%mn * 100))
         prin = .false.
!         if ((latco(1).eq.46.5).and.(lonco(1).eq.-100.5)) prin = .true.
```

```
!======end of assigning parameters

!=== THE FOLLOWING BREAKS DOWN THE FORCING VARIABLES

         tm(1)     = HYSSIB(t)%FORCING(1)
         sh(1)     = HYSSIB(t)%FORCING(2)
         swdown(1) = amax1(HYSSIB(t)%FORCING(3),0.0)
 lwdown(1) = HYSSIB(t)%FORCING(4)
         UWIND     = (HYSSIB(t)%FORCING(5))*(HYSSIB(t)%FORCING(5))
         VWIND     = (HYSSIB(t)%FORCING(6))*(HYSSIB(t)%FORCING(6))
         spdm(1)   = amax1(SQRT(UWIND + VWIND),0.25)
         ps(1)     = HYSSIB(t)%FORCING(7) / 100.0
         ppl(1)    = HYSSIB(t)%FORCING(8) - HYSSIB(t)%FORCING(9)
         ppc(1)    = HYSSIB(t)%FORCING(9)

         if (prin) then
            print *,' '
            print *,'Latitude ',latco(1),'; Longitude ',lonco(1)
            print *,' '
            print *,'Constants: '
            print *,'tf: ',tf
            print *,' '
            print *,'Forcing data: '
            print *,'tm: ',tm(1)
            print *,'ps: ',ps(1)
            print *,'sh: ',sh(1)
            print *,'spdm: ',spdm(1)
            print *,'ppl: ',ppl(1)
            print *,'ppc: ',ppc(1)
            print *,'swdown: ',swdown(1)
            print *,'lwdown: ',lwdown(1)
            print *,' '
         endif

!=== END OF BREAKING DOWN THE FORCING VARIABLES (SUBROUTINE DRIVER)
!=== ==============================================================
!=== ==============================================================
!=== ============BEGIN OF SIMULATION==============================

         bps(1) = (ps(1) / 1000.) ** (gasr / cpair)
         thm(1) = tm(1) / bps(1)
         ros(1) = (ps(1) * 100.0) / (gasr * tm(1))
         ppl(1) = ppl(1) * dtt
         ppc(1) = ppc(1) * dtt
         psb(1) = ps(1) * delsig
         zb(1) = zs(1) + (100.0 * psb(1) / (ros(1) * grav))

! Call radiation
```

```
         call hyssib_radsplit(istrip,tf,nymd,nhms,month,lonco,          &
              latco,ityp,xmiu,cledir,cledfu,cedir,cedfu,cedir2,cedfu2, &
              cedir1,cedfu1,cether,z1,z2,zlt,vcover,visalb,niralb,tg,  &
              sn1,sn2,satcap,sdens,swdown,lwdown,snowfrac,thermk,extk, &
              cosz,salb,radn,radc3,prin)

! Call the LSM
         call hyssib_routines(istrip,grav,pie,dtt,cpair,gasr,           &
              hlat,snomel,clai,cw,cs,dkfac,epsfac,stefan,thres,sskin,  &
              tf,tf_snow,tf_drain,delsig,lonco,latco,ityp,z2,dd,z0,    &
              phsat,poros,bee,satco,slope,zdepth,vcover,sgfg,sdens,    &
              thermk,extk,cosz,zlt,green,chil,rbc,rdc,topt,tll,tu,     &
              defac,ph1,ph2,rootd,rstpar,satcap,wiltp,psilow,topostd,  &
              tdd,bps,psb,ros,thm,zb,tm,ps,sh,spdm,tg,capac1,sn1,      &
              capac2,sn2,gw1,gw2,gw3,tc,td,tsn,ppl,ppc,radn,radc3,     &
              radt,etmass,hflux,roff,swnet,lwnet,qle,qh,qg,qf,qv,qtau, &
              qa,delsurfheat,delcoldcont,snowf,rainf,evap,qs,qrec,qsb, &
              qsm,qfz,qst,delsoilmoist,delswe,delintercept,snowt,      &
              vegtc,baresoilt,avgsurft,radteff,swe,sweveg,soilmoist1,  &
              soilmoist2,soilmoist3,soiltemp,soilwet,potevap,ecanop,   &
              tveg,esoil,rootmoist,canopint,subsnow,subsurf,acond,     &
              snowdepth,sliqfrac,prin)

!=== ===============END OF SIMULATION==============================
!=== ================================================================
!=== ================================================================

!=== Collect state variables

         HYSSIB(T)%TC       = tc(1)
         HYSSIB(T)%TG       = tg(1)
         HYSSIB(T)%TSN      = tsn(1)
         HYSSIB(T)%TD       = td(1)
         HYSSIB(T)%WWW(1)   = gw1(1)
         HYSSIB(T)%WWW(2)   = gw2(1)
         HYSSIB(T)%WWW(3)   = gw3(1)
         HYSSIB(T)%CAPAC(1) = capac1(1)
         HYSSIB(T)%CAPAC(2) = capac2(1)
         HYSSIB(T)%SNOW(1)  = sn1(1)
         HYSSIB(T)%SNOW(2)  = sn2(1)
         HYSSIB(T)%SGFG     = sgfg(1)
         HYSSIB(T)%SDENS    = sdens(1)

!=== Collect the ALMA output variables

         HYSSIB(T)%swnet        = HYSSIB(T)%swnet        + swnet(1)
         HYSSIB(T)%lwnet        = HYSSIB(T)%lwnet        + lwnet(1)
         HYSSIB(T)%qle          = HYSSIB(T)%qle          + qle(1)
```

```
         HYSSIB(T)%qh           = HYSSIB(T)%qh           + qh(1)
         HYSSIB(T)%qg           = HYSSIB(T)%qg           + qg(1)
         HYSSIB(T)%qf           = HYSSIB(T)%qf           + qf(1)
         HYSSIB(T)%qv           = HYSSIB(T)%qv           + qv(1)
         HYSSIB(T)%qtau         = HYSSIB(T)%qtau         + qtau(1)
         HYSSIB(T)%qa           = HYSSIB(T)%qa           + qa(1)
         HYSSIB(T)%delsurfheat  = HYSSIB(T)%delsurfheat + delsurfheat(1)
         HYSSIB(T)%delcoldcont  = HYSSIB(T)%delcoldcont + delcoldcont(1)
         HYSSIB(T)%snowf        = HYSSIB(T)%snowf        + snowf(1)
         HYSSIB(T)%rainf        = HYSSIB(T)%rainf        + rainf(1)
         HYSSIB(T)%evap         = HYSSIB(T)%evap         + evap(1)
         HYSSIB(T)%qs           = HYSSIB(T)%qs           + qs(1)
         HYSSIB(T)%qrec         = HYSSIB(T)%qrec         + qrec(1)
         HYSSIB(T)%qsb          = HYSSIB(T)%qsb          + qsb(1)
         HYSSIB(T)%qsm          = HYSSIB(T)%qsm          + qsm(1)
         HYSSIB(T)%qfz          = HYSSIB(T)%qfz          + qfz(1)
         HYSSIB(T)%qst          = HYSSIB(T)%qst          + qst(1)
         HYSSIB(T)%delsoilmoist = HYSSIB(T)%delsoilmoist+delsoilmoist(1)
         HYSSIB(T)%delswe       = HYSSIB(T)%delswe       + delswe(1)
         HYSSIB(T)%delintercept = HYSSIB(T)%delintercept+delintercept(1)
! These variables are either instantanous or time-averaged,
! depending on value of STATEVAR_AVG flag in lis.crd namelist.
         if (swdown(1).gt.0.0) then
            albedo(1) = ((radn(1,1,1) * salb(1,1,1)) +                &
                         (radn(1,1,2) * salb(1,1,2)) +                &
                         (radn(1,2,1) * salb(1,2,1)) +                &
                         (radn(1,2,2) * salb(1,2,2))) / swdown(1)
         else
            albedo(1) = LIS%d%UDEF
         endif
         if (hyssibdrv%STATEVAR_AVG.eq.0) then
            if (sgfg(1).gt.0.5) then
               HYSSIB(T)%snowt       = snowt(1)
            else
               HYSSIB(T)%snowt       = LIS%d%UDEF
            endif
            HYSSIB(T)%vegtc          = vegtc(1)
            HYSSIB(T)%baresoilt      = baresoilt(1)
            HYSSIB(T)%avgsurft       = avgsurft(1)
            HYSSIB(T)%radteff        = radteff(1)
            if (swdown(1).gt.0.0) then
               HYSSIB(T)%albedo      = albedo(1)
            else
               HYSSIB(T)%albedo      = LIS%d%UDEF
            endif
            HYSSIB(T)%swe            = swe(1)
            HYSSIB(T)%sweveg         = sweveg(1)
            HYSSIB(T)%soilmoist1     = soilmoist1(1)
```

```
            HYSSIB(T)%soilmoist2     = soilmoist2(1)
            HYSSIB(T)%soilmoist3     = soilmoist3(1)
            HYSSIB(T)%soiltemp       = soiltemp(1)
            HYSSIB(T)%soilwet        = soilwet(1)
            if ((sn2(1)+capac2(1)).gt.0.0) then
               HYSSIB(T)%sliqfrac    = sliqfrac(1)
            else
               HYSSIB(T)%sliqfrac    = LIS%d%UDEF
            endif
         else
            if (sgfg(1).gt.0.5) then
               HYSSIB(T)%snowt       = HYSSIB(T)%snowt       + snowt(1)
               HYSSIB(T)%snowtcount  = HYSSIB(T)%snowtcount  + 1
            endif
            HYSSIB(T)%vegtc          = HYSSIB(T)%vegtc       + vegtc(1)
            HYSSIB(T)%baresoilt     = HYSSIB(T)%baresoilt + baresoilt(1)
            HYSSIB(T)%avgsurft      = HYSSIB(T)%avgsurft  + avgsurft(1)
            HYSSIB(T)%radteff       = HYSSIB(T)%radteff   + radteff(1)
            if (swdown(1).gt.0.0) then
               HYSSIB(T)%albedo      = HYSSIB(T)%albedo      + albedo(1)
               HYSSIB(T)%albedocount = HYSSIB(T)%albedocount + 1
            endif
            HYSSIB(T)%swe            = HYSSIB(T)%swe         + swe(1)
            HYSSIB(T)%sweveg         = HYSSIB(T)%sweveg      + sweveg(1)
            HYSSIB(T)%soilmoist1  = HYSSIB(T)%soilmoist1 + soilmoist1(1)
            HYSSIB(T)%soilmoist2  = HYSSIB(T)%soilmoist2 + soilmoist2(1)
            HYSSIB(T)%soilmoist3  = HYSSIB(T)%soilmoist3 + soilmoist3(1)
            HYSSIB(T)%soiltemp       = HYSSIB(T)%soiltemp  + soiltemp(1)
            HYSSIB(T)%soilwet        = HYSSIB(T)%soilwet   + soilwet(1)
            if ((sn2(1)+capac2(1)).gt.0.0) then
               HYSSIB(T)%sliqfrac    = HYSSIB(T)%sliqfrac  + sliqfrac(1)
               HYSSIB(T)%sliqfraccount = HYSSIB(T)%sliqfraccount + 1
            endif
         endif
! Back to typical output variables
         HYSSIB(T)%potevap      = HYSSIB(T)%potevap      + potevap(1)
         HYSSIB(T)%ecanop       = HYSSIB(T)%ecanop       + ecanop(1)
         HYSSIB(T)%tveg         = HYSSIB(T)%tveg         + tveg(1)
         HYSSIB(T)%esoil        = HYSSIB(T)%esoil        + esoil(1)
         HYSSIB(T)%rootmoist    = HYSSIB(T)%rootmoist    + rootmoist(1)
         HYSSIB(T)%canopint     = HYSSIB(T)%canopint     + canopint(1)
         HYSSIB(T)%subsnow      = HYSSIB(T)%subsnow      + subsnow(1)
         HYSSIB(T)%subsurf      = HYSSIB(T)%subsurf      + subsurf(1)
         HYSSIB(T)%acond        = HYSSIB(T)%acond        + acond(1)
         HYSSIB(T)%snowfrac     = HYSSIB(T)%snowfrac     + snowfrac(1)
         HYSSIB(T)%snowdepth    = HYSSIB(T)%snowdepth    + snowdepth(1)

         HYSSIB(t)%count        = HYSSIB(t)%count        + 1
```

```
      enddo

!>>>  END OF HYSSIB_MAIN <<<<<<<<
      return
```

1.79.5 hyssib_mapvegc.F90 (Source File: hyssib_mapvegc.F90)

This subroutine converts the UMD classes to the SIB classes used by SSIB LSM (v 2.5). (Originally from Dag Lohmann at NCEP)

REVISION HISTORY:

```
28 Apr 2002: K Arsenault, Added SSIB LSM to LDAS
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
SUBROUTINE HYSSIB_MAPVEGC(VEGT)
```

1.80 Fortran: Module Interface hyssib_module.f: (Source File: hyssib_module.F90)

Module for 1-D HY-SSiB land model driver variable specification

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, added SSiB LSM 2.5 code to LDAS
14 Nov 2002: Sujay Kumar, Optimized version for LIS
21 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
MODULE hyssib_module
```

CONTENTS:

```
INTEGER :: ts                 !Timestep (seconds)
INTEGER :: maxt               !Maximum tiles per grid
INTEGER :: HYSSIBVEG          !UMD to HY-SSiB Vegetation Class Index value
INTEGER :: COUNT
INTEGER :: SNOWTCOUNT
INTEGER :: ALBEDOCOUNT
INTEGER :: SLIQFRACCOUNT
INTEGER :: ZOBSOIL(1)  !Zobler Soil Classes (LIS%NCH)

REAL :: VEGP(20)     !Static vegetation parameters,   dim(HY-SSiB_NVEGP)
REAL :: VEGIP(11)    !Interpolated monthly parameters,dim(HY-SSiB_NVEGIP)
REAL :: VEGT         !vegetation type of tile
REAL :: ALBSF1       !Date 1 Snow-Free Albedo Value
```

```
      REAL :: ALBSF2       !Date 2 Snow-Free Albedo Value
      REAL :: ALBSF        !Quarterly Snow-Free Albedo dataset

!=== HY-SSiB-Forcing Variables ==============================================
      REAL :: FORCING(10)        ! TILE FORCING..
!=== HY-SSiB-Constant Variables ==============================================
      REAL :: TEMPBOT
      REAL :: TEMPSTD
!=== HY-SSiB-Initial Variables ==============================================
      REAL :: SGFG
      REAL :: SDENS
!=== HY-SSiB-State Variables ================================================
      REAL :: TC
      REAL :: TG
      REAL :: TSN
      REAL :: TD
      REAL :: WWW(3)
      REAL :: CAPAC(2)
      REAL :: SNOW(2)
!=== HY-SSiB-Output =========================================================
      REAL :: swnet
      REAL :: lwnet
      REAL :: qle
      REAL :: qh
      REAL :: qg
      REAL :: qf
      REAL :: qv
      REAL :: qtau
      REAL :: qa
      REAL :: delsurfheat
      REAL :: delcoldcont
      REAL :: snowf
      REAL :: rainf
      REAL :: evap
      REAL :: qs
      REAL :: qrec
      REAL :: qsb
      REAL :: qsm
      REAL :: qfz
      REAL :: qst
      REAL :: delsoilmoist
      REAL :: delswe
      REAL :: delintercept
      REAL :: snowt
      REAL :: vegtc
      REAL :: baresoilt
      REAL :: avgsurft
      REAL :: radteff
```

```
      REAL :: albedo
      REAL :: swe
      REAL :: sweveg
      REAL :: soilmoist1
      REAL :: soilmoist2
      REAL :: soilmoist3
      REAL :: soiltemp
      REAL :: soilwet
      REAL :: potevap
      REAL :: ecanop
      REAL :: tveg
      REAL :: esoil
      REAL :: rootmoist
      REAL :: canopint
      REAL :: subsnow
      REAL :: subsurf
      REAL :: acond
      REAL :: snowfrac
      REAL :: snowdepth
      REAL :: sliqfrac

      end type hyssibdec
!=== End Variable List ====================================================
```

1.80.1 hyssib_output.F90 (Source File: hyssib_output.F90)

This subroutines sets up methods to write HY-SSiB output

REVISION HISTORY:

```
      Dec 2003: Luis-Gustavo Goncalves, Initial version
      Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
 subroutine hyssib_output
```

USES:

```
   use lisdrv_module, only : lis, tile, glbgindex
   use hyssib_varder, only : hyssibdrv
   use spmdMod, only : masterproc
```

1.81 Fortran: Module Interface hyssibpardef_module.F90 (Source File: hyssibpardef_module.F90)

This module contains routines that defines MPI derived data types for HY-SSiB LSM

REVISION HISTORY:

```
   06 Oct 2003: Sujay Kumar, Initial Specification
      Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
 #include <misc.h>
```

INTERFACE:

```
      module hyssibpardef_module
```

USES:

```
      use hyssib_module
      use hyssibdrv_module
      use spmdMod
```

Routine that defines MPI derived data types for HY-SSiB

INTERFACE:

```
      subroutine def_hyssibpar_struct()
```

!ROUTINE : hyssibrst.F90
This program reads restart files for HY-SSiB. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
   1  Oct 1999: Jared Entin, Initial code
   15 Oct 1999: Paul Houser, Significant F90 Revision
   05 Sep 2001: Brian Cosgrove, Modified code to use Dag Lohmann's NOAA
                initial conditions if necessary.  This is controlled with
                local variable NOAAIC.  Normally set to 0 in this subroutine
                but set to 1 if want to use Dag's NOAA IC's.  Changed output
                directory structure, and commented out if-then check so that
                directory is always made.
   28 Apr 2002: Kristi Arsenault, Added SSIB LSM into LDAS
   28 May 2002: Kristi Arsenault, For STARTCODE=4, corrected SNEQV values
                and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
   30 Oct 2003: Matt Rodell, Added back COL,ROW,FGRD,VEGT to restart files
      Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
   RESTART FILE FORMAT(fortran sequential binary):
    YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
    TILE(NCH)%COL        !Grid Col of Tile
    TILE(NCH)%ROW        !Grid Row of Tile
    TILE(NCH)%FGRD       !Fraction of Grid covered by Tile
    TILE(NCH)%VEGT       !Vegetation Type of Tile
    HYSSIB(NCH)%STATES   !Model States in Tile Space
```

INTERFACE:

```
SUBROUTINE HYSSIBRST
```

USES:

```
use lisdrv_module, only : lis, grid, tile
use hyssib_varder, only : hyssibdrv
use hyssib_varder      ! HY-SSiB tile variables
use time_manager
use tile_spmdMod
```

!ROUTINE : hyssib_scatter.F90
Distributes HY-SSiB tiles on to compute nodes

REVISION HISTORY:

```
Apr 2003: Sujay Kumar, Initial Code
Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_scatter()
```

USES:

```
use tile_spmdMod
use hyssib_varder
use hyssibpardef_module
```

!ROUTINE : hyssib_setup.F90
Complete the setup routines for HY-SSiB

REVISION HISTORY:

```
 4 Nov 1999: Paul Houser, Initial Code
28 Apr 2002: Kristi Arsenault, Modified to SSiB LSM 2.5 code to LDAS
21 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_setup()
```

USES:

```
use lisdrv_module, only : lis,tile
use hyssib_varder
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
!=== End Variable List ===================================================
#if ( ! defined OPENDAP )
      if ( masterproc ) then
#endif

      call sethyssibp()
      call hyssib_gfrac()
      call hyssib_albedo()
      call hyssib_coldstart()

      do t=1,lis%d%nch
         hyssib(t)%swnet = 0
         hyssib(t)%lwnet = 0
         hyssib(t)%qle = 0
         hyssib(t)%qh = 0
         hyssib(t)%qg = 0
         hyssib(t)%qf = 0
         hyssib(t)%qv = 0
         hyssib(t)%qtau = 0
         hyssib(t)%qa = 0
         hyssib(t)%delsurfheat = 0
         hyssib(t)%delcoldcont = 0
         hyssib(t)%snowf = 0
         hyssib(t)%rainf = 0
         hyssib(t)%evap = 0
         hyssib(t)%qs = 0
         hyssib(t)%qrec = 0
         hyssib(t)%qsb = 0
         hyssib(t)%qsm = 0
         hyssib(t)%qfz = 0
         hyssib(t)%qst = 0
         hyssib(t)%delsoilmoist = 0
         hyssib(t)%delswe = 0
         hyssib(t)%delintercept = 0
         hyssib(t)%snowt = 0
         hyssib(t)%vegtc = 0
         hyssib(t)%baresoilt = 0
         hyssib(t)%avgsurft = 0
         hyssib(t)%radteff = 0
         hyssib(t)%albedo = 0
         hyssib(t)%swe = 0
         hyssib(t)%sweveg = 0
         hyssib(t)%soilmoist1 = 0
         hyssib(t)%soilmoist2 = 0
         hyssib(t)%soilmoist3 = 0
         hyssib(t)%soiltemp = 0
         hyssib(t)%soilwet = 0
         hyssib(t)%potevap = 0
```

```
         hyssib(t)%ecanop = 0
         hyssib(t)%tveg = 0
         hyssib(t)%esoil = 0
         hyssib(t)%rootmoist = 0
         hyssib(t)%canopint = 0
         hyssib(t)%subsnow = 0
         hyssib(t)%subsurf = 0
         hyssib(t)%acond = 0
         hyssib(t)%snowfrac = 0
         hyssib(t)%snowdepth = 0
         hyssib(t)%sliqfrac = 0

         hyssib(t)%count = 0
         hyssib(t)%snowtcount = 0
         hyssib(t)%albedocount = 0
         hyssib(t)%sliqfraccount = 0
      enddo

#if ( ! defined OPENDAP )
      endif

      if ( npes > 1 ) then
         call hyssib_scatter
      endif
#endif
      return
```

1.81.1 noah_totinit.F90 (Source File: hyssib_totinit.F90)

Initialize NOAH output arrays

REVISION HISTORY:

```
14 Jun 2002: Sujay Kumar, Initial Specification
21 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_totinit()
```

USES:

```
use hyssib_varder         ! HY-SSiB module
use tile_spmdMod
use lisdrv_module, only : lis
```

CONTENTS:

```
do t = 1,di_array(iam)
   hyssib(t)%swnet = 0
   hyssib(t)%lwnet = 0
   hyssib(t)%qle = 0
   hyssib(t)%qh = 0
   hyssib(t)%qg = 0
   hyssib(t)%qf = 0
   hyssib(t)%qv = 0
   hyssib(t)%qtau = 0
   hyssib(t)%qa = 0
   hyssib(t)%delsurfheat = 0
   hyssib(t)%delcoldcont = 0
   hyssib(t)%snowf = 0
   hyssib(t)%rainf = 0
   hyssib(t)%evap = 0
   hyssib(t)%qs = 0
   hyssib(t)%qrec = 0
   hyssib(t)%qsb = 0
   hyssib(t)%qsm = 0
   hyssib(t)%qfz = 0
   hyssib(t)%qst = 0
   hyssib(t)%delsoilmoist = 0
   hyssib(t)%delswe = 0
   hyssib(t)%delintercept = 0
   hyssib(t)%snowt = 0
   hyssib(t)%vegtc = 0
   hyssib(t)%baresoilt = 0
   hyssib(t)%avgsurft = 0
   hyssib(t)%radteff = 0
   hyssib(t)%albedo = 0
   hyssib(t)%swe = 0
   hyssib(t)%sweveg = 0
   hyssib(t)%soilmoist1 = 0
   hyssib(t)%soilmoist2 = 0
   hyssib(t)%soilmoist3 = 0
   hyssib(t)%soiltemp = 0
   hyssib(t)%soilwet = 0
   hyssib(t)%potevap = 0
   hyssib(t)%ecanop = 0
   hyssib(t)%tveg = 0
   hyssib(t)%esoil = 0
   hyssib(t)%rootmoist = 0
   hyssib(t)%canopint = 0
   hyssib(t)%subsnow = 0
   hyssib(t)%subsurf = 0
   hyssib(t)%acond = 0
   hyssib(t)%snowfrac = 0
   hyssib(t)%snowdepth = 0
```

```
      hyssib(t)%sliqfrac = 0

      hyssib(t)%count = 0
      hyssib(t)%snowtcount = 0
      hyssib(t)%albedocount = 0
      hyssib(t)%sliqfraccount = 0
   enddo
   return
```

!ROUTINE : hyssib_varder.F90
Module for 1-D HY-SSiB land model driver variable initialization

REVISION HISTORY:

```
   Apr 2003: Sujay Kumar, Initial Code
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
    module hyssib_varder
```

1.81.2 hyssib_writerst.F90 (Source File: hyssib_writerst.F90)

This program writes restart files for HY-SSiB. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
1  Oct 1999: Jared Entin, Initial code
15 Oct 1999: Paul Houser, Significant F90 Revision
05 Sep 2001: Brian Cosgrove, Modified code to use Dag Lohmann's NOAA
             initial conditions if necessary.  This is controlled with
             local variable NOAAIC.  Normally set to 0 in this subroutine
             but set to 1 if want to use Dag's NOAA IC's.  Changed output
             directory structure, and commented out if-then check so that
             directory is always made.
28 Apr 2002: Kristi Arsenault, Added SSIB LSM into LDAS
28 May 2002: Kristi Arsenault, For STARTCODE=4, corrected SNEQV values
              and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
14 Jun 2003: Sujay Kumar, Separated the write restart from the original code
30 Oct 2003: Matt Rodell, Added back COL,ROW,FGRD,VEGT write statements
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
RESTART FILE FORMAT(fortran sequential binary):
 YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
 TILE(NCH)%COL        !Grid Col of Tile
 TILE(NCH)%ROW        !Grid Row of Tile
 TILE(NCH)%FGRD       !Fraction of Grid covered by tile
```

```
TILE(NCH)%VEGT       !Vegetation Type of Tile
HYSSIB(NCH)%STATES     !Model States in Tile Space
```

INTERFACE:

```
SUBROUTINE HYSSIB_WRITERST()
```

USES:

```
use lisdrv_module, only : lis,tile
use hyssib_varder         ! HY-SSiB tile variables
use time_manager
use tile_spmdMod
```

1.81.3 hyssib_writestats (Source File: hyssib_writestats.F90)

LIS HY-SSiB data writer: Writes HY-SSiB output in grid space

REVISION HISTORY:

```
02 Dec 2003: Sujay Kumar, Initial Version
21 Apr 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
subroutine hyssib_writestats(ftn_stats)
```

USES:

```
use lisdrv_module, only : lis
use hyssib_varder

implicit none
```

ARGUMENTS:

```
integer ::ftn_stats
```

CONTENTS:

```
do t=1,lis%d%glbnch
   if (hyssib(t)%forcing(1).lt.273.15) then
      rainf_in(t) = 0.0
      snowf_in(t) = hyssib(t)%forcing(8)
   else
      rainf_in(t) = hyssib(t)%forcing(8)
      snowf_in(t) = 0.0
   endif
enddo

!-----------------------------------------------------------------------
! General Energy Balance Components
```

```
!-----------------------------------------------------------------------

  call stats(hyssib%swnet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SWnet(W/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%lwnet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'LWnet(W/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qle,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qle(W/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qh,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qh(W/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qg(W/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%qf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qf(W/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%qv,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qv(W/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%qtau,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qtau(N/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%qa,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Qa(W/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%delsurfheat,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'DelSurfHeat(W/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%delcoldcont,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'DelColdCont(W/m2)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------

  call stats(hyssib%snowf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Snowf(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%rainf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
```

```
  write(ftn_stats,998) 'Rainf(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%evap,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Evap(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qs,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qs(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qrec,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qrec(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qsb,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qsb(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qsm,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qsm(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qfz,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qfz(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%qst,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'Qst(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%delsoilmoist,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'DelSoilMoist(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%delswe,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'DelSWE(kg/m2)',vmean,vstdev,vmin,vmax


  call stats(hyssib%delintercept,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'DelIntercept(kg/m2)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------
  if (hyssibdrv%STATEVAR_AVG.eq.1) then
     do t = 1,lis%d%glbnch
        if (hyssib(t)%snowtcount.gt.0) then
           hyssib(t)%snowt = hyssib(t)%snowt/float(hyssib(t)%snowtcount)
        else
           hyssib(t)%snowt = lis%d%udef
        endif
        if (hyssib(t)%albedocount.gt.0) then
           hyssib(t)%albedo = hyssib(t)%albedo/float(hyssib(t)%albedocount)
        else
           hyssib(t)%albedo = lis%d%udef
        endif
```

```
        if (hyssib(t)%sliqfraccount.gt.0) then
           hyssib(t)%sliqfrac = hyssib(t)%sliqfrac/float(hyssib(t)%sliqfraccount)
        else
           hyssib(t)%sliqfrac = lis%d%udef
        endif
     enddo
     hyssib%vegtc = hyssib%vegtc/float(hyssib%count)
     hyssib%baresoilt = hyssib%baresoilt/float(hyssib%count)
     hyssib%avgsurft = hyssib%avgsurft/float(hyssib%count)
     hyssib%radteff = hyssib%radteff/float(hyssib%count)
     hyssib%swe = hyssib%swe/float(hyssib%count)
     hyssib%sweveg = hyssib%sweveg/float(hyssib%count)
  endif

  call stats(hyssib%snowt,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SnowT(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%vegtc,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'VegT(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%baresoilt,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'BaresoilT(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%avgsurft,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'AvgSurfT(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%radteff,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'RadT(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%albedo,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'Albedo(-)',vmean,vstdev,vmin,vmax

  call stats(hyssib%swe,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SWE(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%sweveg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SWEVeg(kg/m2)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------
  if (hyssibdrv%STATEVAR_AVG.eq.1) then
     hyssib%soilmoist1 = hyssib%soilmoist1/float(hyssib%count)
     hyssib%soilmoist2 = hyssib%soilmoist2/float(hyssib%count)
     hyssib%soilmoist3 = hyssib%soilmoist3/float(hyssib%count)
     hyssib%soiltemp = hyssib%soiltemp/float(hyssib%count)
     hyssib%soilwet = hyssib%soilwet/float(hyssib%count)
  endif
```

```
  call stats(hyssib%soilmoist1,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SoilMoist1(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%soilmoist2,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SoilMoist2(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%soilmoist3,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SoilMoist3(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%soiltemp,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SoilTemp(K)',vmean,vstdev,vmin,vmax

  call stats(hyssib%soilwet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SoilWet(-)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------
  call stats(hyssib%potevap,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'PotEvap(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%ecanop,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'ECanop(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%tveg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'TVeg(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%esoil,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'ESoil(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%rootmoist,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'RootMoist(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%canopint,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'CanopInt(kg/m2)',vmean,vstdev,vmin,vmax

  call stats(hyssib%subsnow,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'SubSnow(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%subsurf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,998) 'SubSurf(kg/m2s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%acond,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'ACond(m/s)',vmean,vstdev,vmin,vmax

  call stats(hyssib%snowfrac,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SnowFrac(-)',vmean,vstdev,vmin,vmax
```

```
!-----------------------------------------------------------------------
! Cold Season Processes
!-----------------------------------------------------------------------
  call stats(hyssib%snowdepth,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SnowDepth(m)',vmean,vstdev,vmin,vmax

  call stats(hyssib%sliqfrac,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
  write(ftn_stats,999) 'SliqFrac(-)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Forcing Variables
!-----------------------------------------------------------------------
  if (lis%o%wfor.eq.1) then
     call stats(sqrt(hyssib%forcing(5)*hyssib%forcing(5)+ &
          hyssib%forcing(6)*hyssib%forcing(6)), &
          lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'Wind(m/s)',vmean,vstdev,vmin,vmax

     call stats(rainf_in,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,998) 'Rainf(kg/m2s)',vmean,vstdev,vmin,vmax

     call stats(snowf_in,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,998) 'Snowf(kg/m2s)',vmean,vstdev,vmin,vmax

     call stats(hyssib%forcing(1),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'Tair(K)',vmean,vstdev,vmin,vmax

     call stats(hyssib%forcing(2),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'Qair(kg/kg)',vmean,vstdev,vmin,vmax

     call stats(hyssib%forcing(7),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'PSurf(Pa)',vmean,vstdev,vmin,vmax

     call stats(hyssib%forcing(3),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'SWdown (W/m2)',vmean,vstdev,vmin,vmax

     call stats(hyssib%forcing(4),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999) 'LWdown(W/m2)',vmean,vstdev,vmin,vmax
  endif

998 FORMAT(1X,A18,4E14.3)
999 FORMAT(1X,A18,4F14.3)
  return
```

!ROUTINE : readhyssibcrd.F90

Routine to read HY-SSiB specific parameters from the card file.

REVISION HISTORY:

```
14 Oct 2003: Sujay Kumar, Initial Code
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
      subroutine readhyssibcrd(hyssibdrv)
```

USES:

```
      use hyssibdrv_module
```

1.81.4 sethyssibp.f (Source File: sethyssibp.F90)

This subroutine retrieves HY-SSiB parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added SSiB LSM, Initial Code
13 Oct 2003: Sujay Kumar, Domain independent modifications
   Feb 2004: David Mocko, Conversion from SSiB to HY-SSiB
```

INTERFACE:

```
      SUBROUTINE SETHYSSIBP(LD,LP,tile)
#include "absoft.h"
```

USES:

```
      use lis_module      ! LIS non-model-specific 1-D variables
      use hyssib_varder   ! HY-SSiB tile variables
      use tile_module
#if ( defined OPENDAP )
      use opendap_module
#endif
      implicit none
```

ARGUMENTS:

```
      type (lisdomain) LD
      type (lisparameters) LP
      type (tiledec) tile(LD%NCH)
```

1.81.5 readssibcrd.F90 (Source File: readssibcrd.F90)

Routine to read SSiB specific parameters from the card file.

REVISION HISTORY:

```
14 Oct 2003: Sujay Kumar, Initial Code
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine readssibcrd(ssibdrv)
```

USES:

```
use ssibdrv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=ssib)
print *,'Running SSIB LSM:'
print *,'SSIB Active Restart File: ', ssibdrv%SSIB_RFILE
ssibdrv%ssib_gfractime = 0.0
ssibdrv%ssib_albtime = 0
ssibdrv%ssib_albdchk = 0
ssibdrv%ssib_gfracdchk = 0
ssibdrv%SSIBOPEN=0

close(11)
return
```

1.81.6 setssibp.f (Source File: setssibp.F90)

This subroutine retrieves SSiB parameters - Significant F90 revisions below this subroutine will be required in the future.

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, Added SSiB LSM, Initial Code
13 Oct 2003: Sujay Kumar, Domain independent modifications
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine setssibp
```

USES:

```
      use lisdrv_module, only : grid,tile,lis
      use ssib_varder     ! SSiB tile variables
#if ( defined OPENDAP )
      use opendap_module
#endif
```

1.81.7 ssib_almaout.F90 (Source File: ssib_almaout.F90)

LIS SSiB data writer: Binary and stat files in ALMA convention

REVISION HISTORY:

```
   4 Nov 1999: Jon Radakovich, Initial Code
  28 Apr 2002: Kristi Arsenault, Added SSIB LSM to LDAS
  15 Jun 2003: Sujay Kumar, ALMA version
   1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
  22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_almaout()
```

USES:

```
  use ssib_varder           ! SSiB-specific variables
  use lisdrv_module, only : lis, tile, gindex

  implicit none
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
      IF (MOD(LIS%T%GMT,ssibdrv%WRITEINTN).EQ.0) THEN
         ssibdrv%NUMOUTNH=ssibdrv%NUMOUTNH+1
         WRITE(UNIT=temp1,FMT='(I4,I2,I2)') LIS%T%YR,LIS%T%MO,LIS%T%DA
         READ(UNIT=temp1,FMT='(8A1)') FTIME
         DO I=1,8
            IF (FTIME(I).EQ.(' ')) FTIME(I)='0'
         ENDDO
         WRITE(UNIT=temp1,FMT='(I4)') LIS%T%YR
         READ(UNIT=temp1,FMT='(8A1)') FTIMEC
         DO I=1,4
            IF (FTIMEC(I).EQ.(' ')) FTIMEC(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A8,I3,A1)') '/LIS.EXP',LIS%O%EXPCODE,'.'
         READ(UNIT=temp1,FMT='(80A1)') (FNAME(I),I=1,12)
```

```
         DO I=1,12
            IF (FNAME(I).EQ.(' ')) FNAME(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A40)') LIS%O%ODIR
         READ(UNIT=temp1,FMT='(40A1)') (FBASE(I),I=1,40)
         C=0
         DO I=1,40
            IF (FBASE(I).EQ.(' ').AND.C.EQ.0) C=I-1
         ENDDO

         WRITE(UNIT=temp1,FMT='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
               LIS%O%EXPCODE,'/SSIB/', &
               LIS%T%YR,'/',LIS%T%YR,LIS%T%MO,LIS%T%DA
         READ(UNIT=temp1,FMT='(80A1)') (FYRMODIR(I),I=1,26)
         DO I=1,26
            IF (FYRMODIR(I).EQ.(' ')) FYRMODIR(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A9)')'mkdir -p '
         READ(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9)

         WRITE(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
                                       (FYRMODIR(I),I=1,26)
         READ(UNIT=temp1,FMT='(A80)') MKFYRMO
         CALL SYSTEM(MKFYRMO)
!-----------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------
         IF (LIS%O%WOUT.EQ.1) THEN
            WRITE(UNIT=FBINNAME,FMT='(I4,I2,I2,I2)') LIS%T%YR,LIS%T%MO, &
                                                     LIS%T%DA,LIS%T%HR
            READ(UNIT=FBINNAME,FMT='(10A1)') FTIMEB
            DO I=1,10
               IF (FTIMEB(I).EQ.(' ')) FTIMEB(I)='0'
            ENDDO
            if(lis%o%wout.eq.1) then
               WRITE(UNIT=FBINNAME,FMT='(A9)') '.SSIBgbin'
               READ(UNIT=FBINNAME,FMT='(80A1)') (FSUBGB(I),I=1,9)
            endif
            WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
                      (FYRMODIR(I),I=1,26), &
                      (FNAME(I),I=1,12),(FTIMEB(I),I=1,10), &
                      (FSUBGB(I),I=1,9)
            READ(UNIT=FBINNAME,FMT='(A80)') FILENGB
!-----------------------------------------------------------------------
! Open statistical output file
!-----------------------------------------------------------------------
```

```
            IF (ssibdrv%SSIBopen.EQ.0) THEN
               FILE='SSIBstats.dat'
               CALL OPENFILE(NAME,LIS%O%ODIR,LIS%O%EXPCODE,FILE)
               IF (LIS%O%STARTCODE.EQ.1) THEN
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='UNKNOWN', &
                          POSITION='APPEND')
               ELSE
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='REPLACE')
               ENDIF
               ssibdrv%SSIBopen=1
            ENDIF

            WRITE(65,996)'       Statistical Summary of SSiB Output for:  ', &
                  LIS%T%MO,'/',LIS%T%DA,'/',LIS%T%YR,LIS%T%HR,':',LIS%T%MN,':', &
                  LIS%T%SS
 996        FORMAT(A47,I2,A1,I2,A1,I4,1X,I2,A1,I2,A1,I2)
            WRITE(65,*)
            WRITE(65,997)
 997        FORMAT(T27,'Mean',T41,'StDev',T56,'Min',T70,'Max')
         ENDIF

         ftn = 58
         if(lis%o%wout.eq.1) then
            open(ftn,file=filengb,form='unformatted')
            call ssib_binout(ftn)
            close(ftn)
         endif
         call ssib_writestats(65)
         write(65,*)
         write(65,*)
      ssib%count=0
      ssib%albedocount=0
   endif
   return
```

1.81.8 ssib_atmdrv.f (Source File: ssib_atmdrv.F90)

Transfer forcing from grid to tile space

REVISION HISTORY:

```
15 Oct 1999: Paul Houser, Initial Code
28 Jan 2002: Jon Gottschalck, Added option for different number of
                              forcing variables
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_f2t(t, forcing)
```

USES:

```
use lisdrv_module, only : lis   ! LIS non-model-specific 1-D variables
use spmdMod
use tile_spmdMod
use ssib_varder
```

CONTENTS:

```
DO F=1,lis%f%NFORCE
   ssib(t)%forcing(f)=forcing(f)
enddo
return
```

1.81.9 ssib_gridout.F90 (Source File: ssib_binout.F90)

LIS SSiB data writer: Writes SSiB output in grid space

REVISION HISTORY:

```
02 Dec 2003: Sujay Kumar, Initial Version
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_binout(ftn)
```

USES:

```
use lisdrv_module, only : lis
use drv_output_mod, only : drv_writevar_bin
use ssib_varder

implicit none
```

ARGUMENTS:

```
integer :: ftn
```

CONTENTS:

```
do t=1,lis%d%glbnch
   if (ssib(t)%forcing(1).lt.273.15) then
      rainf(t) = 0.0
      snowf(t) = ssib(t)%forcing(8)
   else
```

```
            rainf(t) = ssib(t)%forcing(8)
            snowf(t) = 0.0
         endif
      enddo

!-----------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------
      ssib%swnet = ssib%swnet/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%swnet)

      ssib%lwnet = ssib%lwnet/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%lwnet)

      ssib%qle = ssib%qle/float(ssib%count)
      call drv_writevar_bin(ftn,ssib%qle)

      ssib%qh = ssib%qh/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qh)

      ssib%qg = ssib%qg/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qg)

      ssib%qf = ssib%qf/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qf)

      ssib%qtau = ssib%qtau/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qtau)

      call drv_writevar_bin(ftn, ssib%delsurfheat)
!-----------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------
      ssib%snowf = ssib%snowf/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%snowf)

      ssib%rainf = ssib%rainf/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%rainf)

      ssib%evap = ssib%evap/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%evap)

      ssib%qs = ssib%qs/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qs)

      ssib%qsb = ssib%qsb/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qsb)
```

```
      ssib%qsm = ssib%qsm/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%qsm)

      call drv_writevar_bin(ftn, ssib%delsoilmoist)

      call drv_writevar_bin(ftn, ssib%delswe)

      call drv_writevar_bin(ftn, ssib%delintercept)

!-----------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------
      if (ssibdrv%STATEVAR_AVG.eq.1) then
         do t = 1,lis%d%glbnch
            if (ssib(t)%albedocount.gt.0) then
               ssib(t)%albedo = ssib(t)%albedo/float(ssib(t)%albedocount)
            else
               ssib(t)%albedo = lis%d%udef
            endif
         enddo
         ssib%vegtc = ssib%vegtc/float(ssib%count)
         ssib%baresoilt = ssib%baresoilt/float(ssib%count)
         ssib%avgsurft = ssib%avgsurft/float(ssib%count)
         ssib%radteff = ssib%radteff/float(ssib%count)
         ssib%swe = ssib%swe/float(ssib%count)
         ssib%sweveg = ssib%sweveg/float(ssib%count)
      endif

      call drv_writevar_bin(ftn, ssib%vegtc)

      call drv_writevar_bin(ftn, ssib%baresoilt)

      call drv_writevar_bin(ftn, ssib%avgsurft)

      call drv_writevar_bin(ftn, ssib%radteff)

      call drv_writevar_bin(ftn, ssib%albedo)

      call drv_writevar_bin(ftn, ssib%swe)

      call drv_writevar_bin(ftn, ssib%sweveg)

!-----------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------
      ssib%soilmoist1 = ssib%soilmoist1/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%soilmoist1)
```

```
      ssib%soilmoist2 = ssib%soilmoist2/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%soilmoist2)

      ssib%soilmoist3 = ssib%soilmoist3/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%soilmoist3)

      ssib%soiltemp = ssib%soiltemp/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%soiltemp)

      ssib%soilwet = ssib%soilwet/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%soilwet)

!-----------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------
      ssib%ecanop = ssib%ecanop/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%ecanop)

      ssib%tveg = ssib%tveg/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%tveg)

      ssib%esoil = ssib%esoil/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%esoil)

      ssib%rootmoist = ssib%rootmoist/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%rootmoist)

      ssib%canopint = ssib%canopint/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%canopint)

      ssib%acond = ssib%acond/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%acond)

      ssib%snowfrac = ssib%snowfrac/float(ssib%count)
      call drv_writevar_bin(ftn, ssib%snowfrac)

!-----------------------------------------------------------------------
! Cold Season Processes
!-----------------------------------------------------------------------

!-----------------------------------------------------------------------
! Forcing Variables
!-----------------------------------------------------------------------
      if (lis%o%wfor.eq.1) then
         call drv_writevar_bin(ftn, sqrt(ssib%forcing(5)*ssib%forcing(5)+ &
                        ssib%forcing(6)*ssib%forcing(6)))

         call drv_writevar_bin(ftn, rainf)
```

```
         call drv_writevar_bin(ftn, snowf)

         call drv_writevar_bin(ftn, ssib%forcing(1))

         call drv_writevar_bin(ftn, ssib%forcing(2))

         call drv_writevar_bin(ftn, ssib%forcing(7))

         call drv_writevar_bin(ftn, ssib%forcing(3))

         call drv_writevar_bin(ftn, ssib%forcing(4))

      endif

 998  FORMAT(1X,A18,4E14.3)
 999  FORMAT(1X,A18,4F14.3)
      return
```

1.81.10 ssib_coldstart.F90 (Source File: ssib_coldstart.F90)

Routine for SSiB initialization from cold start

REVISION HISTORY:

```
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   18 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
       subroutine ssib_coldstart
```

USES:

```
       use lisdrv_module, only : lis, grid, tile,gindex
       use ssib_varder      ! SSIB tile variables
       use time_manager
       use tile_spmdMod
```

CONTENTS:

```
      if (ssibdrv%SSIB_FLGRES.eq.0) then
      if (lis%o%startcode.eq.2) then
         print*,'MSG: ssib_coldstart -- cold-starting ssib', &
                '...using ics from card file',' (', iam, ')'
         print*,'DBG: ssib_coldstart -- nch',lis%d%nch,' (', iam, ')'
!       write(*,*)lis%d%nch,lis%d%lnc*lis%d%lnr
         varfield=0.0
```

```
!       open(1,file='/u2/gustavo/LIS2.3/LIS/tmp/INITIAL/init.txt',&
!       status='unknown')
          do io=1,5
!          read(1,*) varfield
             do c = 1,lis%d%lnc
                do r = 1,lis%d%lnr
                   if (gindex(c,r).ne.-1) then
                      vecvarfield(io,gindex(c,r)) = varfield(c,r)
                   endif
                   if (vecvarfield(io,gindex(c,r)).le.9.9989996E+20) then
                      if(io.eq.1)then
                         vecvarfield(io,gindex(c,r)) = ssibdrv%SSIB_ISM
                      endif
                      if(io.eq.2)then
                         vecvarfield(io,gindex(c,r)) = ssibdrv%SSIB_ISM
                      endif
                      if(io.eq.3)then
                         vecvarfield(io,gindex(c,r)) = ssibdrv%SSIB_IT
                      endif
                      if(io.eq.4)then
                         vecvarfield(io,gindex(c,r)) = ssibdrv%SSIB_IT
                      endif
                      if(io.eq.5)then
                         vecvarfield(io,gindex(c,r)) = ssibdrv%SSIB_IT
                      endif
                   endif
                enddo
             enddo
          enddo
!       CLOSE(1)

          do t=1,lis%d%nch
             SSIB(T)%TMINI       = vecvarfield(5,t) ! TC=TA=TM
             SSIB(T)%TCINI       = vecvarfield(5,t) ! TC=TA=TM
             SSIB(T)%TAINI       = vecvarfield(5,t) ! TC=TA=TM
             SSIB(T)%HTINI       = 0.0 ! no need to specify
             SSIB(T)%QAINI       = 0.0 ! no need to specify
             SSIB(T)%TGSINI      = vecvarfield(3,t) ! top soil temperature
             SSIB(T)%TDINI       = vecvarfield(4,t) ! deep soil temperature
             SSIB(T)%WWWINI(1)   = vecvarfield(1,t) ! top soil moisture
             SSIB(T)%WWWINI(2)   = vecvarfield(2,t) ! 2 layer soil moisture
             SSIB(T)%WWWINI(3)   = vecvarfield(2,t) ! 3 layer soil moisture
             SSIB(T)%CAPACINI(1) = 0.0
             SSIB(T)%CAPACINI(2) = 0.0
          enddo                  ! end of tile loop for card options

          lis%t%yr=lis%t%syr
          lis%t%mo=lis%t%smo
```

```
         lis%t%da=lis%t%sda
         lis%t%hr=lis%t%shr
         lis%t%mn=lis%t%smn
         lis%t%ss=lis%t%sss

         call date2time(lis%t%time,lis%t%doy,lis%t%gmt,lis%t%yr, &
                        lis%t%mo,lis%t%da,lis%t%hr,lis%t%mn,lis%t%ss)
         write(*,*) 'MSG: ssib_coldstart -- Using lis.crd start time ',&
                    lis%t%time, ' (', iam, ')'
      endif
      endif

      if (ssibdrv%SSIB_FLGRES.eq.1) then
         if (LIS%O%STARTCODE.EQ.2) then
            allocate(tmptile(lis%d%nch))
            OPEN(40,FILE=ssibdrv%SSIB_RFILE,FORM='unformatted')

            call timemgr_read_restart(40)
!        call timemgr_restart()
!        call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
!        call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
!        call updatetime(lis%t) !Updates LIS variables.

            WRITE(*,*)'SSIB Restart File Used: ',ssibdrv%SSIB_RFILE

            READ(40) VCLASS,lnc,lnr,NCH !Time, veg class, no. tiles

!   Check for Vegetation Class Conflict
            IF (VCLASS.NE.LIS%P%VCLASS) THEN
               WRITE(*,*) ssibdrv%SSIB_RFILE, &
                          'Vegetation class conflict'
               STOP
            ENDIF

!   Check for Grid Space Conflict
            IF (lnc.NE.LIS%D%lnc.OR.lnr.NE.LIS%D%lnr) THEN
               WRITE(*,*) ssibdrv%SSIB_RFILE, &
                          'Grid space mismatch - SSiB HALTED'
               STOP
            ENDIF

!-- Transfer Restart tile space to LIS tile space
            IF (NCH.NE.LIS%D%NCH) THEN
               WRITE(*,*)'Restart Tile Space Mismatch, Halting..'
               stop
            endif

!-- Allocate temporary arrays.
```

```
            allocate(TMP_COL(NCH))
            allocate(TMP_ROW(NCH))
            allocate(TMP_FGRD(NCH))
            allocate(TMP_VEGT(NCH))
            allocate(SIB_VEGT(NCH))

!-- NOTE: Next four read statements originally removed from LIS
            READ(40) TMP_COL    !Grid Col of Tile
            READ(40) TMP_ROW    !Grid Row of Tile
            READ(40) TMP_FGRD   !Fraction of Grid covered by tile
            READ(40) TMP_VEGT   !Vegetation Type of Tile

!*** TESTING
!        open(unit=31, file='test.arrays.ssibrst.txt', &
!      form='formatted')
! write(31,*) '  COL TCOL  ROW TROW  VEGT TVEGT'
! do t=1,nch
!   write(31,310) TILE(t)%COL,TMP_COL(t),TILE(t)%ROW,TMP_ROW(t),&
!         TILE(t)%VEGT,TMP_VEGT(t)
! enddo
!  310   format(6i5)
!   close(31)

!-- Convert maketiles UMD vegetation classes to Noah SiB classes before
!--  testing for tile definition conflict.
            SIB_VEGT = TILE%VEGT
            do t=1,nch
               call SSIB_MAPVEGC(SIB_VEGT(t))
            enddo

!-- Check for tile definition conflict
            if ((sum(TMP_COL - TILE%COL) .ne. 0) .or. &
                (sum(TMP_ROW - TILE%ROW) .ne. 0) .or. &
                (sum(TMP_VEGT - SIB_VEGT) .ne. 0)) then
               write(*,*) sum(TMP_vegt),sum(sib_vegt)
               write(*,*) 'Restart tile definition mismatch, halting.'
!               stop
            endif
            READ(40) SSIB%TCINI
            READ(40) SSIB%TGSINI
            READ(40) SSIB%TDINI
            READ(40) SSIB%TAINI
            READ(40) SSIB%TMINI
            READ(40) SSIB%HTINI
            READ(40) SSIB%QAINI
            DO L=1,3
               READ(40) TMPTILE
               SSIB%WWWINI(L) = TMPTILE
```

```
            ENDDO
            DO L=1,2
               READ(40) TMPTILE
               SSIB%CAPACINI(L) = TMPTILE
            ENDDO

            CLOSE(40)

            deallocate(tmptile)
            deallocate(TMP_COL)
            deallocate(TMP_ROW)
            deallocate(TMP_FGRD)
            deallocate(TMP_VEGT)
            deallocate(SIB_VEGT)
         endif
      endif
      return
```

1.81.11 ssib_com.F90 (Source File: ssib_com.F90)

Common block internal to the SSiB main program

REVISION HISTORY:

```
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   22 May 2004: David Mocko, Added some variables to common block
```

INTERFACE:

```
  module ssib_common
```

1.82 Fortran: Module Interface ssibdrv_module.f: (Source File: ssibdrv_module.F90)

Module for runtime specific SSiB variables

REVISION HISTORY:

```
   14 Oct 2003: Sujay Kumar, Initial Version
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   19 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
       MODULE ssibdrv_module
```

CONTENTS:

```
integer :: ssibopen          !Keeps track of opening files
integer :: numoutnh          !Counts number of output times for SSiB
integer :: ssib_nvegp        !Number of static vegetation parameter
integer :: ssib_nvegip       !Number of monthly vegetation parameter
integer :: ssib_flgres       !Restart flag
integer :: ssib_nsoilp       !Number of static soil parameters
integer :: ssib_zst          !Number of Zobler soil classes
integer :: ssib_gflag        !Time flag to update gfrac files
integer :: ssib_albtime      !Time flag to update albedo files
integer :: ssib_aflag        !Time flag to update albedo files
integer :: ssib_albdchk      !Day check to interpolate alb values
integer :: ssib_gfracdchk    !Day check to interpolate gfrac value
integer :: STATEVAR_AVG      !Instantaneous (=0) or Time-Averaged (=1) output for state v
CHARACTER*40 :: SSIB_RFILE   !SSIB Active Restart File
CHARACTER*40 :: SSIB_MFILE   !SSIB model init. restart file
CHARACTER*40 :: SSIB_VFILE   !SSIB Static Vegetation Parameter File
CHARACTER*40 :: SSIB_SFILE   !SSIB Soil Parameter File
CHARACTER*40 :: SSIB_MGFILE  !SSIB Monthly Veg. Green Frac.
CHARACTER*40 :: SSIB_ALBFILE !SSIB Quart. Snow-free albedo
CHARACTER*50 :: SSIB_MXSNAL  !SSIB GLDAS max snow albedo
CHARACTER*50 :: SSIB_TBOT    !SSIB GLDAS Bottom Temp
REAL*8 :: SSIB_GFRACTIME     !Time flag to update gfrac files
REAL :: SSIB_ISM             !SSIB Initial Soil Moisture (m3/m3)
REAL :: SSIB_IT              !SSIB Initial Soil Temperature (K)
REAL :: WRITEINTN            !SSIB Output Interval (hours)
end type ssibdrvdec
```

1.82.1 ssib_dynsetup.F90 (Source File: ssib_dynsetup.F90)

Updates the time dependent SSiB variables

REVISION HISTORY:

```
15 Apr 2002: Sujay Kumar, Initial Specification
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_dynsetup
```

USES:

```
use lisdrv_module, only: lis,tile
use ssib_varder
use spmdMod, only : masterproc, npes
use ssibpardef_module
```

CONTENTS:

```
#if ( ! defined OPENDAP )
      if (npes.gt.1) then
         call ssib_gather
      endif
      if (masterproc) then
#endif
         call ssib_gfrac
!         call ssib_alb(lis%d, lis%t, tile)
#if ( ! defined OPENDAP )
      endif
#if (defined SPMD)
      call MPI_BCAST(ssibdrv,1,MPI_SSIBDRV_STRUCT,0, &
          MPI_COMM_WORLD,ier)
#endif
      if ((npes.gt.1).and.((ssibdrv%ssib_gflag.eq.1).or. &
         (ssibdrv%ssib_aflag.eq.1))) then
         call ssib_scatter
      endif
#endif
      return
```

1.82.2 ssib_gather.F90 (Source File: ssib_gather.F90)

Gathers SSiB tiles

REVISION HISTORY:

```
      Apr 2003: Sujay Kumar, Initial Code
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
    6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
       subroutine ssib_gather
```

USES:

```
       use tile_spmdMod
       use ssib_varder
       use ssibpardef_module
```

CONTENTS:

```
#if  (defined SPMD)
      call MPI_GATHERV(ssib(1:di_array(iam)),di_array(iam), &
      MPI_SSIB_STRUCT,ssib,di_array,displs,MPI_SSIB_STRUCT, &
          0,MPI_COMM_WORLD, ierr)
```

```
#endif
      return
```

1.82.3 ssib_gfrac.F90 (Source File: ssib_gfrac.F90)

This subroutine takes vegetation greenness fraction data and the date to interpolate and determine the actual value of the greenness fraction for that date. This actual value is then returned to the main program. The assumption is that the data point is valid for the 16th of the given month, at 00Z.
Gustavo will take advantage of this routine to read the other time varying parameters and interpolate them to the appropriate time frame.

REVISION HISTORY:

```
  28 Apr 2002: K. Arsenault, Added SSiB LSM to LDAS, Initial Code
   1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
  18 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
       subroutine ssib_gfrac
```

USES:

```
       use ssib_varder     ! SSiB tile variables
       use time_manager
       use lisdrv_module, only : grid,tile,lis
 #if ( defined OPENDAP )
       use opendap_module
 #endif
```

CONTENTS:

```
!=== End Variable Definition ============================================

!-----------------------------------------------------------------------
! Declare vegetation parameter data
!-----------------------------------------------------------------------
      data zgreen/&
       0.9050000E+00,  0.2564000E-01,  0.8680600E+00,  0.9132400E+00,&
       0.2475200E+00,  0.6319100E+00,  0.5681800E+00,  0.7978700E+00,&
       0.8364300E+00,  0.4512600E+00,  0.1000000E-03,  0.2083300E+00,&
       0.1000000E-03,&
       0.9050000E+00,  0.2564000E-01,  0.8717700E+00,  0.9170300E+00,&
       0.2475200E+00,  0.6566600E+00,  0.6218900E+00,  0.5319100E+00,&
       0.7172100E+00,  0.4512600E+00,  0.1000000E-03,  0.2083300E+00,&
       0.1000000E-03,&
       0.9050000E+00,  0.4153800E+00,  0.8847300E+00,  0.9226600E+00,&
```

```
0.2475200E+00,  0.5176000E+00,  0.6637200E+00,  0.3623200E+00,&
0.2577300E+00,  0.4512600E+00,  0.1000000E-03,  0.4411800E+00,&
0.1000000E-03,&
0.9050000E+00,  0.7594900E+00,  0.9061000E+00,  0.9247000E+00,&
0.6637200E+00,  0.6527400E+00,  0.6972100E+00,  0.5681800E+00,&
0.7246400E+00,  0.4512600E+00,  0.1000000E-03,  0.7594900E+00,&
0.1000000E-03,&
0.9050000E+00,  0.8875700E+00,  0.9164200E+00,  0.9266400E+00,&
0.8104700E+00,  0.6527400E+00,  0.8104700E+00,  0.5681800E+00,&
0.1736100E+00,  0.4512600E+00,  0.1000000E-03,  0.8875700E+00,&
0.1000000E-03,&
0.9050000E+00,  0.9252000E+00,  0.9259300E+00,  0.9045800E+00,&
0.8680600E+00,  0.7246400E+00,  0.9079900E+00,  0.5681800E+00,&
0.5681800E+00,  0.6218900E+00,  0.1000000E-03,  0.9252000E+00,&
0.1000000E-03,&
0.9050000E+00,  0.8364300E+00,  0.9293700E+00,  0.9021600E+00,&
0.6040900E+00,  0.8712500E+00,  0.8132000E+00,  0.5681800E+00,&
0.5681800E+00,  0.9200800E+00,  0.1000000E-03,  0.8364300E+00,&
0.1000000E-03,&
0.9050000E+00,  0.6967200E+00,  0.8209400E+00,  0.9126500E+00,&
0.5854000E+00,  0.7966000E+00,  0.3943200E+00,  0.8680600E+00,&
0.7246400E+00,  0.6970300E+00,  0.1000000E-03,  0.6967200E+00,&
0.1000000E-03,&
0.9050000E+00,  0.3306900E+00,  0.7123000E+00,  0.8982800E+00,&
0.4990000E+00,  0.7654600E+00,  0.4434600E+00,  0.6505600E+00,&
0.8403400E+00,  0.7567000E-01,  0.1000000E-03,  0.3439200E+00,&
0.1000000E-03,&
0.9050000E+00,  0.1656400E+00,  0.6145700E+00,  0.8548200E+00,&
0.3834400E+00,  0.6146100E+00,  0.5434800E+00,  0.5154600E+00,&
0.8680600E+00,  0.4512600E+00,  0.1000000E-03,  0.1785700E+00,&
0.1000000E-03,&
0.9050000E+00,  0.1538000E-01,  0.8599500E+00,  0.8733600E+00,&
0.2487600E+00,  0.5086500E+00,  0.5531000E+00,  0.6302500E+00,&
0.8875700E+00,  0.4512600E+00,  0.1000000E-03,  0.1470600E+00,&
0.1000000E-03,&
0.9050000E+00,  0.2564000E-01,  0.8599500E+00,  0.9132400E+00,&
0.1984100E+00,  0.7898900E+00,  0.4975100E+00,  0.7978700E+00,&
0.9132400E+00,  0.4512600E+00,  0.1000000E-03,  0.2083300E+00,&
0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
```

```
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03/
data zxcover/&
  0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
  0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
  0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
  0.1000000E-04,&
  0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
  0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
  0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
  0.1000000E-04,&
```

```
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.9800000E+00,  0.7500000E+00,  0.7500000E+00,  0.7500000E+00,&
0.5000000E+00,  0.3000000E+00,  0.9000000E+00,  0.1000000E+00,&
0.1000000E+00,  0.3000000E+00,  0.1000000E-04,  0.7500000E-01,&
0.1000000E-04,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
0.1000000E-04,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
0.1000000E-04,&
```

```
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-04,  0.1000000E-03,&
  0.1000000E-04/
data zzlt/&
  0.5014160E+01,  0.3900000E+00,  0.3456000E+01,  0.6570000E+01,&
  0.4040000E+00,  0.1766000E+01,  0.7040000E+00,  0.5780000E+00,&
  0.1076000E+01,  0.3776000E+00,  0.1000000E-03,  0.4800000E-01,&
  0.1000000E-03,&
  0.5014160E+01,  0.3900000E+00,  0.3556000E+01,  0.6870000E+01,&
  0.4040000E+00,  0.1546000E+01,  0.8040000E+00,  0.5780000E+00,&
  0.9760000E+00,  0.3776000E+00,  0.1000000E-03,  0.4800000E-01,&
```

```
0.1000000E-03,&
0.5014160E+01,  0.6500000E+00,  0.3956000E+01,  0.7370000E+01,&
0.4040000E+00,  0.1416000E+01,  0.9040000E+00,  0.4480000E+00,&
0.7760000E+00,  0.3776000E+00,  0.1000000E-03,  0.6800000E-01,&
0.1000000E-03,&
0.5014160E+01,  0.1580000E+01,  0.4856000E+01,  0.7570000E+01,&
0.9040000E+00,  0.1216000E+01,  0.1004000E+01,  0.2880000E+00,&
0.2760000E+00,  0.3776000E+00,  0.1000000E-03,  0.1580000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.3380000E+01,  0.5456000E+01,  0.7770000E+01,&
0.1604000E+01,  0.1186000E+01,  0.1604000E+01,  0.2580000E+00,&
0.5760000E+00,  0.3776000E+00,  0.1000000E-03,  0.3380000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.5080000E+01,  0.6156000E+01,  0.8070000E+01,&
0.2304000E+01,  0.1416000E+01,  0.3304000E+01,  0.2580000E+00,&
0.1760000E+00,  0.5076000E+00,  0.1000000E-03,  0.5080000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.5380000E+01,  0.6456000E+01,  0.7870000E+01,&
0.4304000E+01,  0.2606000E+01,  0.4304000E+01,  0.2580000E+00,&
0.1760000E+00,  0.1737600E+01,  0.1000000E-03,  0.5380000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.4880000E+01,  0.6456000E+01,  0.7670000E+01,&
0.2904000E+01,  0.5206000E+01,  0.3804000E+01,  0.8080000E+00,&
0.2760000E+00,  0.1937600E+01,  0.1000000E-03,  0.4880000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.3780000E+01,  0.5756000E+01,  0.7570000E+01,&
0.2004000E+01,  0.4556000E+01,  0.1804000E+01,  0.1508000E+01,&
0.4760000E+00,  0.1477600E+01,  0.1000000E-03,  0.3780000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.1630000E+01,  0.4556000E+01,  0.7370000E+01,&
0.1304000E+01,  0.3816000E+01,  0.1104000E+01,  0.1148000E+01,&
0.5760000E+00,  0.3776000E+00,  0.1000000E-03,  0.1680000E+00,&
0.1000000E-03,&
0.5014160E+01,  0.6500000E+00,  0.3256000E+01,  0.6870000E+01,&
0.8040000E+00,  0.2806000E+01,  0.9040000E+00,  0.7480000E+00,&
0.6760000E+00,  0.3776000E+00,  0.1000000E-03,  0.6800000E-01,&
0.1000000E-03,&
0.5014160E+01,  0.3900000E+00,  0.3256000E+01,  0.6570000E+01,&
0.5040000E+00,  0.1866000E+01,  0.8040000E+00,  0.5780000E+00,&
0.8760000E+00,  0.3776000E+00,  0.1000000E-03,  0.4800000E-01,&
0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
```

```
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,  0.1000000E-03,&
  0.1000000E-03/
data x0x/&
  0.2652970E+01,  0.5201000E+00,  0.5706300E+00,  0.1112210E+01,&
  0.6414000E+00,  0.8427100E+00,  0.7771000E-01,  0.2446700E+00,&
  0.6559000E-01,  0.7524000E-01,  0.1118000E-01,  0.1448500E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.5201000E+00,  0.5696600E+00,  0.1102780E+01,&
  0.6414000E+00,  0.8087800E+00,  0.7779000E-01,  0.2446700E+00,&
```

```
  0.6549000E-01,  0.7524000E-01,  0.1118000E-01,  0.1448500E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.6664900E+00,  0.5656600E+00,  0.1087660E+01,&
  0.6414000E+00,  0.7875000E+00,  0.7785000E-01,  0.2272100E+00,&
  0.6521000E-01,  0.7524000E-01,  0.1118000E-01,  0.1752100E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.9105300E+00,  0.5654400E+00,  0.1081830E+01,&
  0.8633500E+00,  0.7284100E+00,  0.7788000E-01,  0.1998800E+00,&
  0.6360000E-01,  0.7524000E-01,  0.1118000E-01,  0.2871900E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.1031200E+01,  0.5592300E+00,  0.1076120E+01,&
  0.9728300E+00,  0.7284100E+00,  0.7779000E-01,  0.1998800E+00,&
  0.6480000E-01,  0.7524000E-01,  0.1118000E-01,  0.4302000E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.1043680E+01,  0.5524400E+00,  0.1067790E+01,&
  0.1005600E+01,  0.7875000E+00,  0.7712000E-01,  0.1998800E+00,&
  0.6331000E-01,  0.7575000E-01,  0.1118000E-01,  0.5087600E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.1041940E+01,  0.5497000E+00,  0.1073310E+01,&
  0.9967700E+00,  0.9266800E+00,  0.7594000E-01,  0.1998800E+00,&
  0.6331000E-01,  0.7767000E-01,  0.1118000E-01,  0.5200300E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.1037530E+01,  0.5497000E+00,  0.1078960E+01,&
  0.1011190E+01,  0.9715300E+00,  0.7658000E-01,  0.2674000E+00,&
  0.6360000E-01,  0.7782000E-01,  0.1118000E-01,  0.5009500E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.1036510E+01,  0.5562600E+00,  0.1081830E+01,&
  0.9965000E+00,  0.9658800E+00,  0.7776000E-01,  0.2923300E+00,&
  0.6446000E-01,  0.7745000E-01,  0.1118000E-01,  0.4503800E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.9170700E+00,  0.5686600E+00,  0.1087660E+01,&
  0.9386100E+00,  0.9555100E+00,  0.7790000E-01,  0.2803400E+00,&
  0.6480000E-01,  0.7524000E-01,  0.1118000E-01,  0.2973700E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.6664900E+00,  0.5725100E+00,  0.1102780E+01,&
  0.8346400E+00,  0.9204000E+00,  0.7785000E-01,  0.2580600E+00,&
  0.6510000E-01,  0.7524000E-01,  0.1118000E-01,  0.1752100E+00,&
  0.1011000E-01,&
  0.2652970E+01,  0.5201000E+00,  0.5725100E+00,  0.1112210E+01,&
  0.7049800E+00,  0.8427100E+00,  0.7779000E-01,  0.2446700E+00,&
  0.6537000E-01,  0.7524000E-01,  0.1118000E-01,  0.1448500E+00,&
  0.1011000E-01/
data xd/&
  0.2737261E+02,  0.1366377E+02,  0.1813464E+02,  0.1376361E+02,&
  0.9193320E+01,  0.1390777E+02,  0.2185200E+00,  0.2812600E+01,&
  0.1638000E+00,  0.1062900E+00,  0.6000000E-04,  0.6314240E+01,&
  0.4000000E-04,&
  0.2737261E+02,  0.1366377E+02,  0.1814677E+02,  0.1380041E+02,&
```

```
  0.9193320E+01,  0.1376090E+02,  0.2265800E+00,  0.2812600E+01,&
  0.1548100E+00,  0.1062900E+00,  0.6000000E-04,  0.6314240E+01,&
  0.4000000E-04,&
  0.2737261E+02,  0.1461883E+02,  0.1819051E+02,  0.1385740E+02,&
  0.9193320E+01,  0.1367074E+02,  0.2332800E+00,  0.2662290E+01,&
  0.1343400E+00,  0.1062900E+00,  0.6000000E-04,  0.7639520E+01,&
  0.4000000E-04,&
  0.2737261E+02,  0.1569677E+02,  0.1825890E+02,  0.1387880E+02,&
  0.9903400E+01,  0.1344527E+02,  0.2389500E+00,  0.2390910E+01,&
  0.6191000E-01,  0.1062900E+00,  0.6000000E-04,  0.1070958E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1632865E+02,  0.1829956E+02,  0.1389946E+02,&
  0.1030010E+02,  0.1344527E+02,  0.2605400E+00,  0.2390910E+01,&
  0.1096800E+00,  0.1062900E+00,  0.6000000E-04,  0.1278272E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1662263E+02,  0.1833903E+02,  0.1392915E+02,&
  0.1053455E+02,  0.1367074E+02,  0.2988000E+00,  0.2390910E+01,&
  0.5103000E-01,  0.1229900E+00,  0.6000000E-04,  0.1356813E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1666297E+02,  0.1835387E+02,  0.1390953E+02,&
  0.1091967E+02,  0.1425275E+02,  0.3251800E+00,  0.2390910E+01,&
  0.5103000E-01,  0.2152100E+00,  0.6000000E-04,  0.1366182E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1660123E+02,  0.1835387E+02,  0.1388922E+02,&
  0.1068047E+02,  0.1459719E+02,  0.3130700E+00,  0.2974600E+01,&
  0.6191000E-01,  0.2289700E+00,  0.6000000E-04,  0.1349985E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1641343E+02,  0.1831739E+02,  0.1387880E+02,&
  0.1044517E+02,  0.1452246E+02,  0.2649800E+00,  0.3137710E+01,&
  0.9547000E-01,  0.1996100E+00,  0.6000000E-04,  0.1301951E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1572679E+02,  0.1823553E+02,  0.1385740E+02,&
  0.1016423E+02,  0.1443002E+02,  0.2438100E+00,  0.3062460E+01,&
  0.1096800E+00,  0.1062900E+00,  0.6000000E-04,  0.1090759E+02,&
  0.4000000E-04,&
  0.2737261E+02,  0.1461883E+02,  0.1810866E+02,  0.1380041E+02,&
  0.9814290E+01,  0.1422050E+02,  0.2332800E+00,  0.2907360E+01,&
  0.1225000E+00,  0.1062900E+00,  0.6000000E-04,  0.7639520E+01,&
  0.4000000E-04,&
  0.2737261E+02,  0.1366377E+02,  0.1810866E+02,  0.1376361E+02,&
  0.9417390E+01,  0.1390777E+02,  0.2265800E+00,  0.2812600E+01,&
  0.1450200E+00,  0.1062900E+00,  0.6000000E-04,  0.6314240E+01,&
  0.4000000E-04/
data z1/&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
```

```
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03,&
  0.1000000E+01,  0.1150000E+02,  0.1600000E+02,  0.8500000E+01,&
  0.7000000E+01,  0.1000000E+02,  0.1000000E+00,  0.2000000E+01,&
  0.1000000E+00,  0.1000000E+00,  0.1000000E-01,  0.1150000E+02,&
  0.1000000E-03/
data z2/&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
```

```
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00,&
  0.3500000E+02,  0.2000000E+02,  0.2000000E+02,  0.1700000E+02,&
  0.1400000E+02,  0.1800000E+02,  0.6000000E+00,  0.5000000E+01,&
  0.5000000E+00,  0.6000000E+00,  0.1000000E+00,  0.2000000E+02,&
  0.1000000E+00/
data xdc/&
  0.2858700E+03,  0.2113200E+03,  0.2985200E+03,  0.5654100E+03,&
  0.1852000E+03,  0.2301300E+03,  0.2443000E+02,  0.1036000E+03,&
```

```
  0.2311000E+02,  0.2286000E+02,  0.2376000E+02,  0.1949000E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2113200E+03,  0.3013500E+03,  0.5870500E+03,&
  0.1852000E+03,  0.2244200E+03,  0.2463000E+02,  0.1036000E+03,&
  0.2294000E+02,  0.2286000E+02,  0.2376000E+02,  0.1949000E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2187800E+03,  0.3124600E+03,  0.6234600E+03,&
  0.1852000E+03,  0.2215700E+03,  0.2480000E+02,  0.1023500E+03,&
  0.2262000E+02,  0.2286000E+02,  0.2376000E+02,  0.1964400E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2434000E+03,  0.3312300E+03,  0.6381300E+03,&
  0.2048700E+03,  0.2164100E+03,  0.2496000E+02,  0.1007200E+03,&
  0.2189000E+02,  0.2286000E+02,  0.2376000E+02,  0.2014400E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2948700E+03,  0.3458300E+03,  0.6528600E+03,&
  0.2330100E+03,  0.2164100E+03,  0.2572000E+02,  0.1007200E+03,&
  0.2230000E+02,  0.2286000E+02,  0.2376000E+02,  0.2071300E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.3459000E+03,  0.3619400E+03,  0.6750500E+03,&
  0.2620800E+03,  0.2215700E+03,  0.2774000E+02,  0.1007200E+03,&
  0.2182000E+02,  0.2301000E+02,  0.2376000E+02,  0.2107900E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.3551800E+03,  0.3685400E+03,  0.6602400E+03,&
  0.3443100E+03,  0.2500700E+03,  0.3006000E+02,  0.1007200E+03,&
  0.2182000E+02,  0.2436000E+02,  0.2376000E+02,  0.2113100E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.3418400E+03,  0.3685400E+03,  0.6454900E+03,&
  0.2870900E+03,  0.2885700E+03,  0.2886000E+02,  0.1053000E+03,&
  0.2189000E+02,  0.2469000E+02,  0.2376000E+02,  0.2104200E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.3072200E+03,  0.3528500E+03,  0.6381300E+03,&
  0.2495800E+03,  0.2780300E+03,  0.2590000E+02,  0.1079400E+03,&
  0.2216000E+02,  0.2404000E+02,  0.2376000E+02,  0.2081500E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2448400E+03,  0.3236500E+03,  0.6231300E+03,&
  0.2211200E+03,  0.2668400E+03,  0.2511000E+02,  0.1065900E+03,&
  0.2230000E+02,  0.2286000E+02,  0.2376000E+02,  0.2018800E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2187800E+03,  0.2927900E+03,  0.5870500E+03,&
  0.2008900E+03,  0.2475700E+03,  0.2480000E+02,  0.1044900E+03,&
  0.2244000E+02,  0.2286000E+02,  0.2376000E+02,  0.1964400E+03,&
  0.2850000E+02,&
  0.2858700E+03,  0.2113200E+03,  0.2927900E+03,  0.5654100E+03,&
  0.1892600E+03,  0.2301300E+03,  0.2464000E+02,  0.1036000E+03,&
  0.2277000E+02,  0.2286000E+02,  0.2376000E+02,  0.1949000E+03,&
  0.2850000E+02/
data xbc/&
  0.5430000E+01,  0.6936000E+02,  0.8590000E+01,  0.8800000E+00,&
```

```
0.7850000E+01,  0.2661000E+02,  0.2207000E+02,  0.2188000E+02,&
0.1761000E+02,  0.4351000E+02,  0.3592951E+05,  0.5600000E+03,&
0.3546177E+05,&
0.5430000E+01,  0.6936000E+02,  0.8450000E+01,  0.8600000E+00,&
0.7850000E+01,  0.3044000E+02,  0.2053000E+02,  0.2188000E+02,&
0.1942000E+02,  0.4351000E+02,  0.3592951E+05,  0.5600000E+03,&
0.3546177E+05,&
0.5430000E+01,  0.4257000E+02,  0.7980000E+01,  0.8400000E+00,&
0.7850000E+01,  0.3295000E+02,  0.1934000E+02,  0.2673000E+02,&
0.2446000E+02,  0.4351000E+02,  0.3592951E+05,  0.4019700E+03,&
0.3546177E+05,&
0.5430000E+01,  0.1897000E+02,  0.7180000E+01,  0.8300000E+00,&
0.3810000E+01,  0.4003000E+02,  0.1838000E+02,  0.3712000E+02,&
0.6928000E+02,  0.4351000E+02,  0.3592951E+05,  0.1855200E+03,&
0.3546177E+05,&
0.5430000E+01,  0.1035000E+02,  0.6810000E+01,  0.8200000E+00,&
0.2400000E+01,  0.4003000E+02,  0.1516000E+02,  0.3712000E+02,&
0.3303000E+02,  0.4351000E+02,  0.3592951E+05,  0.9801000E+02,&
0.3546177E+05,&
0.5430000E+01,  0.7880000E+01,  0.6480000E+01,  0.8100000E+00,&
0.1860000E+01,  0.3295000E+02,  0.1068000E+02,  0.3712000E+02,&
0.8702000E+02,  0.3568000E+02,  0.3592951E+05,  0.7224000E+02,&
0.3546177E+05,&
0.5430000E+01,  0.7610000E+01,  0.6360000E+01,  0.8200000E+00,&
0.1290000E+01,  0.1870000E+02,  0.8300000E+01,  0.3712000E+02,&
0.8702000E+02,  0.1449000E+02,  0.3592951E+05,  0.6938000E+02,&
0.3546177E+05,&
0.5430000E+01,  0.8090000E+01,  0.6360000E+01,  0.8300000E+00,&
0.1600000E+01,  0.1318000E+02,  0.9330000E+01,  0.1722000E+02,&
0.6928000E+02,  0.1281000E+02,  0.3592951E+05,  0.7434000E+02,&
0.3546177E+05,&
0.5430000E+01,  0.9570000E+01,  0.6660000E+01,  0.8300000E+00,&
0.2040000E+01,  0.1420000E+02,  0.1457000E+02,  0.1317000E+02,&
0.4003000E+02,  0.1669000E+02,  0.3592951E+05,  0.8988000E+02,&
0.3546177E+05,&
0.5430000E+01,  0.1847000E+02,  0.7400000E+01,  0.8400000E+00,&
0.2820000E+01,  0.1559000E+02,  0.1760000E+02,  0.1497000E+02,&
0.3303000E+02,  0.4351000E+02,  0.3592951E+05,  0.1757600E+03,&
0.3546177E+05,&
0.5430000E+01,  0.4257000E+02,  0.8880000E+01,  0.8600000E+00,&
0.4210000E+01,  0.1933000E+02,  0.1934000E+02,  0.1906000E+02,&
0.2810000E+02,  0.4351000E+02,  0.3592951E+05,  0.4019700E+03,&
0.3546177E+05,&
0.5430000E+01,  0.6936000E+02,  0.8880000E+01,  0.8800000E+00,&
0.6400000E+01,  0.2661000E+02,  0.2053000E+02,  0.2188000E+02,&
0.2165000E+02,  0.4351000E+02,  0.3592951E+05,  0.5600000E+03,&
0.3546177E+05/
```

```
      zeroi=0
      numi=16
      ssibdrv%ssib_gflag = 0
!-----------------------------------------------------------------------
! Determine Monthly data Times (Assume Monthly value valid at DA=16)
!-----------------------------------------------------------------------
      if (lis%t%da.lt.16) then
         mo1=lis%t%mo-1
         yr1=lis%t%yr
         if (mo1.eq.0) then
            mo1=12
            yr1=lis%t%yr-1
         endif
         mo2=lis%t%mo
         yr2=lis%t%yr
      else
         mo1=lis%t%mo
         yr1=lis%t%yr
         mo2=lis%t%mo+1
         yr2=lis%t%yr
         if (mo2.eq.13) then
            mo2=1
            yr2=lis%t%yr+1
         endif
      endif

      call date2time(time1,doy1,gmt1,yr1,mo1,numi,zeroi,zeroi,zeroi)
      call date2time(time2,doy2,gmt2,yr2,mo2,numi,zeroi,zeroi,zeroi)

!-----------------------------------------------------------------------
!  Weights to be used to interpolate greenness fraction values.
!-----------------------------------------------------------------------
      wt1= (time2-lis%t%time)/(time2-time1)
      wt2= (lis%t%time-time1)/(time2-time1)
!-----------------------------------------------------------------------
!  Determine if monthly vegetation data needs to be updated
!-----------------------------------------------------------------------
      if (time2.gt.ssibdrv%ssib_gfractime) then
         ssibdrv%ssib_gfractime = time2
         ssibdrv%ssib_gflag = 1
      endif

!-----------------------------------------------------------------------
!  Interpolate monthly vegetation values once daily
!-----------------------------------------------------------------------
      if (ssibdrv%ssib_gfracdchk.ne.lis%t%da) then
         ssibdrv%ssib_gflag = 1
```

```
      do i=1,lis%d%nch
         SSIB(I)%VEGIP(1)  = (wt1 * ZGREEN(TILE(I)%VEGT,mo1,1)) + &
                             (wt2 * ZGREEN(TILE(I)%VEGT,mo2,1))
         SSIB(I)%VEGIP(2)  = (wt1 * ZGREEN(TILE(I)%VEGT,mo1,2)) + &
                             (wt2 * ZGREEN(TILE(I)%VEGT,mo2,2))
         SSIB(I)%VEGIP(3)  = (wt1 * ZXCOVER(TILE(I)%VEGT,mo1,1)) + &
                             (wt2 * ZXCOVER(TILE(I)%VEGT,mo2,1))
         SSIB(I)%VEGIP(4)  = (wt1 * ZXCOVER(TILE(I)%VEGT,mo1,2)) + &
                             (wt2 * ZXCOVER(TILE(I)%VEGT,mo2,2))
         SSIB(I)%VEGIP(5)  = (wt1 * ZZLT(TILE(I)%VEGT,mo1,1)) + &
                             (wt2 * ZZLT(TILE(I)%VEGT,mo2,1))
         SSIB(I)%VEGIP(6)  = (wt1 * ZZLT(TILE(I)%VEGT,mo1,2)) + &
                             (wt2 * ZZLT(TILE(I)%VEGT,mo2,2))
         SSIB(I)%VEGIP(7)  = (wt1 * XOX(TILE(I)%VEGT,mo1)) + &
                             (wt2 * XOX(TILE(I)%VEGT,mo2))
         SSIB(I)%VEGIP(8)  = (wt1 * XD(TILE(I)%VEGT,mo1)) + &
                             (wt2 * XD(TILE(I)%VEGT,mo2))
         SSIB(I)%VEGIP(9)  = (wt1 * Z1(TILE(I)%VEGT,mo1)) + &
                             (wt2 * Z1(TILE(I)%VEGT,mo2))
         SSIB(I)%VEGIP(10) = (wt1 * Z2(TILE(I)%VEGT,mo1)) + &
                             (wt2 * Z2(TILE(I)%VEGT,mo2))
         SSIB(I)%VEGIP(11) = (wt1 * XBC(TILE(I)%VEGT,mo1)) + &
                             (wt2 * XBC(TILE(I)%VEGT,mo2))
         SSIB(I)%VEGIP(12) = (wt1 * XDC(TILE(I)%VEGT,mo1)) + &
                             (wt2 * XDC(TILE(I)%VEGT,mo2))
      enddo
      ssibdrv%ssib_gfracdchk = lis%t%da

      if (lis%o%wparam.eq.1) then
         allocate(gfracout(lis%d%lnc,lis%d%lnr))
         do i=1,lis%d%nch
            if (grid(i)%lat*1000.ge.lis%d%gridDesc(4).and. &
                grid(i)%lat*1000.le.lis%d%gridDesc(7).and. &
                grid(i)%lon*1000.ge.lis%d%gridDesc(5).and. &
                grid(i)%lon*1000.le.lis%d%gridDesc(8)) then
               rindex = tile(i)%row - (lis%d%gridDesc(4)-lis%d%gridDesc(44)) &
                        / lis%d%gridDesc(9)
               cindex = tile(i)%col - (lis%d%gridDesc(5)-lis%d%gridDesc(45)) &
                        / lis%d%gridDesc(10)
               gfracout(cindex,rindex) = SSIB(I)%VEGIP(1)*1.0
            endif
         enddo
         open(32,file="gfracout.bin",form='unformatted')
         write(32) gfracout
         close(32)
         deallocate(gfracout)
      endif
   endif
```

```
      return
```

1.82.4 ssib_main.f (Source File: ssib_main.F90)

SSIB LAND-SURFACE MODEL, UNCOUPLED 1-D COLUMN: VERSION 2.5 OCT 2001

REVISION HISTORY:

```
   1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
  25 May 2004: David Mocko, removed duplicate code and provided ALMA outputs
  PROGRAM HISTORY LOG (prior to LIS)
  VERSION 1.0  --  01 MAR 1999
  VERSION 1.1  --  08 MAR 1999
  VERSION 2.0  --  27 JUL 1999
  VERSION 2.1  --  23 OCT 2000
  VERSION 2.2  --  07 FEB 2001
  VERSION 2.3  --  07 MAY 2001 = operational Eta implementation
  VERSION 2.4  --  27 JUN 2001 = ops Eta with NO physics changes
  VERSION 2.5  --  18 OCT 2001
   lis VERSION --  28 APR 2002 = SSIB Main added to lis Driver
                                 (NASA GSFC)
  VERSION 2.5.1--  28 MAY 2002 = Updated changes in SSIB LSM along
                                 with correction to SOILRZ and SOIL1M.
  VERSION 2.6  --  24 JUN 2003 = Updated to SSiB LSM v2.6
    Physics changes:
      in SUBROUTINE SFLX change CSOIL from 1.26E+6 to 2.00E+6
      in SUBROUTINE SFLX change ZBOT from -3.0 to -8.0
      Replaced de-bugged SUBROUTINE TDFCND
      Removed SUBROUTINE REDPRM and moved the parameters to other
       locations throughout ssib$_-$main and ssib$_-$physics subroutines.
     VERSION 2.5.2 --  31 MAY 2002
       Fix in FUNCTION DEVAP related to FX calculation
     VERSION 2.6   --  Includes changes to certain parameters and
                       snow-soil physics
```

INTERFACE:

```
      subroutine ssib_main
```

USES:

```
      use lisdrv_module, only : lis,grid,tile
      use ssib_varder      ! SSiB tile variables
      use tile_spmdMod
      use ssib_common
```

CONTENTS:

```
!=== Convert lis Timestep varname to SSIB timestep varname (DT) (sec)
      do t = 1, di_array(iam)
         DTT = float(lis%t%TS)

!=== RESET SOME LOCAL ACCUMULATIVE VARS
         RNOFFS = 0.
         SNM = 0.
         SIBSU = 0.
         BEDO = 0
         SOILDIF = 0
         SOILDRA = 0
         RNOFFT = 0
         RNOFFB = 0
         ITER = 1
!===END RESET SOME LOCAL ACCUMULATIVE VARS

         latco = grid(tile(t)%index)%lat
         lonco = grid(tile(t)%index)%lon
         prin = .false.
!         if ((latco.eq.66.5).and.(lonco.eq.-179.5)) prin = .true.

         CALL CONSTS

!==== At this point start to assign SSIB parameters to the respective variables

!***********************************************
!========static parameters
         TRAN(1,1,1) = SSIB(T)%VEGP(1)
         TRAN(2,1,1) = SSIB(T)%VEGP(2)
         TRAN(1,2,1) = SSIB(T)%VEGP(3)
         TRAN(2,2,1) = SSIB(T)%VEGP(4)
         TRAN(1,3,1) = SSIB(T)%VEGP(5)
         TRAN(2,3,1) = SSIB(T)%VEGP(6)
         TRAN(1,1,2) = SSIB(T)%VEGP(7)
         TRAN(2,1,2) = SSIB(T)%VEGP(8)
         TRAN(1,2,2) = SSIB(T)%VEGP(9)
         TRAN(2,2,2) = SSIB(T)%VEGP(10)
         TRAN(1,3,2) = SSIB(T)%VEGP(11)
         TRAN(2,3,2) = SSIB(T)%VEGP(12)
         REF(1,1,1) = SSIB(T)%VEGP(13)
         REF(2,1,1) = SSIB(T)%VEGP(14)
         REF(1,2,1) = SSIB(T)%VEGP(15)
         REF(2,2,1) = SSIB(T)%VEGP(16)
         REF(1,3,1) = SSIB(T)%VEGP(17)
         REF(2,3,1) = SSIB(T)%VEGP(18)
         REF(1,1,2) = SSIB(T)%VEGP(19)
         REF(2,1,2) = SSIB(T)%VEGP(20)
```

```
        REF(1,2,2) = SSIB(T)%VEGP(21)
        REF(2,2,2) = SSIB(T)%VEGP(22)
        REF(1,3,2) = SSIB(T)%VEGP(23)
        REF(2,3,2) = SSIB(T)%VEGP(24)
        RSTPAR(1,1) = SSIB(T)%VEGP(25)
        RSTPAR(1,2) = SSIB(T)%VEGP(26)
        RSTPAR(1,3) = SSIB(T)%VEGP(27)
        RSTPAR(2,1) = SSIB(T)%VEGP(28)
        RSTPAR(2,2) = SSIB(T)%VEGP(29)
        RSTPAR(2,3) = SSIB(T)%VEGP(30)
        SOREF(1) = SSIB(T)%VEGP(31)
        SOREF(2) = SSIB(T)%VEGP(32)
        SOREF(3) = SSIB(T)%VEGP(33)
        CHIL(1) = SSIB(T)%VEGP(34)
        CHIL(2) = SSIB(T)%VEGP(35)
        TOPT(1) = SSIB(T)%VEGP(36)
        TOPT(2) = SSIB(T)%VEGP(37)
        TLL(1) = SSIB(T)%VEGP(38)
        TLL(2) = SSIB(T)%VEGP(39)
        TU(1) = SSIB(T)%VEGP(40)
        TU(2) = SSIB(T)%VEGP(41)
        DEFAC(1) = SSIB(T)%VEGP(42)
        DEFAC(2) = SSIB(T)%VEGP(43)
        PH1(1) = SSIB(T)%VEGP(44)
        PH1(2) = SSIB(T)%VEGP(45)
        PH2(1) = SSIB(T)%VEGP(46)
        PH2(2) = SSIB(T)%VEGP(47)
        ROOTD(1) = SSIB(T)%VEGP(48)
        ROOTD(2) = SSIB(T)%VEGP(49)
        BEE = SSIB(T)%VEGP(50)
        PHSAT = SSIB(T)%VEGP(51)
        SATCO = SSIB(T)%VEGP(52)
        POROS = SSIB(T)%VEGP(53)
        SLOPE = SSIB(T)%VEGP(54)
        ZDEPTH(1) = SSIB(T)%VEGP(55)
        ZDEPTH(2) = SSIB(T)%VEGP(56)
        ZDEPTH(3) = SSIB(T)%VEGP(57)

!========monthly parameters
        GREEN(1)  = SSIB(T)%VEGIP(1)
        GREEN(2)  = SSIB(T)%VEGIP(2)
        VCOVER(1) = SSIB(T)%VEGIP(3)
        VCOVER(2) = SSIB(T)%VEGIP(4)
        ZLT(1)    = SSIB(T)%VEGIP(5)
        ZLT(2)    = SSIB(T)%VEGIP(6)
        Z0        = SSIB(T)%VEGIP(7)
        D         = SSIB(T)%VEGIP(8)
        Z1        = SSIB(T)%VEGIP(9)
```

```
         Z2        = SSIB(T)%VEGIP(10)
         RBC       = SSIB(T)%VEGIP(11)
         RDC       = SSIB(T)%VEGIP(12)

!======other parameters (state)
         TC       = SSIB(T)%TCINI
         TGS      = SSIB(T)%TGSINI
         TD       = SSIB(T)%TDINI
         TA       = SSIB(T)%TAINI
         TM       = SSIB(T)%TMINI
         HT       = SSIB(T)%HTINI
         QA       = SSIB(T)%QAINI
         WWW(1)   = SSIB(T)%WWWINI(1)
         WWW(2)   = SSIB(T)%WWWINI(2)
         WWW(3)   = SSIB(T)%WWWINI(3)
         CAPAC(1) = SSIB(T)%CAPACINI(1)
         CAPAC(2) = SSIB(T)%CAPACINI(2)
         ZWIND    = SSIB(T)%ZWINDINI
         ZMET     = SSIB(T)%ZMETINI
         ILW      = SSIB(T)%ILWINI
         ITRUNK   = SSIB(T)%ITRUNKINI

         VWIND  = (ssib(t)%FORCING(6))*(ssib(t)%FORCING(6))
         UWIND  = (ssib(t)%FORCING(5))*(ssib(t)%FORCING(5))
         SFCSPD = SQRT( UWIND + VWIND )
         swdown = ssib(t)%FORCING(3)
         rnetm =  ssib(t)%FORCING(4)
         tprec =  ssib(t)%FORCING(8)
         cprec =  ssib(t)%FORCING(9)
         tm =  ssib(t)%FORCING(1)
         um =  SFCSPD
         pr =  ssib(t)%FORCING(7)/100.
         QM =  ssib(t)%FORCING(2)
!         em = em * 98.59 *10. /0.622
         em = QM * pr / (0.622 + QM)

              ! specific humidity [kg/kg]  to vapor pressure [mb]
              !                    measured k83 pressure 98.59 Kpa

         um = amax1(um,0.25)
!         ustarm = mustar/100.
!         swdown = amax1(swdown,0.1)
         ppl = tprec - cprec
         ppc = cprec

         startsm = ((www(1) * zdepth(1)) + (www(2) * zdepth(2)) + &
                    (www(3) * zdepth(3))) * poros
         startint = capac(1)
```

```
         startswe = capac(2)

         if (prin) then
            print *,' '
            print *,'Latitude ',grid(tile(t)%index)%lat, &
                  '; Longitude ',grid(tile(t)%index)%lon
            print *,' '
            print *,'Constants: '
            print *,'tf: ',tf
            print *,' '
            print *,'Forcing data: '
            print *,'tm: ',tm
            print *,'pr: ',pr
            print *,'qm: ',qm
            print *,'um: ',um
            print *,'ppl: ',ppl
            print *,'ppc: ',ppc
            print *,'swdown: ',swdown
            print *,'rnetm: ',rnetm
            print *,' '
         endif

! ttii = WWW(3) + 100.
!      IF (ttii.eq.WWW(3)) THEN
!          WRITE (*,*) '--> Variable ',WWW(3),t,dtt,ttii
!   STOP
!       ENDIF !
!======end of assigning parameters
!      write(6,*) 'SOIL WETNESS FRACTION INITIALISATION: ',www(1),www(2),www(3)

         iyearh=lis%t%yr
         imonthh=lis%t%mo
         idayh=lis%t%da
         isech=(lis%t%hr*3600)+(lis%t%mn*60)
         if (prin) print *,'asdf',iyearh,imonthh,idayh,isech

         CALL ZENITHH(grid(tile(t)%index)%lat,grid(tile(t)%index)%lon, &
                      dtt,iyearh,imonthh,idayh,isech,angulosolar,prin)

!=== THE FOLLOWING BREAKS DOWN THE FORCING VARIABLES
!     THE FOLLOWING PROGRAM ONLY NEED TO BE CALLED ONCE FOR A LATITUDE
         IF (int(TILE(t)%VEGT).EQ.12) then
            CALL CROPS(grid(tile(t)%index)%lat,dum,lis%t%doy,NSX,VCOVER)
!,CHIL,ZLT,GREEN,RSTPAR,TOPT,TL,TU,DEFAC,PH2,PH1)
         endif

         RHOAIR = (pr*100.0) / GASR / TM
```

```
         SUNANG = angulosolar

         cloud = (1160.*sunang - swdown) / (963. * sunang)
         cloud = amax1(cloud,0.)
         cloud = amin1(cloud,1.)

         difrat = 0.0604 / ( sunang-0.0223 ) + 0.0683
         if ( difrat .lt. 0. ) difrat = 0.
         if ( difrat .gt. 1. ) difrat = 1.

         difrat = difrat + ( 1. - difrat ) * cloud
         vnrat = ( 580. - cloud*464. ) / ( ( 580. - cloud*499. ) &
               + ( 580. - cloud*464. ) )
         FRAC(1,1) = (1.-DIFRAT)*VNRAT
         FRAC(1,2) = DIFRAT*VNRAT
         FRAC(2,1) = (1.-DIFRAT)*(1.-VNRAT)
         FRAC(2,2) = DIFRAT*(1.-VNRAT)

         radn(1,1) = (1.-difrat)*vnrat*swdown
         radn(1,2) = difrat*vnrat*swdown
         radn(2,1) = (1.-difrat)*(1.-vnrat)*swdown
         radn(2,2) = difrat*(1.-vnrat)*swdown
         radn(3,2) = rnetm

         if (prin) then
            print *,'Radsplit: '
            print *,'radn_11: ',radn(1,1)
            print *,'radn_12: ',radn(1,2)
            print *,'radn_21: ',radn(2,1)
            print *,'radn_22: ',radn(2,2)
            print *,'sunang: ',sunang
         endif

         ppl = ppl * DTT
         ppc = ppc * DTT

!=== END OF BREAKING DOWN THE FORCING VARIABLES (SUBROUTINE DRIVER)
!=== ================================================================
!=== ================================================================
!=== ============BEGIN OF SIMULATION==============================

         CALL RADAB

         if (prin) then
            print *,'salb: ',salb(1,1),salb(1,2),salb(2,1),salb(2,2)
            print *,' '
            print *,'Initial conditions: '
            print *,'capac_1: ',capac(1)
```

```
      print *,'capac_2: ',capac(2)
      print *,'w_1: ',www(1)
      print *,'w_2: ',www(2)
      print *,'w_3: ',www(3)
      print *,'waterinlayer_1: ',(www(1) * poros * zdepth(1))
      print *,'waterinlayer_2: ',(www(2) * poros * zdepth(2))
      print *,'waterinlayer_3: ',(www(3) * poros * zdepth(3))
      print *,' '
      print *,'--------------------------------------------------'
      print *,' '
      print *,'w_1: ',www(1)
      print *,'w_2: ',www(2)
      print *,'w_3: ',www(3)
      print *,'phsat: ',phsat
      print *,'bee: ',bee
      print *,'CALLING ROOT'
   endif

   CALL ROOT1

   if (prin) then
      print *,'phsoil_1: ',phsoil(1)
      print *,'phsoil_2: ',phsoil(2)
      print *,'phsoil_3: ',phsoil(3)

      print *,' '
      print *,'--------------------------------------------------'
      print *,' '
      print *,'stefan: ',stefan
      print *,'vcover_1: ',vcover(1)
      print *,'vcover_2: ',vcover(2)
      print *,'thermk: ',thermk
      print *,'tc: ',tc
      print *,'tgs: ',tgs
      print *,'radn_11: ',radn(1,1)
      print *,'radn_12: ',radn(1,2)
      print *,'radn_21: ',radn(2,1)
      print *,'radn_22: ',radn(2,2)
      print *,'CALLED RADUSE'
      print *,'radt_1: ',radt(1)
      print *,'radt_2: ',radt(2)
      print *,'par: ',par(1)
      print *,'pd: ',pd(1)

      print *,' '
      print *,'--------------------------------------------------'
      print *,' '
      print *,'ityp: ',TILE(t)%VEGT
```

```
            print *,'zlt_1: ',zlt(1)
            print *,'zlt_2: ',zlt(2)
            print *,'green: ',green(1)
            print *,'vcover_1: ',vcover(1)
            print *,'vcover_2: ',vcover(2)
            print *,'chil: ',chil(1)
            print *,'rstpar_1: ',rstpar(1,1)
            print *,'rstpar_2: ',rstpar(1,2)
            print *,'rstpar_3: ',rstpar(1,3)
            print *,'sunang: ',sunang
            print *,'par: ',par(1)
            print *,'pd: ',pd(1)
            print *,'CALLING STOMAT'
         endif

         CALL STOMAT1
         RSTUN = RST(1)

! ** WATER BALANCE CHECK
         TOTWB = WWW(1) * POROS * ZDEPTH(1) &
             + WWW(2) * POROS * ZDEPTH(2)  &
             + WWW(3) * POROS * ZDEPTH(3)  &
             + CAPAC(1) + CAPAC(2)

         if (prin) then
            print *,'rst_1: ',rst(1)
            print *,'rst_2: ',rst(2)

            print *,' '
            print *,'--------------------------------------------------'
            print *,' '
            print *,'dt: ',dtt
            print *,'snomel: ',snomel
            print *,'clai: ',clai
            print *,'cw: ',cw
            print *,'tf: ',tf
            print *,'poros: ',poros
            print *,'satco: ',satco
            print *,'zdepth_1: ',zdepth(1)
            print *,'zdepth_2: ',zdepth(2)
            print *,'zdepth_3: ',zdepth(3)
            print *,'capac_1: ',capac(1)
            print *,'capac_2: ',capac(2)
            print *,'w_1: ',www(1)
            print *,'w_2: ',www(2)
            print *,'w_3: ',www(3)
            print *,'satcap_1: ',satcap(1)
            print *,'satcap_2: ',satcap(2)
```

```
      print *,'tm: ',tm
      print *,'ppl: ',ppl
      print *,'ppc: ',ppc
      print *,'tc: ',tc
      print *,'tgs: ',tgs
      print *,'cg: ',cg
      print *,'CALLING INTERC'
   endif

   CALL INTERC

   if (prin) then
      print *,'tc: ',tc
      print *,'tgs: ',tgs
      print *,'capac_1: ',capac(1)
      print *,'capac_2: ',capac(2)
      print *,'cc: ',ccx
      print *,'cg: ',cg
      print *,'roff: ',roff
   endif

   currwb = (((www(1) * zdepth(1)) + (www(2) * zdepth(2)) + &
              (www(3) * zdepth(3))) * poros) + capac(1)   + &
              capac(2) + roff - ((ppl + ppc) / 1000.0)
   if ((abs(currwb-totwb).gt.0.000001).or.(prin)) then
      print *,' '
      print *,'INTERC CAPAC DIFF: ',(currwb-totwb)
      print *,'point: ',t,'  ',lonco,'  ',latco
      print *,'currtotal: ',currwb
      print *,'starttotal: ',totwb
      print *,'capac_1: ',capac(1)
      print *,'capac_2: ',capac(2)
      print *,'w_1: ',www(1)
      print *,'w_2: ',www(2)
      print *,'w_3: ',www(3)
      print *,'waterinlayer_1: ',(www(1) * poros * zdepth(1))
      print *,'waterinlayer_2: ',(www(2) * poros * zdepth(2))
      print *,'waterinlayer_3: ',(www(3) * poros * zdepth(3))
      print *,'ppl: ',(ppl/1000.)
      print *,'ppc: ',(ppc/1000.)
      print *,'roff: ',roff
      print *,'tm: ',tm
      print *,'tc: ',tc
      print *,'tgs: ',tgs
   endif
   if (abs(currwb-totwb).gt.0.000001) then
      print *,'interc capac error!'
      stop
```

```
      endif

      if (prin) then
         print *,'tc: ',tc
         print *,'tgs: ',tgs
         print *,'tm: ',tm
         print *,'qm: ',qm
         print *,'CALLING TEMRES'
      endif

      currwb = (((www(1) * zdepth(1)) + (www(2) * zdepth(2)) + &
                 (www(3) * zdepth(3))) * poros) + capac(1)   + &
                 capac(2) + roff - ((ppl + ppc) / 1000.0)
      if ((abs(currwb-totwb).gt.0.000001).or.(prin)) then
         print *,' '
         print *,'TEMRES CAPAC DIFF: ',(currwb-totwb)
         print *,'point: ',t,'  ',lonco,'  ',latco
         print *,'currtotal: ',currwb
         print *,'starttotal: ',totwb
         print *,'capac_1: ',capac(1)
         print *,'capac_2: ',capac(2)
         print *,'w_1: ',www(1)
         print *,'w_2: ',www(2)
         print *,'w_3: ',www(3)
         print *,'waterinlayer_1: ',(www(1) * poros * zdepth(1))
         print *,'waterinlayer_2: ',(www(2) * poros * zdepth(2))
         print *,'waterinlayer_3: ',(www(3) * poros * zdepth(3))
         print *,'ppl: ',(ppl/1000.)
         print *,'ppc: ',(ppc/1000.)
         print *,'roff: ',roff
         print *,'tm: ',tm
         print *,'tc: ',tc
         print *,'tgs: ',tgs
      endif
      if (abs(currwb-totwb).gt.0.000001) then
         print *,'temres capac error!'
         stop
      endif

   CALL TEMRS1(prin)

      currwb = (((www(1) * zdepth(1)) + (www(2) * zdepth(2)) + &
                 (www(3) * zdepth(3))) * poros) + capac(1)   + &
                 capac(2) + roff - ((ppl + ppc) / 1000.0)
      if ((abs(currwb-totwb).gt.0.000001).or.(prin)) then
         print *,' '
         print *,'TEMRES CAPAC DIFF: ',(currwb-totwb)
         print *,'point: ',t,'  ',lonco,'  ',latco
```

```
      print *,'currtotal: ',currwb
      print *,'starttotal: ',totwb
      print *,'capac_1: ',capac(1)
      print *,'capac_2: ',capac(2)
      print *,'w_1: ',www(1)
      print *,'w_2: ',www(2)
      print *,'w_3: ',www(3)
      print *,'waterinlayer_1: ',(www(1) * poros * zdepth(1))
      print *,'waterinlayer_2: ',(www(2) * poros * zdepth(2))
      print *,'waterinlayer_3: ',(www(3) * poros * zdepth(3))
      print *,'ppl: ',(ppl/1000.)
      print *,'ppc: ',(ppc/1000.)
      print *,'roff: ',roff
      print *,'tm: ',tm
      print *,'tc: ',tc
      print *,'tgs: ',tgs
   endif
   if (abs(currwb-totwb).gt.0.000001) then
      print *,'temres capac error!'
      stop
   endif

   if (prin) then
      print *,' '
      print *,'--------------------------------------------------'
      print *,' '
      print *,'tc: ',tc
      print *,'tgs: ',tgs
      print *,'tm: ',tm
      print *,'qm: ',qm
      print *,'CALLING UPDATE'
   endif

   CALL UPDAT1(prin)

   ENDWB = WWW(1) * POROS * ZDEPTH(1)  &
         + WWW(2) * POROS * ZDEPTH(2)   &
         + WWW(3) * POROS * ZDEPTH(3)   &
         + CAPAC(1) + CAPAC(2) - (PPL+PPC)/1000. + ETMASS/1000. + ROFF
   ERROR = TOTWB - ENDWB

   IF (ABS(ERROR).GT.0.0001) THEN
      WRITE(24,882) ERROR
      print *,' '
      print *,'Water Balance ERROR!'
      print *,'Error: ',ERROR
      print *,'latco: ',latco
      print *,'lonco: ',lonco
```

```
            print *,'www_1: ',www(1)
            print *,'www_2: ',www(2)
            print *,'www_3: ',www(3)
            print *,'waterinlayer_1: ',(www(1) * poros * zdepth(1))
            print *,'waterinlayer_2: ',(www(2) * poros * zdepth(2))
            print *,'waterinlayer_3: ',(www(3) * poros * zdepth(3))
            print *,'capac_1: ',capac(1)
            print *,'capac_2: ',capac(2)
            print *,'ppl: ',(ppl/1000.)
            print *,'ppc: ',(ppc/1000.)
            print *,'etmass: ',(etmass/1000.)
            print *,'roff: ',roff
            stop
         endif
 882     FORMAT(1X,'WARNING WATER BALANCE AFTER UPDATE ',f8.6)

         CBAL = RADT(1) - CHF - (ECT+HC+ECI)/DTT
         GBAL = RADT(2) - SHF - (EGT+EGI+HG+EGS)/DTT
         ZLHS = RADT(1) + RADT(2) - CHF - SHF
         ZRHS = HFLUX + (ECT + ECI + EGT + EGI + EGS)/DTT

         IF (ABS(ZLHS-ZRHS).GT.1.) WRITE(24,881) ABS(ZLHS-ZRHS)
 881     FORMAT(1X,' WARNING ENERGY BALANCE ',f8.6)

! latent heat flux
         ectw = ect / dtt
         eciw = eci / dtt
         egtw = egt / dtt
         egiw = egi / dtt
         egsw = egs / dtt
         if (prin) then
            print *,'ect: ',ect
            print *,'eci: ',eci
            print *,'egt: ',egt
            print *,'egi: ',egi
            print *,'egs: ',egs
            print *,'dtt: ',dtt
         endif

! soil moisture
         w1i = WWW(1) * POROS * ZDEPTH(1) * 1000.
         w2i = WWW(2) * POROS * ZDEPTH(2) * 1000.
         w3i = WWW(3) * POROS * ZDEPTH(3) * 1000.
         if (swdown.gt.0.0) then
            suralbedo = ((radn(1,1) * salb(1,1)) + (radn(1,2) * salb(1,2)) + &
                         (radn(2,1) * salb(2,1)) + (radn(2,2) * salb(2,2))) / swdown
         else
            suralbedo = LIS%d%UDEF
```

```
         endif
         soilwet = (zdepth(1) + zdepth(2) + zdepth(3)) * &
                   ((-exp(ph2(1)) / phsat) ** (-bee))
         soilwet = (((w1i + w2i + w3i) / poros / 1000.) - soilwet) / &
                   ((zdepth(1) + zdepth(2) + zdepth(3)) - soilwet)

!=== ===============END OF SIMULATION==============================
!===================================================================
!===================================================================

!      PRINT*,'  ------------------------------------'
!      PRINT*,'  State Variables '
!      PRINT*,'  ------------------------------------'
!      WRITE(*,*) SSIB(T)%T1,' T1...Skin temperature (K)'
!      WRITE(*,*)(SSIB(T)%STC(IJ), IJ=1,NSOIL),' STC'
!      WRITE(*,*)(SSIB(T)%SMC(IJ), IJ=1,NSOIL),' SMC'
!      WRITE(*,*)(SSIB(T)%SH2O(IJ), IJ=1,NSOIL),' SH2O'
!      WRITE(*,*) SSIB(T)%CMC,' CMC...Canopy water content (m)'
!      WRITE(*,*) SSIB(T)%SNOWH,' SNOWH...Actual snow depth (m)'
!      WRITE(*,*) SSIB(T)%SNEQV,' SNEQV...Water equiv snow depth (m)'
!      WRITE(*,*) 'CH= ',SSIB(T)%CH,'   CM= ',SSIB(T)%CM
!      PRINT*,'  ------------------------------------'

!=== Collect the output variables into SSIB(T)%RETURN
!      IF (ABS(ERROR).GT.0.0001) THEN
!         write(24,*) ERROR, (WWW(1) * POROS * ZDEPTH(1)),  &
!                            (WWW(2) * POROS * ZDEPTH(2)),  &
!                            (WWW(3) * POROS * ZDEPTH(3)),  &
!         CAPAC(1) , CAPAC(2) , PPL/1000. , ETMASS/1000. , ROFF
!      ENDIF

!=== Collect state variables

      SSIB(T)%TCINI    = TC       ! CANOPY TEMPERATURE (K)
      SSIB(T)%TGSINI    = TGS      ! SOIL SURFACE TEMPERATURE (K)
      SSIB(T)%TDINI    = TD       ! DEEP SOIL TEMPERATURE (K)
      SSIB(T)%TAINI    = TA       ! TEMPERATURE AT CANOPY AIR SPACE (K)
      SSIB(T)%TMINI    = TM       ! TEMPERATURE AT LOWEST MODEL LAYER (K)
      SSIB(T)%HTINI    = HT
      SSIB(T)%QAINI    = QA
      SSIB(T)%WWWINI(1)    = WWW(1)   !SOIL MOISTURE
      SSIB(T)%WWWINI(2)    = WWW(2)
      SSIB(T)%WWWINI(3)    = WWW(3)
      SSIB(T)%CAPACINI(1)  = CAPAC(1) !INTERCEPTION AT CANOPY
      SSIB(T)%CAPACINI(2)  = CAPAC(2) !SNOW DEPTH

!=== Collect the ALMA output variables
      SSIB(T)%swnet        = SSIB(T)%swnet    + swcan + swgnd
```

```
      SSIB(T)%lwnet        = SSIB(T)%lwnet    + radn(3,2) - zlwup
      SSIB(T)%qle          = SSIB(T)%qle      + (ectw+eciw+egtw+egiw+egsw)
      SSIB(T)%qh           = SSIB(T)%qh       + hflux
      SSIB(T)%qg           = SSIB(T)%qg       + shf
      SSIB(T)%qf           = SSIB(T)%qf       + (smelt * snomel / dtt)
      SSIB(T)%qtau         = SSIB(T)%qtau     + (drag * um)
      SSIB(T)%delsurfheat  = SSIB(T)%delsurfheat + (chf*dtt)
      if (ssib(t)%FORCING(1).lt.tf) then
         SSIB(T)%snowf        = SSIB(T)%snowf + tprec
      else
         SSIB(T)%rainf        = SSIB(T)%rainf + tprec
      endif
      SSIB(T)%evap         = SSIB(T)%evap     + (etmass/dtt)
      SSIB(T)%qs           = SSIB(T)%qs       + max((roff - (q3g * dtt)),0.0) * 1000. / dtt
      SSIB(T)%qsb          = SSIB(T)%qsb      + (q3g * dtt) * 1000. / dtt
      SSIB(T)%qsm          = SSIB(T)%qsm      + smelt * 1000. / dtt
      SSIB(T)%delsoilmoist = SSIB(T)%delsoilmoist + ((((www(1) * zdepth(1)) + &
                                                      (www(2) * zdepth(2)) + &
                                                      (www(3) * zdepth(3))) * &
                                                       poros) - startsm) * 1000.
      SSIB(T)%delswe       = SSIB(T)%delswe       + (capac(2) - startswe) * 1000.
      SSIB(T)%delintercept = SSIB(T)%delintercept + (capac(1) - startint) * 1000.
! These variables are either instantanous or time-averaged,
! depending on value of STATEVAR_AVG flag in lis.crd namelist.
      if (ssibdrv%STATEVAR_AVG.eq.0) then
         SSIB(T)%vegtc           = tc
         SSIB(T)%baresoilt       = tgs
         SSIB(T)%avgsurft        = (tgs * (1.0 - vcover(1))) + (tc * vcover(1))
         SSIB(T)%radteff         = tgeff
         if (swdown.gt.0.0) then
            SSIB(T)%albedo       = suralbedo
         else
            SSIB(T)%albedo       = LIS%d%UDEF
         endif
         SSIB(T)%swe             = capac(2) * 1000.
         if (tc.lt.tf) then
            SSIB(T)%sweveg       = capac(1) * 1000.
         else
            SSIB(T)%sweveg       = 0.0
         endif
         SSIB(T)%soilmoist1      = w1i
         SSIB(T)%soilmoist2      = w2i
         SSIB(T)%soilmoist3      = w3i
         SSIB(T)%soiltemp        = td
         SSIB(T)%soilwet         = soilwet
      else
         SSIB(T)%vegtc           = SSIB(T)%vegtc       + tc
         SSIB(T)%baresoilt       = SSIB(T)%baresoilt   + tgs
```

```
         SSIB(T)%avgsurft        = SSIB(T)%avgsurft    + (tgs * (1.0 - vcover(1))) + (tc * vco
         SSIB(T)%radteff         = SSIB(T)%radteff     + tgeff
         if (swdown.gt.0.0) then
            SSIB(T)%albedo       = SSIB(T)%albedo      + suralbedo
            SSIB(T)%albedocount  = SSIB(T)%albedocount + 1
         endif
         SSIB(T)%swe             = SSIB(T)%swe         + (capac(2) * 1000.)
         if (tc.lt.tf) then
            SSIB(T)%sweveg       = SSIB(T)%sweveg      + (capac(1) * 1000.)
         else
            SSIB(T)%sweveg       = SSIB(T)%sweveg      + 0.0
         endif
         SSIB(T)%soilmoist1      = SSIB(T)%soilmoist1  + w1i
         SSIB(T)%soilmoist2      = SSIB(T)%soilmoist2  + w2i
         SSIB(T)%soilmoist3      = SSIB(T)%soilmoist3  + w3i
         SSIB(T)%soiltemp        = SSIB(T)%soiltemp    + td
         SSIB(T)%soilwet         = SSIB(T)%soilwet     + soilwet
      endif
! Back to typical output variables
      SSIB(T)%ecanop       = SSIB(T)%ecanop    + ((eciw + egiw) / hlat)
      SSIB(T)%tveg         = SSIB(T)%tveg      + ((ectw + egtw) / hlat)
      SSIB(T)%esoil        = SSIB(T)%esoil     + (egsw / hlat)
      SSIB(T)%rootmoist    = SSIB(T)%rootmoist + (w1i + w2i)
      SSIB(T)%canopint     = SSIB(T)%canopint  + (capac(1) * 1000.)
      SSIB(T)%acond        = SSIB(T)%acond     + (1.0 / ra)
      SSIB(T)%snowfrac     = SSIB(T)%snowfrac  + scov2

      enddo                     ! end di_array

      ssib%count = ssib%count + 1

!>>>  END OF SSIB_MAIN <<<<<<<<
!      stop

      return
```

1.82.5 ssib_mapvegc.F90 (Source File: ssib_mapvegc.F90)

This subroutine converts the UMD classes to the SIB classes used by SSIB LSM (v 2.5). (Originally from Dag Lohmann at NCEP)

REVISION HISTORY:

```
  28 Apr 2002: K Arsenault, Added SSIB LSM to LDAS
   1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   5 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_mapvegc(VEGT)
```

CONTENTS:

```
!  Convert UMD Classes to SSiB Classes.
      IF (VEGT.EQ.1)  SIBVEG = 4
      IF (VEGT.EQ.2)  SIBVEG = 1
      IF (VEGT.EQ.3)  SIBVEG = 5
      IF (VEGT.EQ.4)  SIBVEG = 2
      IF (VEGT.EQ.5)  SIBVEG = 3
      IF (VEGT.EQ.6)  SIBVEG = 3
      IF (VEGT.EQ.7)  SIBVEG = 6
      IF (VEGT.EQ.8)  SIBVEG = 8
      IF (VEGT.EQ.9)  SIBVEG = 9
      IF (VEGT.EQ.10) SIBVEG = 7
      IF (VEGT.EQ.11) SIBVEG = 12
      IF (VEGT.EQ.12) SIBVEG = 11
      IF (VEGT.EQ.13) SIBVEG = 11
      IF (VEGT.GT.13) SIBVEG = 7

      VEGT = SIBVEG
      return
```

1.83 Fortran: Module Interface ssib_module.f: (Source File: ssib_module.F90)

Module for 1-D SSiB land model driver variable specification

REVISION HISTORY:

```
28 Apr 2002: Kristi Arsenault, added SSiB LSM 2.5 code to LDAS
14 Nov 2002: Sujay Kumar, Optimized version for LIS
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
MODULE ssib_module
```

CONTENTS:

```
      INTEGER :: ts                  !Timestep (seconds)
      INTEGER :: maxt                !Maximum tiles per grid
      INTEGER :: SIBVEG              !UMD to SiB Vegetation Class Index value
      INTEGER :: NSLAY               !Number of SSiB soil layers (4)
      INTEGER :: COUNT
      INTEGER :: ALBEDOCOUNT
```

```
      INTEGER :: ZOBSOIL(1)  !Zobler Soil Classes (LIS%NCH)
      INTEGER :: ITRUNKINI
      INTEGER :: ILWINI

      REAL :: VEGP(57)     !Static vegetation parameters,   dim(SSIB_NVEGP)
      REAL :: VEGIP(12)    !Interpolated monthly parameters,dim(SSIB_NVEGIP)
      REAL :: VEGT         !vegetation type of tile
      REAL :: XDTT

!=== SSiB-Forcing Variables ============================================
      REAL :: FORCING(10)        ! TILE FORCING..
!=== SSiB-Initial Variables ============================================
      REAL :: ZWINDINI
      REAL :: ZMETINI
!=== SSiB-State Variables ==============================================
      REAL :: TCINI
      REAL :: TGSINI
      REAL :: TDINI
      REAL :: TAINI
      REAL :: TMINI
      REAL :: HTINI
      REAL :: QAINI
      REAL :: WWWINI(3)
      REAL :: CAPACINI(2)
!==== SSiB-Output ======================================================
      REAL :: swnet
      REAL :: lwnet
      REAL :: qle
      REAL :: qh
      REAL :: qg
      REAL :: qf
      REAL :: qtau
      REAL :: delsurfheat
      REAL :: snowf
      REAL :: rainf
      REAL :: evap
      REAL :: qs
      REAL :: qsb
      REAL :: qsm
      REAL :: delsoilmoist
      REAL :: delswe
      REAL :: delintercept
      REAL :: vegtc
      REAL :: baresoilt
      REAL :: avgsurft
      REAL :: radteff
      REAL :: albedo
      REAL :: swe
```

```
      REAL :: sweveg
      REAL :: soilmoist1
      REAL :: soilmoist2
      REAL :: soilmoist3
      REAL :: soiltemp
      REAL :: soilwet
      REAL :: ecanop
      REAL :: tveg
      REAL :: esoil
      REAL :: rootmoist
      REAL :: canopint
      REAL :: acond
      REAL :: snowfrac

      end type ssibdec
!=== End Variable List ================================================
```

1.83.1 ssib_output.F90 (Source File: ssib_output.F90)

This subroutine sets up methods to write SSiB output

REVISION HISTORY:

```
1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_output
```

USES:

```
use lisdrv_module, only : lis, tile, glbgindex
use ssib_varder, only : ssibdrv
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
      if (lis%o%wsingle.eq.1) then
!-----------------------------------------------------------------
! Writes each output variables to a separate file
!-----------------------------------------------------------------
         if (mod(lis%t%gmt,ssibdrv%writeintn).eq.0) then
            do i = 1,33
               call ssib_singlegather(i,var)
               if (masterproc) then
                  call ssib_singleout(lis, tile, glbgindex, var, i)
               endif
```

```
            enddo
            call ssib_totinit
         endif
      else
!-----------------------------------------------------------------
! Writes bundled output
!-----------------------------------------------------------------
         if (mod(lis%t%gmt,ssibdrv%writeintn).eq.0) then
            if (npes.gt.1) then
               call ssib_gather
            endif
            if (masterproc) then
               call ssib_almaout()
            endif
            call ssib_totinit
         endif
      endif
      return
```

1.84 Fortran: Module Interface ssibpardef_module.F90 (Source File: ssibpardef_module.F90)

This module contains routines that defines MPI derived data types for SSiB LSM

REVISION HISTORY:

```
   06 Oct 2003: Sujay Kumar, Initial Specification
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   22 May 2004: David Mocko, made compatible with SiB-lings
 #include <misc.h>
```

INTERFACE:

```
       module ssibpardef_module
```

USES:

```
       use ssib_module
       use ssibdrv_module
       use spmdMod
```

Routine that defines MPI derived data types for SSiB

INTERFACE:

```
       subroutine def_ssibpar_struct
```

1.84.1 ssibrst.F90 (Source File: ssibrst.F90)

This program reads restart files for SSiB. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
 1 Oct 1999: Jared Entin, Initial Code
15 Oct 1999: Paul Houser, Significant F90 Revision
05 Sep 2001: Brian Cosgrove, Modified code to use Dag Lohmann's NOAA
             initial conditions if necessary.  This is controlled with
             local variable NOAAIC.  Normally set to 0 in this subroutine
             but set to 1 if want to use Dag's NOAA IC's.  Changed output
             directory structure, and commented out if-then check so that
             directory is always made.
28 Apr 2002: Kristi Arsenault, Added SSIB LSM into LDAS
28 May 2002: Kristi Arsenault, For STARTCODE=4, corrected SNEQV values
             and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
30 Oct 2003: Matt Rodell, Added back COL,ROW,FGRD,VEGT to restart files
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 5 May 2004: David Mocko, made compatible with SiB-lings
RESTART FILE FORMAT(fortran sequential binary):
 YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
 TILE(NCH)%COL        !Grid Col of Tile
 TILE(NCH)%ROW        !Grid Row of Tile
 TILE(NCH)%FGRD       !Fraction of Grid covered by Tile
 TILE(NCH)%VEGT       !Vegetation Type of Tile
 SSIB(NCH)%STATES     !Model States in Tile Space
```

INTERFACE:

```
      SUBROUTINE SSIBRST
```

USES:

```
      use lisdrv_module, only : lis, grid, tile
      use ssib_varder, only : ssibdrv
      use ssib_varder      ! SSiB tile variables
      use time_manager
      use tile_spmdMod
```

CONTENTS:

```
!=== End Variable Definition ==============================================

!=== Read Active Archive File =============================================
      print*,'DBG: ssibrst -- in ssibrst',' (',iam,')'
!<kluge don't do ssibrst -- see ssib_coldstart>
!</kluge don't do ssibrst -- see ssib_coldstart>
!      write(*,*) masterproc
```

```
      if (masterproc) then
         if (LIS%O%STARTCODE.EQ.1) then
            allocate(tmptile(lis%d%nch))
            OPEN(40,FILE=ssibdrv%SSIB_RFILE,FORM='unformatted')

            call timemgr_read_restart(40)
!            call timemgr_restart()
!            call get_curr_date(lis%t%yr,lis%t%mo,lis%t%da,curSec)
!            call sec2time(curSec,lis%t%hr,lis%t%mn,lis%t%ss)
!            call updatetime(lis%t) !Updates LIS variables.

            WRITE(*,*)'SSIB Restart File Used: ',ssibdrv%SSIB_RFILE

            READ(40) VCLASS,lnc,lnr,NCH !Time, veg class, no. tiles

!   Check for Vegetation Class Conflict
            IF (VCLASS.NE.LIS%P%VCLASS) THEN
               WRITE(*,*) ssibdrv%SSIB_RFILE, &
                          'Vegetation class conflict'
               STOP
            ENDIF

!   Check for Grid Space Conflict
            IF (lnc.NE.LIS%D%lnc.OR.lnr.NE.LIS%D%lnr) THEN
               WRITE(*,*) ssibdrv%SSIB_RFILE, &
                          'Grid space mismatch - SSiB HALTED'
               STOP
            ENDIF

!-- Transfer Restart tile space to LIS tile space
            IF (NCH.NE.LIS%D%NCH) THEN
               WRITE(*,*)'Restart Tile Space Mismatch, Halting..'
               stop
            endif

!-- Allocate temporary arrays.
            allocate(TMP_COL(NCH))
            allocate(TMP_ROW(NCH))
            allocate(TMP_FGRD(NCH))
            allocate(TMP_VEGT(NCH))
            allocate(SIB_VEGT(NCH))

!-- NOTE: Next four read statements originally removed from LIS
            READ(40) TMP_COL    !Grid Col of Tile
            READ(40) TMP_ROW    !Grid Row of Tile
            READ(40) TMP_FGRD   !Fraction of Grid covered by tile
            READ(40) TMP_VEGT   !Vegetation Type of Tile
```

```
!*** TESTING
!        open(unit=31, file='test.arrays.ssibrst.txt', &
!      form='formatted')
! write(31,*) '  COL TCOL  ROW TROW  VEGT TVEGT'
! do t=1,nch
!   write(31,310) TILE(t)%COL,TMP_COL(t),TILE(t)%ROW,TMP_ROW(t),&
!         TILE(t)%VEGT,TMP_VEGT(t)
! enddo
!  310   format(6i5)
!   close(31)

!-- Convert maketiles UMD vegetation classes to Noah SiB classes before
!--  testing for tile definition conflict.
            SIB_VEGT = TILE%VEGT
            do t=1,nch
               call SSIB_MAPVEGC(SIB_VEGT(t))
            enddo

!-- Check for tile definition conflict
            if ((sum(TMP_COL - TILE%COL) .ne. 0) .or. &
                (sum(TMP_ROW - TILE%ROW) .ne. 0) .or. &
                (sum(TMP_VEGT - SIB_VEGT) .ne. 0)) then
               write(*,*) 'Restart tile definition mismatch, halting.'
               stop
            endif
            READ(40) SSIB%TCINI
            READ(40) SSIB%TGSINI
            READ(40) SSIB%TDINI
            READ(40) SSIB%TAINI
            READ(40) SSIB%TMINI
            READ(40) SSIB%HTINI
            READ(40) SSIB%QAINI
            DO L=1,3
               READ(40) TMPTILE
               SSIB%WWWINI(L) = TMPTILE
            ENDDO
            DO L=1,2
               READ(40) TMPTILE
               SSIB%CAPACINI(L) = TMPTILE
            ENDDO

            CLOSE(40)

            deallocate(tmptile)
            deallocate(TMP_COL)
            deallocate(TMP_ROW)
            deallocate(TMP_FGRD)
            deallocate(TMP_VEGT)
```

```
            deallocate(SIB_VEGT)
         endif
      endif
      return
```

1.84.2 ssib_scatter.F90 (Source File: ssib_scatter.F90)

Distributes SSiB tiles on to compute nodes

REVISION HISTORY:

```
      Apr 2003: Sujay Kumar, Initial Code
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
    6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
       subroutine ssib_scatter
```

USES:

```
       use tile_spmdMod
       use ssib_varder
       use ssibpardef_module
```

CONTENTS:

```
!=== End Variable List ====================================================
#if (defined SPMD)
      call MPI_SCATTERV(ssib,di_array,displs, &
          MPI_SSIB_STRUCT,ssib,di_array(iam),MPI_SSIB_STRUCT, &
          0,MPI_COMM_WORLD,ierr)
#endif
      return
```

1.84.3 ssib_setup.F90 (Source File: ssib_setup.F90)

Complete the setup routines for SSiB

REVISION HISTORY:

```
    4 Nov 1999: Paul Houser, Initial Code
   28 Apr 2002: Kristi Arsenault, Modified to SSiB LSM 2.5 code to LDAS
    1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
   22 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
subroutine ssib_setup
```

USES:

```
use lisdrv_module, only : lis,tile
use ssib_varder
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
!=== End Variable List ====================================================
#if ( ! defined OPENDAP )
      if ( masterproc ) then
#endif

      call setssibp
      call ssib_gfrac
!      call ssib_alb(lis%d, lis%t, tile)
      call ssib_coldstart

      do t=1,lis%d%nch
         ssib(t)%swnet = 0
         ssib(t)%lwnet = 0
         ssib(t)%qle = 0
         ssib(t)%qh = 0
         ssib(t)%qg = 0
         ssib(t)%qf = 0
         ssib(t)%qtau = 0
         ssib(t)%delsurfheat = 0
         ssib(t)%snowf = 0
         ssib(t)%rainf = 0
         ssib(t)%evap = 0
         ssib(t)%qs = 0
         ssib(t)%qsb = 0
         ssib(t)%qsm = 0
         ssib(t)%delsoilmoist = 0
         ssib(t)%delswe = 0
         ssib(t)%delintercept = 0
         ssib(t)%vegtc = 0
         ssib(t)%baresoilt = 0
         ssib(t)%avgsurft = 0
         ssib(t)%radteff = 0
         ssib(t)%albedo = 0
         ssib(t)%swe = 0
         ssib(t)%sweveg = 0
         ssib(t)%soilmoist1 = 0
         ssib(t)%soilmoist2 = 0
         ssib(t)%soilmoist3 = 0
```

```
         ssib(t)%soiltemp = 0
         ssib(t)%soilwet = 0
         ssib(t)%ecanop = 0
         ssib(t)%tveg = 0
         ssib(t)%esoil = 0
         ssib(t)%rootmoist = 0
         ssib(t)%canopint = 0
         ssib(t)%acond = 0
         ssib(t)%snowfrac = 0

         ssib(t)%count = 0
         ssib(t)%albedocount = 0
      enddo

#if ( ! defined OPENDAP )
      endif

      if ( npes > 1 ) then
         call ssib_scatter
      endif
#endif
      return
```

1.84.4 ssib_singlegather.F90 (Source File: ssib_singlegather.F90)

Gather single variable for output

REVISION HISTORY:

```
22 May 2004: David Mocko, Conversion from HY-SSiB to SSiB
```

INTERFACE:

```
subroutine ssib_singlegather(index, var)
```

USES:

```
use lisdrv_module, only : lis
use tile_spmdMod
use ssib_varder
use ssibpardef_module

implicit none
```

ARGUMENTS:

```
integer :: index             ! Index of SSiB variable
real    :: var(lis%d%glbnch) ! SSiB variable being gathered
```

CONTENTS:

```
      do t = 1,di_array(iam)
         select case (index)
         case(1)
            var_temp(t) = ssib(t)%swnet/float(ssib(t)%count)
         case(2)
            var_temp(t) = ssib(t)%lwnet/float(ssib(t)%count)
         case(3)
            var_temp(t) = ssib(t)%qle/float(ssib(t)%count)
         case(4)
            var_temp(t) = ssib(t)%qh/float(ssib(t)%count)
         case(5)
            var_temp(t) = ssib(t)%qg/float(ssib(t)%count)
         case(6)
            var_temp(t) = ssib(t)%qf/float(ssib(t)%count)
         case(7)
            var_temp(t) = ssib(t)%qtau/float(ssib(t)%count)
         case(8)
            var_temp(t) = ssib(t)%delsurfheat
         case(9)
            var_temp(t) = ssib(t)%snowf/float(ssib(t)%count)
         case(10)
            var_temp(t) = ssib(t)%rainf/float(ssib(t)%count)
         case(11)
            var_temp(t) = ssib(t)%evap/float(ssib(t)%count)
         case(12)
            var_temp(t) = ssib(t)%qs/float(ssib(t)%count)
         case(13)
            var_temp(t) = ssib(t)%qsb/float(ssib(t)%count)
         case(14)
            var_temp(t) = ssib(t)%qsm/float(ssib(t)%count)
         case(15)
            var_temp(t) = ssib(t)%delsoilmoist
         case(16)
            var_temp(t) = ssib(t)%delswe
         case(17)
            var_temp(t) = ssib(t)%delintercept
         case(18)
            if (ssibdrv%STATEVAR_AVG.eq.1) then
               var_temp(t) = ssib(t)%vegtc/float(ssib(t)%count)
            else
               var_temp(t) = ssib(t)%vegtc
            endif
         case(19)
            if (ssibdrv%STATEVAR_AVG.eq.1) then
               var_temp(t) = ssib(t)%baresoilt/float(ssib(t)%count)
            else
```

```
      var_temp(t) = ssib(t)%baresoilt
   endif
case(20)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%avgsurft/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%avgsurft
   endif
case(21)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%radteff/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%radteff
   endif
case(22)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      if (ssib(t)%albedocount.gt.0) then
         var_temp(t) = ssib(t)%albedo/float(ssib(t)%albedocount)
      else
         var_temp(t) = lis%d%udef
      endif
   else
      var_temp(t) = ssib(t)%albedo
   endif
case(23)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%swe/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%swe
   endif
case(24)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%sweveg/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%sweveg
   endif
case(25)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%soilmoist1/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%soilmoist1
   endif
case(26)
   if (ssibdrv%STATEVAR_AVG.eq.1) then
      var_temp(t) = ssib(t)%soilmoist2/float(ssib(t)%count)
   else
      var_temp(t) = ssib(t)%soilmoist2
   endif
```

```
         case(27)
            if (ssibdrv%STATEVAR_AVG.eq.1) then
               var_temp(t) = ssib(t)%soilmoist3/float(ssib(t)%count)
            else
               var_temp(t) = ssib(t)%soilmoist3
            endif
         case(28)
            if (ssibdrv%STATEVAR_AVG.eq.1) then
               var_temp(t) = ssib(t)%soiltemp/float(ssib(t)%count)
            else
               var_temp(t) = ssib(t)%soiltemp
            endif
         case(29)
            if (ssibdrv%STATEVAR_AVG.eq.1) then
               var_temp(t) = ssib(t)%soilwet/float(ssib(t)%count)
            else
               var_temp(t) = ssib(t)%soilwet
            endif
         case(30)
            var_temp(t) = ssib(t)%ecanop/float(ssib(t)%count)
         case(31)
            var_temp(t) = ssib(t)%tveg/float(ssib(t)%count)
         case(32)
            var_temp(t) = ssib(t)%esoil/float(ssib(t)%count)
         case(33)
            var_temp(t) = ssib(t)%rootmoist/float(ssib(t)%count)
         case(34)
            var_temp(t) = ssib(t)%canopint/float(ssib(t)%count)
         case(35)
            var_temp(t) = ssib(t)%acond/float(ssib(t)%count)
         case(36)
            var_temp(t) = ssib(t)%snowfrac/float(ssib(t)%count)
         case(37)
            if (lis%o%wfor.eq.1) then
               var_temp(t) = sqrt(ssib(t)%forcing(5)*ssib(t)%forcing(5)+ &
               ssib(t)%forcing(6)*ssib(t)%forcing(6))
            endif
         case(38)
            if (lis%o%wfor.eq.1) then
               if (ssib(t)%forcing(1).lt.273.15) then
                  var_temp(t) = 0.0
               else
                  var_temp(t) = ssib(t)%forcing(8)
               endif
            endif
         case(39)
            if (lis%o%wfor.eq.1) then
               if (ssib(t)%forcing(1).lt.273.15) then
```

```
                 var_temp(t) = ssib(t)%forcing(8)
              else
                 var_temp(t) = 0.0
              endif
           endif
        case(40)
           if (lis%o%wfor.eq.1) then
              var_temp(t) = ssib(t)%forcing(1)
           endif
        case(41)
           if (lis%o%wfor.eq.1) then
              var_temp(t) = ssib(t)%forcing(2)
           endif
        case(42)
           if (lis%o%wfor.eq.1) then
              var_temp(t) = ssib(t)%forcing(7)
           endif
        case(43)
           if (lis%o%wfor.eq.1) then
              var_temp(t) = ssib(t)%forcing(3)
           endif
        case(44)
           if (lis%o%wfor.eq.1) then
              var_temp(t) = ssib(t)%forcing(4)
           endif
        end select
     enddo
#if (defined SPMD)
     call MPI_GATHERV(var_temp(1:di_array(iam)),di_array(iam), &
          MPI_REAL,var,di_array,displs,MPI_REAL,0,MPI_COMM_WORLD,ierr)
#endif
     return
```

1.84.5 ssib_singleout.F90 (Source File: ssib_singleout.F90)

Write output file for a single HY-SSiB variable

REVISION HISTORY:

```
14 Jun 2002: Sujay Kumar, Initial Specification
22 May 2004: David Mocko, Conversion from HY-SSiB to SSiB
```

INTERFACE:

```
subroutine ssib_singleout(ld,tile,gindex,var_array,index)
```

USES:

```
use lis_module            ! LIS non-model-specific 1-D variables
use tile_module           ! LIS non-model-specific tile variables
use ssib_varder, only : ssibdrv
use drv_output_mod, only : t2gr

implicit none
```

ARGUMENTS:

```
type (lisdec)  :: ld       !data structure for lis domain specific variables
type (tiledec) :: tile(ld%d%glbnch) !tile array for the modeled domain
integer        :: gindex(ld%d%lnc, ld%d%lnr) !2-d array for mapping from 2d to 1d
real           :: var_array(ld%d%glbnch) !array of variable that is being output
integer        :: index   !Index of the output varible in the ALMA list.
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
      IF (MOD(LD%T%GMT,ssibdrv%WRITEINTN).EQ.0) THEN
         ssibdrv%NUMOUTNH=ssibdrv%NUMOUTNH+1
         length = len(trim(vname1(index)))
         WRITE(UNIT=temp1,FMT='(A10)') VNAME1(index)
         READ(UNIT=temp1,FMT='(10A1)') (FVARNAME(I),I=1,length)
         WRITE(UNIT=temp1,FMT='(I4,I2,I2)') LD%T%YR,LD%T%MO,LD%T%DA
         READ(UNIT=temp1,FMT='(8A1)') FTIME
         DO I=1,8
            IF (FTIME(I).EQ.(' ')) FTIME(I)='0'
         ENDDO
         WRITE(UNIT=temp1,FMT='(I4)') LD%T%YR
         READ(UNIT=temp1,FMT='(8A1)') FTIMEC
         DO I=1,4
            IF (FTIMEC(I).EQ.(' ')) FTIMEC(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A8,I3,A1)') '/LIS.EXP',LD%O%EXPCODE,'.'
         READ(UNIT=temp1,FMT='(80A1)') (FNAME(I),I=1,12)
         DO I=1,12
            IF (FNAME(I).EQ.(' ')) FNAME(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A40)') LD%O%ODIR
         READ(UNIT=temp1,FMT='(40A1)') (FBASE(I),I=1,40)
         C=0
         DO I=1,40
            IF (FBASE(I).EQ.(' ').AND.C.EQ.0) C=I-1
         ENDDO
```

```
         WRITE(UNIT=temp1,FMT='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
               LD%O%EXPCODE,'/SSIB/', &
               LD%T%YR,'/',LD%T%YR,LD%T%MO,LD%T%DA
         READ(UNIT=temp1,FMT='(80A1)') (FYRMODIR(I),I=1,26)
         DO I=1,26
            IF (FYRMODIR(I).EQ.(' ')) FYRMODIR(I)='0'
         ENDDO

         WRITE(UNIT=temp1,FMT='(A9)')'mkdir -p '
         READ(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9)

         WRITE(UNIT=temp1,FMT='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
                                       (FYRMODIR(I),I=1,26)
         READ(UNIT=temp1,FMT='(A80)') MKFYRMO
         CALL SYSTEM(MKFYRMO)
!-----------------------------------------------------------------------
! Generate file name for BINARY output
!-----------------------------------------------------------------------
         IF (LD%O%WOUT.EQ.1) THEN
            WRITE(UNIT=FBINNAME,FMT='(I4,I2,I2,I2)') LD%T%YR,LD%T%MO, &
                                                     LD%T%DA,LD%T%HR
            READ(UNIT=FBINNAME,FMT='(10A1)') FTIMEB
            DO I=1,10
               IF (FTIMEB(I).EQ.(' ')) FTIMEB(I)='0'
            ENDDO
            WRITE(UNIT=FBINNAME,FMT='(A9)') '.SSIBgbin'
            READ(UNIT=FBINNAME,FMT='(80A1)') (FSUBGB(I),I=1,9)
            WRITE(UNIT=FBINNAME,FMT='(80A1)')(FBASE(I),I=1,C), &
                      (FYRMODIR(I),I=1,26), &
                      (FNAME(I),I=1,12),(FTIMEB(I),I=1,10), &
                      (FVARNAME(I),I=1,length),(FSUBGB(I),I=1,9)
            READ(UNIT=FBINNAME,FMT='(A80)') FILENGB
!-----------------------------------------------------------------------
! Open statistical output file
!-----------------------------------------------------------------------
            IF (ssibdrv%SSIBopen.EQ.0) THEN
               FILE='SSIBstats.dat'
               CALL OPENFILE(NAME,LD%O%ODIR,LD%O%EXPCODE,FILE)
               IF (LD%O%STARTCODE.EQ.1) THEN
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='UNKNOWN', &
                          POSITION='APPEND')
               ELSE
                  OPEN(65,FILE=NAME,FORM='FORMATTED',STATUS='REPLACE')
               ENDIF
               ssibdrv%SSIBopen=1
            ENDIF

            WRITE(65,996)'        Statistical Summary of SSiB Output for:  ', &
```

```
                 LD%T%MO,'/',LD%T%DA,'/',LD%T%YR,LD%T%HR,':',LD%T%MN,':', &
                 LD%T%SS
996        FORMAT(A47,I2,A1,I2,A1,I4,1X,I2,A1,I2,A1,I2)
           WRITE(65,*)
           WRITE(65,997)
997        FORMAT(T27,'Mean',T41,'StDev',T56,'Min',T70,'Max')
        ENDIF

        if (ld%o%wout.eq.1) then
           open(58,file=filengb,form='unformatted')
        endif
        if (ld%o%wout.eq.1) then
           call t2gr(var_array,gtmp,ld%d%glbngrid,ld%d%glbnch,tile);
           write(58) gtmp
           call stats(var_array,ld%d%udef,ld%d%glbnch, vmean,vstdev, vmin, vmax);
           write(65,999) vname(index),vmean,vstdev,vmin,vmax
        endif

998     FORMAT(1X,A18,4E14.3)
999     FORMAT(1X,A18,4F14.3)
     endif
     return
```

1.84.6 ssib_totinit.F90 (Source File: ssib_totinit.F90)

Initialize SSiB output arrays

REVISION HISTORY:

```
22 May 2004: David Mocko, Conversion from HY-SSiB to SSiB
```

INTERFACE:

```
subroutine ssib_totinit
```

USES:

```
use ssib_varder         ! SSiB module
use tile_spmdMod
use lisdrv_module, only : lis
```

CONTENTS:

```
!=== End Variable List ====================================================
      do t = 1,di_array(iam)
         ssib(t)%swnet = 0
         ssib(t)%lwnet = 0
         ssib(t)%qle = 0
```

```
      ssib(t)%qh = 0
      ssib(t)%qg = 0
      ssib(t)%qf = 0
      ssib(t)%qtau = 0
      ssib(t)%delsurfheat = 0
      ssib(t)%snowf = 0
      ssib(t)%rainf = 0
      ssib(t)%evap = 0
      ssib(t)%qs = 0
      ssib(t)%qsb = 0
      ssib(t)%qsm = 0
      ssib(t)%delsoilmoist = 0
      ssib(t)%delswe = 0
      ssib(t)%delintercept = 0
      ssib(t)%vegtc = 0
      ssib(t)%baresoilt = 0
      ssib(t)%avgsurft = 0
      ssib(t)%radteff = 0
      ssib(t)%albedo = 0
      ssib(t)%swe = 0
      ssib(t)%sweveg = 0
      ssib(t)%soilmoist1 = 0
      ssib(t)%soilmoist2 = 0
      ssib(t)%soilmoist3 = 0
      ssib(t)%soiltemp = 0
      ssib(t)%soilwet = 0
      ssib(t)%ecanop = 0
      ssib(t)%tveg = 0
      ssib(t)%esoil = 0
      ssib(t)%rootmoist = 0
      ssib(t)%canopint = 0
      ssib(t)%acond = 0
      ssib(t)%snowfrac = 0

      ssib(t)%count = 0
      ssib(t)%albedocount = 0
   enddo
   return
```

1.84.7 ssib_varder.F90 (Source File: ssib_varder.F90)

Module for 1-D SSiB land model driver variable initialization

REVISION HISTORY:

```
Apr 2003: Sujay Kumar, Initial Code
```

```
1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
6 May 2004: David Mocko, made compatible with SiB-lings
```

INTERFACE:

```
module ssib_varder
```

USES:

```
use ssib_module
use tile_spmdMod
use ssibpardef_module
use ssibdrv_module
```

1.84.8 ssib_varder_ini (Source File: ssib_varder.F90)

Reads in runtime ssib parameters, allocates memory for variables

INTERFACE:

```
subroutine ssib_varder_ini(nch)
```

USES:

```
#if ( defined OPENDAP )
      use opendap_module
#endif
```

CONTENTS:

```
      if (masterproc) then
         call readssibcrd(ssibdrv)
      endif
      call def_ssibpar_struct
#if (defined SPMD)
      call MPI_BCAST(ssibdrv, 1, MPI_SSIBDRV_STRUCT, 0, &
                     MPI_COMM_WORLD, ierr)
#endif
      if (masterproc) then
         allocate(ssib(nch))
      else
         allocate(ssib(di_array(iam)))
      endif
      return
      end subroutine ssib_varder_ini
```

1.84.9 ssib_writerst.F90 (Source File: ssib_writerst.F90)

This program writes restart files for SSiB. This includes all relevant water/energy storages, tile information, and time information. It also rectifies changes in the tile space.

REVISION HISTORY:

```
 1 Oct 1999: Jared Entin, Initial Code
15 Oct 1999: Paul Houser, Significant F90 Revision
05 Sep 2001: Brian Cosgrove, Modified code to use Dag Lohmann's NOAA
             initial conditions if necessary.  This is controlled with
             local variable NOAAIC.  Normally set to 0 in this subroutine
             but set to 1 if want to use Dag's NOAA IC's.  Changed output
             directory structure, and commented out if-then check so that
             directory is always made.
28 Apr 2002: Kristi Arsenault, Added SSIB LSM into LDAS
28 May 2002: Kristi Arsenault, For STARTCODE=4, corrected SNEQV values
             and put SMC, SH2O, STC limit for GDAS and GEOS forcing.
14 Jun 2003: Sujay Kumar, Separated the write restart from the original code
30 Oct 2003: Matt Rodell, Added back COL,ROW,FGRD,VEGT write statements
 1 Mar 2004: Luis Gustavo G de Goncalves, SSiB in LIS
 6 May 2004: David Mocko, made compatible with SiB-lings
RESTART FILE FORMAT(fortran sequential binary):
 YR,MO,DA,HR,MN,SS,VCLASS,NCH !Restart time,Veg class,no.tiles, no.soil lay
 TILE(NCH)%COL        !Grid Col of Tile
 TILE(NCH)%ROW        !Grid Row of Tile
 TILE(NCH)%FGRD       !Fraction of Grid covered by Tile
 TILE(NCH)%VEGT       !Vegetation Type of Tile
 SSIB(NCH)%STATES     !Model States in Tile Space
```

INTERFACE:

```
      subroutine ssib_writerst
```

USES:

```
      use lisdrv_module, only : lis,tile
      use ssib_varder        ! SSiB tile variables
      use time_manager
      use tile_spmdMod
```

CONTENTS:

```
!=== End Variable Definition ============================================

      if (masterproc) then
!=== Restart Writing (2 files are written = active and archive)

         IF ((lis%t%gmt.eq.(24-ssibdrv%writeintn)) &
            .or.lis%t%endtime.eq.1) THEN
            allocate(tmptilen(lis%d%nch))
```

```
!Active archive restart
            OPEN(40,FILE=SSIBDRV%SSIB_RFILE,FORM='unformatted')
            call timemgr_write_restart(40)

!Veg class, no tiles
            WRITE(40) LIS%P%VCLASS,LIS%D%lnc,LIS%D%lnr,LIS%D%NCH

!== NOTE: Next four write statements originally commented out in LIS
            WRITE(40) TILE%COL  !Grid Col of Tile
            WRITE(40) TILE%ROW  !Grid Row of Tile
            WRITE(40) TILE%FGRD !Fraction of Grid covered by tile
            WRITE(40) TILE%VEGT !Vegetation Type of Tile
!            WRITE(40) SSIB%T1   !SSIB Skin Temperature (K)
!            WRITE(40) SSIB%CMC  !SSIB Canopy Water Content
!            WRITE(40) SSIB%SNOWH !SSIB Actual Snow Depth
!            WRITE(40) SSIB%SNEQV !SSIB Water Equivalent Snow Depth
            WRITE(40) SSIB%TCINI
            WRITE(40) SSIB%TGSINI
            WRITE(40) SSIB%TDINI
            WRITE(40) SSIB%TAINI
            WRITE(40) SSIB%TMINI
            WRITE(40) SSIB%HTINI
            WRITE(40) SSIB%QAINI
            DO L=1,3
               DO T=1,LIS%D%NCH
                  TMPTILEN(T)=SSIB(T)%WWWINI(L)
                  ttii = SSIB(T)%WWWINI(L) + 100.
                  IF (ttii.eq.SSIB(T)%WWWINI(L)) THEN
                     WRITE (*,*) '--> Variable ',ttii,SSIB(T)%WWWINI(L),T
                     STOP
                  ENDIF
               ENDDO
               WRITE(40) TMPTILEN !SSiB Soil Moisture (3 layers)
!               WRITE(40) SSIB%WWWINI(L) !SSiB Soil Moisture (3 layers)
            ENDDO
            DO L=1,2
               DO T=1,LIS%D%NCH
                  TMPTILEN(T)=SSIB(T)%CAPACINI(L)
               ENDDO
               WRITE(40) TMPTILEN !SSIB Water on canopy/ground (2 layers)
            ENDDO

            CLOSE(40)

            WRITE(*,*)'SSiB Active Restart Written: ',SSIBDRV%SSIB_RFILE

            WRITE(unit=temp,fmt='(i4,i2,i2,i2)') LIS%T%YR,LIS%T%MO, &
                                                 LIS%T%DA,LIS%T%HR
```

```
            READ(UNIT=temp,fmt='(10A1)') FTIME
            DO I=1,10
               IF (FTIME(I).EQ.(' ')) FTIME(I)='0'
            ENDDO
            WRITE(UNIT=temp,fmt='(A4,I3,A6,I4,A1,I4,I2,I2,A8,I3,A1)') &
                  '/EXP',LIS%O%EXPCODE,'/SSIB/',LIS%T%YR,'/',LIS%T%YR, &
                  LIS%T%MO,LIS%T%DA,'/LIS.EXP',LIS%O%EXPCODE,'.'
            READ(UNIT=temp,fmt='(80a1)') (FNAME(I),I=1,38)
            DO I=1,73
               IF (FNAME(I).EQ.(' ')) FNAME(I)='0'
            ENDDO

            WRITE(UNIT=temp,fmt='(a9)') 'mkdir -p '
            READ(UNIT=temp,fmt='(80a1)') (FMKDIR(I),I=1,9)
            WRITE(UNIT=temp,fmt='(A4,I3,A6,I4,A1,I4,I2,I2)') &
                  '/EXP',LIS%O%EXPCODE,'/SSIB/', &
                  LIS%T%YR,'/',LIS%T%YR,LIS%T%MO,LIS%T%DA
            READ(UNIT=temp,fmt='(80a1)') (FYRMODIR(I),I=1,26)
            DO I=1,26
               IF(FYRMODIR(I).EQ.(' '))FYRMODIR(I)='0'
            ENDDO

            WRITE(UNIT=temp,fmt='(a8)')'.SSIBrst'
            READ(UNIT=temp,FMT='(80A1)') (FSUBS(I),I=1,8)

            WRITE(UNIT=temp,fmt='(a40)') LIS%O%ODIR
            READ(UNIT=temp,FMT='(80A1)') (FBASE(I),I=1,80)
            C=0
            DO I=1,80
               IF (FBASE(I).EQ.(' ').AND.C.EQ.0) C=I-1
            ENDDO
            WRITE(UNIT=temp,FMT='(80A1)') (FBASE(I),I=1,C), &
                 (FNAME(I),I=1,38),(FTIME(I),I=1,10),(FSUBS(I),I=1,8)
            READ(UNIT=temp,fmt='(a80)') FILEN

            WRITE(UNIT=temp,FMT='(80A1)')(FMKDIR(I),I=1,9), &
                 (FBASE(I),I=1,C),(FYRMODIR(I),I=1,26)
            READ(UNIT=temp,fmt='(a80)') MKFYRMO

!== Archive File Name Generation Complete
!== Make the directories for the SSIB restart file
            CALL SYSTEM(MKFYRMO)

!== Archive File Name Generation Complete
            print *,filen
            OPEN(40,FILE=FILEN,status='unknown',FORM='unformatted')
            call timemgr_write_restart(40)
!Veg class, no. tiles
```

```
            WRITE(40) LIS%P%VCLASS,LIS%D%lnc,LIS%D%lnr,LIS%D%NCH
!== NOTE: Next four write statements originally commented out in LIS
            WRITE(40) TILE%COL  !Grid Col of Tile
            WRITE(40) TILE%ROW  !Grid Row of Tile
            WRITE(40) TILE%FGRD !Fraction of Grid covered by Tile
            WRITE(40) TILE%VEGT !Vegetation Type of Tile
            WRITE(40) SSIB%TCINI
            WRITE(40) SSIB%TGSINI
            WRITE(40) SSIB%TDINI
            WRITE(40) SSIB%TAINI
            WRITE(40) SSIB%TMINI
            WRITE(40) SSIB%HTINI
            WRITE(40) SSIB%QAINI
            DO L=1,3
               DO T=1,LIS%D%NCH
                  TMPTILEN(T)=SSIB(T)%WWWINI(L)
               ENDDO
               WRITE(40) TMPTILEN !SSiB Soil Moisture (3 layers)
            ENDDO
            DO L=1,2
               DO T=1,LIS%D%NCH
                  TMPTILEN(T)=SSIB(T)%CAPACINI(L)
               ENDDO
               WRITE(40) TMPTILEN !SSiB Water on canopy/ground (2 layers)
            ENDDO

            CLOSE(40)

            WRITE(*,*)'SSiB Archive Restart Written: ',FILEN
            deallocate(tmptilen)
         ENDIF                  !End restart writing (active and archive files)
      endif                     ! if masterproc

      return
```

1.84.10 ssib_tileout.F90 (Source File: ssib_writestats.F90)

LIS HY-SSiB data writer: Writes HY-SSiB output in tile space

REVISION HISTORY:

```
02 Dec 2003: Sujay Kumar, Initial Version
22 May 2004: David Mocko, Conversion from HY-SSiB to SSiB
```

INTERFACE:

```
subroutine ssib_writestats(ftn_stats)
```

USES:

```
use lisdrv_module, only : lis, tile
use ssib_varder

    implicit none
```

ARGUMENTS:

```
    integer :: ftn_stats
```

CONTENTS:

```
      do t=1,lis%d%glbnch
         if (ssib(t)%forcing(1).lt.273.15) then
            rainf(t) = 0.0
            snowf(t) = ssib(t)%forcing(8)
         else
            rainf(t) = ssib(t)%forcing(8)
            snowf(t) = 0.0
         endif
      enddo

!-----------------------------------------------------------------------
! General Energy Balance Components
!-----------------------------------------------------------------------
      call stats(ssib%swnet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SWnet(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%lwnet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'LWnet(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%qle,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Qle(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%qh,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Qh(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%qg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Qg(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%qf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Qf(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%qtau,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Qtau(W/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%delsurfheat,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'DelSurfHeat(J/m2)',vmean,vstdev,vmin,vmax
```

```
!-----------------------------------------------------------------------
! General Water Balance Components
!-----------------------------------------------------------------------
      call stats(ssib%snowf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Snowf(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%rainf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Rainf(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%evap,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Evap(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%qs,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Qs(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%qsb,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Qsb(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%qsm,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'Qsm(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%delsoilmoist,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'DelSoilMoist(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%delswe,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'DelSWE(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%delintercept,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'DelIntercept(kg/m2)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Surface State Variables
!-----------------------------------------------------------------------
      if (ssibdrv%STATEVAR_AVG.eq.1) then
         do t = 1,lis%d%glbnch
            if (ssib(t)%albedocount.gt.0) then
               ssib(t)%albedo = ssib(t)%albedo/float(ssib(t)%albedocount)
            else
               ssib(t)%albedo = lis%d%udef
            endif
         enddo
         ssib%vegtc = ssib%vegtc/float(ssib%count)
         ssib%baresoilt = ssib%baresoilt/float(ssib%count)
         ssib%avgsurft = ssib%avgsurft/float(ssib%count)
         ssib%radteff = ssib%radteff/float(ssib%count)
         ssib%swe = ssib%swe/float(ssib%count)
         ssib%sweveg = ssib%sweveg/float(ssib%count)
      endif
```

```
      call stats(ssib%vegtc,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'VegT(K)',vmean,vstdev,vmin,vmax

      call stats(ssib%baresoilt,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'BaresoilT(K)',vmean,vstdev,vmin,vmax

      call stats(ssib%avgsurft,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'AvgSurfT(K)',vmean,vstdev,vmin,vmax

      call stats(ssib%radteff,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'RadT(K)',vmean,vstdev,vmin,vmax

      call stats(ssib%albedo,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'Albedo(-)',vmean,vstdev,vmin,vmax

      call stats(ssib%swe,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SWE(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%sweveg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SWEVeg(kg/m2)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Subsurface State Variables
!-----------------------------------------------------------------------
      if (ssibdrv%STATEVAR_AVG.eq.1) then
         ssib%soilmoist1 = ssib%soilmoist1/float(ssib%count)
         ssib%soilmoist2 = ssib%soilmoist2/float(ssib%count)
         ssib%soilmoist3 = ssib%soilmoist3/float(ssib%count)
         ssib%soiltemp = ssib%soiltemp/float(ssib%count)
         ssib%soilwet = ssib%soilwet/float(ssib%count)
      endif

      call stats(ssib%soilmoist1,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SoilMoist1(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%soilmoist2,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SoilMoist2(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%soilmoist3,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SoilMoist3(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%soiltemp,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SoilTemp(K)',vmean,vstdev,vmin,vmax

      call stats(ssib%soilwet,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SoilWet(-)',vmean,vstdev,vmin,vmax
```

```
!-----------------------------------------------------------------------
! Evaporation Components
!-----------------------------------------------------------------------
      call stats(ssib%ecanop,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'ECanop(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%tveg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'TVeg(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%esoil,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'ESoil(kg/m2s)',vmean,vstdev,vmin,vmax

      call stats(ssib%rootmoist,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'RootMoist(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%canopint,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,998) 'CanopInt(kg/m2)',vmean,vstdev,vmin,vmax

      call stats(ssib%acond,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'ACond(m/s)',vmean,vstdev,vmin,vmax

      call stats(ssib%snowfrac,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
      write(ftn_stats,999) 'SnowFrac(-)',vmean,vstdev,vmin,vmax

!-----------------------------------------------------------------------
! Cold Season Processes
!-----------------------------------------------------------------------

!-----------------------------------------------------------------------
! Forcing Variables
!-----------------------------------------------------------------------
      if (lis%o%wfor.eq.1) then
         call stats(sqrt(ssib%forcing(5)*ssib%forcing(5)+ &
                    ssib%forcing(6)*ssib%forcing(6)), &
                    lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'Wind(m/s)',vmean,vstdev,vmin,vmax

         call stats(rainf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,998) 'Rainf(kg/m2s)',vmean,vstdev,vmin,vmax

         call stats(snowf,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,998) 'Snowf(kg/m2s)',vmean,vstdev,vmin,vmax

         call stats(ssib%forcing(1),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'Tair(K)',vmean,vstdev,vmin,vmax

         call stats(ssib%forcing(2),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'Qair(kg/kg)',vmean,vstdev,vmin,vmax
```

```
         call stats(ssib%forcing(7),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'PSurf(Pa)',vmean,vstdev,vmin,vmax

         call stats(ssib%forcing(3),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'SWdown (W/m2)',vmean,vstdev,vmin,vmax

         call stats(ssib%forcing(4),lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
         write(ftn_stats,999) 'LWdown(W/m2)',vmean,vstdev,vmin,vmax

      endif

 998  FORMAT(1X,A18,4E14.3)
 999  FORMAT(1X,A18,4F14.3)
      return
```

!ROUTINE:canhtset.F90
This subroutine reads in canopy height information into CLM2 for now

REVISION HISTORY:

```
    15 Nov 2002: Jon Gottschalck; Initial code
```

INTERFACE:

```
 subroutine canhtset ()
```

USES:

```
   use clm_varder          ! CLM2 tile variables
   use clm_varpar     , only : maxpatch
   use clm_varmap     , only : numpatch
   use clm_varctl     , only : clmdrv
   use lis_module
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Open canopy heights file and read into temporary variables
!-----------------------------------------------------------------------
  open(57,file=clmdrv%clm2_chtfile,status='old')
  read(57,*) (pthtop(i),i=1,maxpatch)
  read(57,*) (pthbot(i),i=1,maxpatch)
  close(57)
!-----------------------------------------------------------------------
! Assign canopy top and height values for each
! vegetation type to CLM2 tiles
!-----------------------------------------------------------------------
  do t=1,numpatch
```

```
   clm(t)%htop = pthtop(clm(t)%itypveg)
   clm(t)%hbot = pthbot(clm(t)%itypveg)
  enddo
  return
```

1.84.11 clm2_almaout.F90 (Source File: clm2_almaout.F90)

CLM output writer.

REVISION HISTORY:

```
   29 Oct. 1999: Jon Radakovich; Initial code
   27 Sep. 2000: Brian Cosgrove; Major revisions to enable CLM to
                 output ALMA/LDAS variables
   27 Sep. 2000: Added abitrary root zone cutoff value of .05 so
                 that root zone is considered only those levels
                 with a rooting value >= .05
   19 Mar  2000: Cosgrove; Changed code to allow LDAS GRIB output.  Removed
                 calls to LATS4D that had been used for grib output and
                 added a call to griboutclm which outputs GRIB CLM data.
   05 Apr 2002: Jon Gottschalck; Modified code to work with CLM2
   15 Jun 2003; Sujay Kumar; ALMA version of the output routine
```

INTERFACE:

```
 subroutine clm2_almaout ()
```

USES:

```
   use lisdrv_module, only : lis, tile
   use clm_varctl, only : clmdrv
 #if ( defined USE_NETCDF )
   use netcdf
 #endif
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
  if(mod(lis%t%gmt,clmdrv%writeintc2).eq.0)then
!-----------------------------------------------------------------------
! Generate directory structure and file names for CLM Output
!-----------------------------------------------------------------------
     write(unit=temp,fmt='(i4,i2,i2)')lis%t%yr,lis%t%mo,lis%t%da
     read(unit=temp,fmt='(8a1)')ftime
     do i=1,8
```

```
         if(ftime(i).eq.(' '))ftime(i)='0'
      enddo

      write(unit=temp,fmt='(i4)')lis%t%yr
      read(unit=temp,fmt='(8a1)')ftimec
      do i=1,4
         if(ftimec(i).eq.(' '))ftimec(i)='0'
      enddo
#if 0
      write(unit=temp,fmt='(a6,i3,a1)')'/LIS.E',lis%o%expcode,'.'
      read(unit=temp,fmt='(80a1)') (fname(i),i=1,11)
      do i=1,10
         if(fname(i).eq.(' '))fname(i)='0'
      enddo
#endif
      write(unit=temp,fmt='(a40)') lis%o%odir
      read(unit=temp,fmt='(40a1)') (fbase(i),i=1,40)
      c=0
      do i=1,40
         if(fbase(i).eq.(' ').and.c.eq.0)c=i-1
      enddo

      write(unit=temp,fmt='(a4,i3,a6,i4,a1,i4,i2,i2)')'/EXP', &
           lis%o%expcode,'/CLM2/', &
           lis%t%yr,'/',lis%t%yr,lis%t%mo,lis%t%da
      read(unit=temp,fmt='(80a1)') (fyrmodir(i),i=1,26)
      do i=1,26
         if(fyrmodir(i).eq.(' '))fyrmodir(i)='0'
      enddo

      write(unit=temp,fmt='(a9)')'mkdir -p '
      read(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9)

      write(unit=temp,fmt='(80a1)')(fmkdir(i),i=1,9),(fbase(i),i=1,c), &
           (fyrmodir(i),i=1,26)
      read(unit=temp,fmt='(a80)')mkfyrmo

!-----------------------------------------------------------------------
! Make the directories for the CLM2 output files
!-----------------------------------------------------------------------
      call system(mkfyrmo)
!-----------------------------------------------------------------------
! Generate file name for binary output
!-----------------------------------------------------------------------
      write(unit=fbinname, fmt='(i4,i2,i2,i2)') lis%t%yr,lis%t%mo, &
           lis%t%da,lis%t%hr
      read(unit=fbinname,fmt='(10a1)') ftimeb
      do i=1,10
```

```
         if(ftimeb(i).eq.(' '))ftimeb(i)='0'
      enddo
      if(lis%o%wout.eq.1) then
         write(unit=fbinname,fmt='(a5)') '.gs4r'
         read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
      elseif(lis%o%wout.eq.2) then
         write(unit=fbinname,fmt='(a5)') '.grb '
         read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
      elseif(lis%o%wout.eq.3) then
         write(unit=fbinname,fmt='(a5)') '.nc  '
         read(unit=fbinname,fmt='(80a1)') (fsubgb(i),i=1,5)
      endif
         write(unit=fbinname,fmt='(69a1)')(fbase(i),i=1,c), &
              (fyrmodir(i),i=1,26),'/',&
              (ftimeb(i),i=1,10), &
              (fsubgb(i),i=1,5)
         read(unit=fbinname,fmt='(a69)')filengb
      clmdrv%numoutc2=clmdrv%numoutc2+1    !Counts number of output times
!-----------------------------------------------------------------------
! Write statistical output
!-----------------------------------------------------------------------
      if(clmdrv%clm2open.eq.0)then
         file='CLMstats.dat'
         call openfile(name,lis%o%odir,lis%o%expcode,file)
         if(lis%o%startcode.eq.1)then
            open(60,file=name,form='formatted',status='unknown', &
                 position='append')
         else
            open(60,file=name,form='formatted',status='replace')
         endif
         clmdrv%clm2open=1
      endif

      write(60,996)'       Statistical Summary of CLM Output for:  ', &
           lis%t%mo,'/',lis%t%da,'/',lis%t%yr,lis%t%hr,':', &
           lis%t%mn,':',lis%t%ss
996   format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
      write(60,*)
      write(60,997)
997   format(t26,'Mean',t40,'StDev',t54,'Min',t68,'Max')

      ftn = 57
      if(lis%o%wout.eq.1)then
         open(ftn,file=filengb,form='unformatted')
         call clm2_binout(ftn)
         close(ftn)
      elseif(lis%o%wout.eq.2)then
           call baopen (ftn,filengb, iret)
```

```
          call clm2_gribout(ftn)
          call baclose(ftn,iret)
     elseif(lis%o%wout.eq.3) then !netcdf
#if ( defined USE_NETCDF )
          iret = nf90_create(path=trim(filengb),cmode=nf90_clobber,ncid=ftn)
          call noah_netcdfout(ftn)
          iret = nf90_close(ftn)
#endif
     endif
     call clm2_writestats(60)
  end if
```

1.84.12 clm2_binout.F90 (Source File: clm2_binout.F90)

LIS CLM2 data writer: Write CLM2 output in binary format

REVISION HISTORY:

```
02 Dec  2003: Sujay Kumar; Initial Version
```

INTERFACE:

```
subroutine clm2_binout(ftn)
```

USES:

```
use lisdrv_module, only : lis
use drv_output_mod, only : drv_writevar_bin
use clm_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, only : nlevsoi
use clm_varmap, only : patchvec
use clm_varder
```

CONTENTS:

```
  soilmtc=0.0
  delsoilmoist = 0.0
  delswe = 0.0
  soilmr=0.0
  soilwtc=0.0

!-----------------------------------------------------------------------
! Net Surface Shortwave (absorbed) Radiation (W/m2)
!-----------------------------------------------------------------------
  clm%totfsa=clm%totfsa/float(clm%count)
  call drv_writevar_bin(ftn,clm%totfsa)
!-----------------------------------------------------------------------
```

```
! Net Surface Longwave Radiation (W/m2)
!-----------------------------------------------------------------------
  clm%toteflx_lwrad_net=-1.0*clm%toteflx_lwrad_net/float(clm%count)
  call drv_writevar_bin(ftn,clm%toteflx_lwrad_net)
!-----------------------------------------------------------------------
! Latent Heat Flux (W/m2)
!-----------------------------------------------------------------------
  clm%toteflx_lh_tot=clm%toteflx_lh_tot/float(clm%count)
  call drv_writevar_bin(ftn,clm%toteflx_lh_tot)
!-----------------------------------------------------------------------
! Sensible Heat Flux (W/m2)
!-----------------------------------------------------------------------
  clm%toteflx_sh_tot=clm%toteflx_sh_tot/float(clm%count)
  call drv_writevar_bin(ftn,clm%toteflx_sh_tot)
!-----------------------------------------------------------------------
! Ground Heat Flux (W/m2)
!-----------------------------------------------------------------------
  clm%toteflx_soil_grnd=clm%toteflx_soil_grnd/float(clm%count)
  call drv_writevar_bin(ftn,clm%toteflx_soil_grnd)
!-----------------------------------------------------------------------
! General Water Balance Components (Time Averaged)
! Snowfall (kg/m2s)
!-----------------------------------------------------------------------
  clm%totsnow = clm%totsnow/float(clm%count)
  call drv_writevar_bin(ftn,clm%totsnow)
!-----------------------------------------------------------------------
! Rainfall (kg/m2s)
!-----------------------------------------------------------------------
  clm%totrain = clm%totrain/float(clm%count)
  call drv_writevar_bin(ftn,clm%totrain)
!-----------------------------------------------------------------------
! Total Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  clm%totqflx_evap = clm%totqflx_evap/float(clm%count)
  call drv_writevar_bin(ftn,clm%totqflx_evap)
!-----------------------------------------------------------------------
! Surface Runoff (kg/m2s)
!-----------------------------------------------------------------------
  clm%totqflx_surf = clm%totqflx_surf/float(clm%count)
  call drv_writevar_bin(ftn,clm%totqflx_surf)
!-----------------------------------------------------------------------
! Subsurface Runoff (kg/m2s)
!-----------------------------------------------------------------------
  clm%totqflx_drain = clm%totqflx_drain/float(clm%count)
  call drv_writevar_bin(ftn,clm%totqflx_drain)
!-----------------------------------------------------------------------
! Snowmelt (kg/m2s)
!-----------------------------------------------------------------------
```

```
  snowmelt=clm%totqflx_snomelt/float(clm%count)
  call drv_writevar_bin(ftn,snowmelt)
!-----------------------------------------------------------------------
! Calculation of total column soil moisture
! Total soil moisture (liquid+ice) in each layer
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     do t=1,lis%d%glbnch
        soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
     enddo
  enddo

  do m=1,nlevsoi
     do t=1,lis%d%glbnch
        soilmtc(t)=soilmtc(t)+soilm(t,m)
     enddo
  enddo
!-----------------------------------------------------------------------
! Change in Soil Moisture (kg/m2)
!-----------------------------------------------------------------------
  delsoilmoist = (soilmtc-clm%soilmtc_prev)/float(clm%count)
  call drv_writevar_bin(ftn,delsoilmoist)
!-----------------------------------------------------------------------
! Change in Snow water equivalent (kg/m2)
!-----------------------------------------------------------------------
  delswe = (clm%h2osno-clm%h2osno_prev)/float(clm%count)
  call drv_writevar_bin(ftn,delswe)
!-----------------------------------------------------------------------
! Surface State Variables
! Average Surface Temperature Calculation
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     snowt(t)=0.
     if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
           snowt(t)=clm(t)%t_soisno(clm(t)%snl+1)
        endif
     endif
     if(snowt(t)==0.)snowt(t)=lis%d%udef
  enddo
!-----------------------------------------------------------------------
! AvgSurfT is the average surface temperature which depends on
! the snow temperature, bare soil temperature and canopy temperature
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     if(snowt(t).ne.lis%d%udef)then
        asurft(t)=clm(t)%frac_sno*snowt(t)+ &
             clm(t)%frac_veg_nosno*clm(t)%t_veg+  &
```

```
             (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
             clm(t)%t_grnd
     else
        asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
             (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
     endif
  enddo

  clm%totqflx_ecanop=clm%totqflx_ecanop/float(clm%count)
  cantrn=(clm%totqflx_tran_veg/float(clm%count))
  bare=(clm%totqflx_evap_grnd/float(clm%count))
  snowevp=(clm%totqflx_sub_snow/float(clm%count))
  potevp=lis%d%udef
!-----------------------------------------------------------------------
! Snow Temperature Calculation
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     snowtemp(t)=0.
     if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
           totaldepth(t)=0.
           do i=clm(t)%snl+1,0    ! Compute total depth of snow layers
              totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
           enddo

           do i=clm(t)%snl+1,0    ! Compute snow temperature
              snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
           enddo
           snowtemp(t)=snowtemp(t)/totaldepth(t)
        endif
        if(snowtemp(t).eq.0)snowtemp(t)=lis%d%udef
     endif
  enddo
!-----------------------------------------------------------------------
! Snow Temperature (K)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,snowtemp)
!-----------------------------------------------------------------------
! Canopy Temperature(K)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%t_veg)
!-----------------------------------------------------------------------
! Bare Soil Surface Temperature(K)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%t_grnd)
!-----------------------------------------------------------------------
! Average Surface Temperature(K)
!-----------------------------------------------------------------------
```

```
  call drv_writevar_bin(ftn,asurft)
!-----------------------------------------------------------------------
! Effective Radiative Surface Temperature (K)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%t_rad)
!-----------------------------------------------------------------------
! Surface Albedo
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%surfalb)
!-----------------------------------------------------------------------
! Snow Water Equivalent (kg/m2)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%h2osno)
!-----------------------------------------------------------------------
! Subsurface State Variables
! Average Layer Soil Temperature (K)
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     call drv_writevar_bin(ftn,clm%t_soisno(m))
  enddo
!-----------------------------------------------------------------------
! Subsurface State Variables
! Average Layer Soil Moisture (kg/m2)
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     do c=1,lis%d%glbnch
        tempvar(c)=soilm(c,m)
     enddo
     call drv_writevar_bin(ftn,tempvar)
  enddo
!-----------------------------------------------------------------------
! Root Zone Soil Moisture (kg/m2)
! Calculation of root zone soil moisture
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     soilmr(t)=0.
     do m=1,nlevsoi
        soilmr(t)=soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
     enddo
  enddo
  call drv_writevar_bin(ftn,soilmr)
!-----------------------------------------------------------------------
! Total Soil Wetness
! Calculation of Total column soil wetness and root zone soil wetness
! soilwtc = (vertically averaged soilm - wilting point)/
!          (vertically averaged layer porosity - wilting point)
! where average soilm is swetint, the wilting point is swetwilt,
! and avgwatsat is average porosity.
```

```
! totaldepth represents the total depth of all of the layers
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     swetint(t)=0.
     swetintr(t)=0.
     totaldepth(t)=0.
     avgwatsat(t)=0.
     do m=1,nlevsoi
        avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
        totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
        swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
        swetintr(t)=swetintr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
     enddo
     avgwatsat(t)=avgwatsat(t)/totaldepth(t)
     swetint(t)=(swetint(t)/denh2o)/totaldepth(t)
     swetintr(t)=(swetintr(t)/denh2o)/totaldepth(t)
     soilwtc(t)=100.*swetint(t)/avgwatsat(t)
  enddo
  call drv_writevar_bin(ftn,soilwtc)
!-----------------------------------------------------------------------
! Evaporation Components
! Vegetation Transpiration (kg/m2s)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,cantrn)
!-----------------------------------------------------------------------
! Canopy Water Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn, clm%totqflx_ecanop/2.501E6)
!-----------------------------------------------------------------------
! Bare Soil Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,bare)
!-----------------------------------------------------------------------
! Canopy Interception (kg/m2)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn, clm%canopint)
!-----------------------------------------------------------------------
! Aerodynamic Conductance (m/s)
!-----------------------------------------------------------------------
  call drv_writevar_bin(ftn,clm%acond)

  if(lis%o%wfor .eq. 1) then
!-----------------------------------------------------------------------
! Forcing Data Write Option On
!-----------------------------------------------------------------------
! Wind (m/s)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v))
```

```
!-----------------------------------------------------------------------
! Rainf (kg/m2s)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_rain)
!-----------------------------------------------------------------------
! Snowf (kg/m2s)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_snow)
!-----------------------------------------------------------------------
! Tair (K)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_t)
!-----------------------------------------------------------------------
! Qair (kg/kg)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_q)
!-----------------------------------------------------------------------
! PSurf (Pa)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_pbot)
!-----------------------------------------------------------------------
! SWdown (W/m2)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_solad(1)*100.0/35.0)
!-----------------------------------------------------------------------
! LWdown (W/m2)
!-----------------------------------------------------------------------
     call drv_writevar_bin(ftn,clm%forc_lwrad)
  end if
```

1.85 Fortran: Module Interface clm2drv_module.F90 (Source File: clm2drv_module.F90)

Module for runtime specific CLM2 variables

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Version
```

INTERFACE:

```
module clm2drv_module
```

ARGUMENTS:

```
type clmdrvdec
   integer :: numoutc2       !Counts number of output times for CLM2
   integer :: clm2open       ! Keeps track of opening files
```

```
      character*40 :: clm2_rfile !CLM2 Active restart file
      character*40 :: clm2_vfile !CLM2 Vegetation Tile Specification File
      character*40 :: clm2_chtfile !CLM2 Canopy Heights File
      real :: clm2_ism          !CLM2 intial soil moisture
      real :: clm2_it           !CLM2 initial soil temperature
      real :: clm2_iscv         !CLM2 Initial snow mass
      real :: writeintc2        !CLM2 Output Interval (hours)
   end type clmdrvdec
```

1.85.1 clm2_dynsetup.F90 (Source File: clm2_dynsetup.F90)

Updates the time dependent CLM variables

REVISION HISTORY:

```
   15 Apr 2002: Sujay Kumar   Initial Specification
```

INTERFACE:

```
 subroutine clm2_dynsetup()
```

USES:

```
   use lisdrv_module, only: lis,tile
   use spmdMod, only : masterproc, npes
   use clm2pardef_module
   use clm2_laitable, only : readlaitable
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endif
     call clm2lairead
!     call readlaitable
#if ( ! defined OPENDAP )
  endif
#if (defined SPMD)
!  call MPI_BCAST(lis%p%laiflag,1,MPI_INTEGER,0, &
!       MPI_COMM_WORLD,ier)
!  call MPI_BCAST(lis%p%saiflag,1,MPI_INTEGER,0, &
!       MPI_COMM_WORLD,ier)
!  if(npes > 1 .and. lis%p%laiflag==1 .and. &
!       lis%p%saiflag ==1 ) then
!<notes>
! For now always do a scatter because clm2lairead temporally interpolates
! lai/sai each time-step
!</notes>
```

```
  if ( npes > 1 ) then
!     call clm2_scatter
     call clm2_scatterlai
  endif
#endif
```

1.85.2 clm2_gather (Source File: clm2_gather.F90)

Gathers CLM tiles

REVISION HISTORY:

```
   30 Jan 2003: Sujay Kumar Initial Specification
```

INTERFACE:

```
 subroutine clm2_gather()
```

USES:

```
   use clm_varder
   use tile_spmdMod
   use clm2pardef_module
```

CONTENTS:

```
#if  (defined SPMD)
  call MPI_GATHERV(clm(1:di_array(iam)),di_array(iam), &
       MPI_CLM_STRUCT,clm,di_array,displs,MPI_CLM_STRUCT, &
       0,MPI_COMM_WORLD, ierr)
#endif
```

1.85.3 clm2_gribout.F90 (Source File: clm2_gribout.F90)

LIS CLM2 data writer: Write CLM2 output in Grib format

REVISION HISTORY:

```
   02 Dec  2003: Sujay Kumar; Initial Version
```

INTERFACE:

```
 subroutine clm2_gribout(ftn)
```

USES:

```
use lis_module
use lisdrv_module, only : lis, gindex
use drv_output_mod, only : drv_writevar_grib
use clm_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, only : nlevsoi
use clm_varmap, only : patchvec
use clm_varctl, only : clmdrv
use clm_varder
use time_manager, only : tick
```

CONTENTS:

```
soilmtc=0.0
delsoilmoist = 0.0
delswe = 0.0
soilmr=0.0
soilwtc=0.0

interval = clmdrv%writeintc2
hr1=lis%t%hr
da1=lis%t%da
mo1=lis%t%mo
yr1=lis%t%yr
mn1=lis%t%mn
ss1=0
ts1=-3600*24
dummygmt=1.0
dummytime=1.0

write(unit=temp1,fmt='(i4,i2,i2)')yr1,mo1,da1
read(unit=temp1,fmt='(8a1)')tod
do i=1,8
   if(tod(i).eq.(' '))tod(i)='0'
enddo
today=tod(1)//tod(2)//tod(3)//tod(4)//tod(5) &
     //tod(6)//tod(7)//tod(8)

call tick(dummytime,doy1,dummygmt,yr1,mo1,da1,hr1,mn1,ss1,ts1)
write(unit=temp1,fmt='(i4,i2,i2)')yr1,mo1,da1
read(unit=temp1,fmt='(8a1)')yes
do i=1,8
   if(yes(i).eq.(' '))yes(i)='0'
enddo
yesterday=yes(1)//yes(2)//yes(3)//yes(4)//yes(5) &
     //yes(6)//yes(7)//yes(8)

do i=1,25
   kpds(i)=0
```

```
  enddo
  kpds(1)=221               !id for gsfc products
  kpds(2)=221               !id for clm2 model (change value for other models)
  kpds(4)=192               !bms flag... don't worry about this.
  kpds(12)=0                !assume output time minute always = 0
  kpds(13)=1                !forecast time unit (hours)
  kpds(17)=int((clmdrv%writeintc2*3600.0)/lis%t%ts) !number of time steps in
  !averaged/accum variables
  kpds(18)=0                !grib version -- left as 0 in ncep products
  kpds(19)=1                !version number of kpds.tbl for lisas.
  kpds(20)=0                !none missing from averages/accumulations (always4)
  kpds(23)=221              !gsfc id#
  kpds(24)=0                !does not apply to lisas output
  kpds(25)=0

  open (unit = 69, file = './src/tables/KPDS_completeclm.tbl')
  do k = 1, 42
     read(69,*)
  end do

  do c=1,lis%d%lnc
     do r=1,lis%d%lnr
        if(gindex(c,r).gt.0.5) then
           lismask(c,r)=.true.
        else
           lismask(c,r)=.false.
        endif
     enddo
  enddo

!-----------------------------------------------------------------------
! Net Surface Shortwave (Absorbed) Radiation (W/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totfsa=clm%totfsa/float(clm%count)
  call drv_writevar_grib(ftn,clm%totfsa,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Net Surface Longwave Radiation (W/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%toteflx_lwrad_net=-1.0*clm%toteflx_lwrad_net/float(clm%count)
  call drv_writevar_grib(ftn,clm%toteflx_lwrad_net,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Latent Heat Flux (W/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%toteflx_lh_tot=clm%toteflx_lh_tot/float(clm%count)
  call drv_writevar_grib(ftn,clm%toteflx_lh_tot,kpds,lismask,interval,today,yesterday)
```

```
!-----------------------------------------------------------------------
! Sensible Heat Flux (W/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%toteflx_sh_tot=clm%toteflx_sh_tot/float(clm%count)
  call drv_writevar_grib(ftn,clm%toteflx_sh_tot,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Ground Heat Flux (W/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%toteflx_soil_grnd=clm%toteflx_soil_grnd/float(clm%count)
  call drv_writevar_grib(ftn,clm%toteflx_soil_grnd,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! General Water Balance Components (Time Averaged)
! Snowfall (kg/m2s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totsnow = clm%totsnow/float(clm%count)
  call drv_writevar_grib(ftn,clm%totsnow,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Rainfall (kg/m2s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totrain = clm%totrain/float(clm%count)
  call drv_writevar_grib(ftn,clm%totrain,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Total Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totqflx_evap = clm%totqflx_evap/float(clm%count)
  call drv_writevar_grib(ftn,clm%totqflx_evap,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Surface Runoff (kg/m2s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totqflx_surf = clm%totqflx_surf/float(clm%count)
!  clm%totqflx_surf = (clm%totqflx_surf/float(clm%count))*interval*3600.0
  call drv_writevar_grib(ftn,clm%totqflx_surf,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Subsurface Runoff (kg/m2s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  clm%totqflx_drain = clm%totqflx_drain/float(clm%count)
!  clm%totqflx_drain = (clm%totqflx_drain/float(clm%count))*interval*3600.0
  call drv_writevar_grib(ftn,clm%totqflx_drain,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Snowmelt (kg/m2s)
!-----------------------------------------------------------------------
```

```
  call readkpdsclm(69,kpds)
  snowmelt=clm%totqflx_snomelt/float(clm%count)
!  snowmelt=(clm%qflx_snomelt/float(clm%count))*interval*3600.0
  call drv_writevar_grib(ftn,snowmelt,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Snowmelt Heat flux(W/m2)
!-----------------------------------------------------------------------
!  call readkpdsclm(69,kpds)
!  snowmeltflux=clm%totqflx_snomelt/float(clm%count)
!  call drv_writevar_grib(ftn,snowmeltflux,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Total soil moisture (liquid+ice) in each layer
! Calculation of total column soil moisture
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     do t=1,lis%d%glbnch
        soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
     enddo
  enddo

  do m=1,nlevsoi
     do t=1,lis%d%glbnch
        soilmtc(t)=soilmtc(t)+soilm(t,m)
     enddo
  enddo
!-----------------------------------------------------------------------
! Change in Soil Moisture (kg/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  delsoilmoist = (soilmtc-clm%soilmtc_prev)/float(clm%count)
  call drv_writevar_grib(ftn,delsoilmoist,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Change in Snow water equivalent (kg/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  delswe = (clm%h2osno-clm%h2osno_prev)/float(clm%count)
  call drv_writevar_grib(ftn,delswe,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Surface State Variables
! Average Surface Temperature Calculation
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     snowt(t)=0.
     if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
           snowt(t)=clm(t)%t_soisno(clm(t)%snl+1)
        endif
     endif
```

```
     if(snowt(t)==0.)snowt(t)=lis%d%udef
  enddo
!-----------------------------------------------------------------------
! AvgSurfT is the average surface temperature which depends on
! the snow temperature, bare soil temperature and canopy temperature
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     if(snowt(t).ne.lis%d%udef)then
        asurft(t)=clm(t)%frac_sno*snowt(t)+ &
             clm(t)%frac_veg_nosno*clm(t)%t_veg+  &
             (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
             clm(t)%t_grnd
     else
        asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
             (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
     endif
  enddo

!-----------------------------------------------------------------------
! Snow Temperature Calculation
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     snowtemp(t)=0.
     if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
           totaldepth(t)=0.
           do i=clm(t)%snl+1,0    ! Compute total depth of snow layers
              totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
           enddo

           do i=clm(t)%snl+1,0    ! Compute snow temperature
              snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
           enddo
           snowtemp(t)=snowtemp(t)/totaldepth(t)
        endif
        if(snowtemp(t).eq.0)snowtemp(t)=lis%d%udef
     endif
  enddo
!-----------------------------------------------------------------------
! Snow Temperature (K)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,snowtemp,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Canopy Temperature(K)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%t_veg,kpds,lismask,interval,today,yesterday)
```

```
!-----------------------------------------------------------------------
! Bare Soil Surface Temperature(K)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%t_grnd,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Average Surface Temperature(K)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,asurft,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Effective Radiative Surface Temperature (K)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%t_rad,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Surface Albedo
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%surfalb,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Snow Water Equivalent (kg/m2)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%h2osno,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Subsurface State Variables
! Average Layer Soil Temperature (K)
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%t_soisno(m),kpds,lismask,interval,today,yesterday)
  enddo
!-----------------------------------------------------------------------
! Average Layer Soil Moisture (kg/m2)
!-----------------------------------------------------------------------
  do m=1,nlevsoi
     do c=1,lis%d%glbnch
        tempvar(c)=soilm(c,m)
     enddo
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,tempvar,kpds,lismask,interval,today,yesterday)
  enddo
!-----------------------------------------------------------------------
! Root Zone Soil Moisture (kg/m2)
! Calculation of root zone soil moisture
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
```

```
     soilmr(t)=0.
     do m=1,nlevsoi
        soilmr(t)=soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
     enddo
  enddo
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,soilmr,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Total Soil Wetness
! Calculation of Total column soil wetness and root zone soil wetness
! soilwtc = (vertically averaged soilm - wilting point)/
!          (vertically averaged layer porosity - wilting point)
! where average soilm is swetint, the wilting point is swetwilt,
! and avgwatsat is average porosity.
! totaldepth represents the total depth of all of the layers
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     swetint(t)=0.
     swetintr(t)=0.
     totaldepth(t)=0.
     avgwatsat(t)=0.
     do m=1,nlevsoi
        avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
        totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
        swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
        swetintr(t)=swetintr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
     enddo
     avgwatsat(t)=avgwatsat(t)/totaldepth(t)
     swetint(t)=(swetint(t)/denh2o)/totaldepth(t)
     swetintr(t)=(swetintr(t)/denh2o)/totaldepth(t)
     soilwtc(t)=100.*swetint(t)/avgwatsat(t)
  enddo
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,soilwtc,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Evaporation Components
! Vegetation Transpiration (kg/m2s)
!-----------------------------------------------------------------------
  cantrn=(clm%totqflx_tran_veg/float(clm%count))
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,cantrn,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Canopy Water Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  clm%totqflx_ecanop=clm%totqflx_ecanop/float(clm%count)
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%totqflx_ecanop/2.501E6,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
```

```
! Potential Evaporation (kg/m2s)
!-----------------------------------------------------------------------
!  potevp=lis%d%udef
!-----------------------------------------------------------------------
! Bare Soil Evaporation (kg/m2s)
!-----------------------------------------------------------------------
  bare=(clm%totqflx_evap_grnd/float(clm%count))
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,bare,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Canopy Interception
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%canopint,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Snowpack Evaporation (kg/m2s)
!-----------------------------------------------------------------------
!  snowevp=(clm%totqflx_sub_snow/float(clm%count))
!  call readkpdsclm(69,kpds)
!  call drv_writevar_grib(ftn,snowevp,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Aerodynamic Conductance (m/s)
!-----------------------------------------------------------------------
  call readkpdsclm(69,kpds)
  call drv_writevar_grib(ftn,clm%acond,kpds,lismask,interval,today,yesterday)

  if(lis%o%wfor .eq. 1) then
!-----------------------------------------------------------------------
! Forcing Data Write Option On
!-----------------------------------------------------------------------
! Wind (m/s)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
                            kpds,lismask,interval,today,yesterday)
!    call readkpdsclm(69,kpds)
!     call drv_writevar_grib(ftn,clm%forc_u,kpds,lismask,interval,today,yesterday)
!    call readkpdsclm(69,kpds)
!     call drv_writevar_grib(ftn,clm%forc_v,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Rainf (kg/m2s) - ** NOTE - CHANGE TO ACPCP-Convective Precip Field
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_rain,kpds,lismask,interval,today,yesterday)
!     clm%totrain = clm%totrain/float(clm%count)
!     call drv_writevar_grib(ftn,clm%totrain,kpds,lismask,interval,today,yesterday)
!     call drv_writevar_grib(ftn,(clm%forc_rain)*interval*3600.0,kpds,lismask,interval,today,ye
!-----------------------------------------------------------------------
```

```
! Snowf (kg/m2s)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_snow,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Tair (K)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_t,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! Qair (kg/kg)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_q,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! PSurf (Pa)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_pbot,kpds,lismask,interval,today,yesterday)
!-----------------------------------------------------------------------
! SWdown (W/m2)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_solad(1)*100.0/35.0,kpds,lismask,interval,today,yeste
!-----------------------------------------------------------------------
! LWdown (W/m2)
!-----------------------------------------------------------------------
     call readkpdsclm(69,kpds)
     call drv_writevar_grib(ftn,clm%forc_lwrad,kpds,lismask,interval,today,yesterday)
  end if
  close(69)
```

1.85.4 clmlairead.F90: (Source File: clm2lairead.F90)

This program reads in AVHRR LAI data for CLM

REVISION HISTORY:

```
27 Nov 2001: Jon Gottschalck; Initial code
20 Feb 2002: Jon Gottschalck; Modified to use for 1/4 and 2x2.5 using 1/8 degree monthly da
01 Oct 2002: Jon Gottschalck; Modified to add MODIS LAI data
```

INTERFACE:

```
subroutine clm2lairead ()
```

USES:

```
use lisdrv_module, only : lis
use clm_varder
```

1.85.5 clm2_output.F90 (Source File: clm2_output.F90)

This subroutines sets up methods to write CLM2 output

INTERFACE:

```
subroutine clm2_output
```

USES:

```
use lisdrv_module, only : lis, tile
use clm_varctl, only : clmdrv
use spmdMod, only : masterproc, npes
```

CONTENTS:

```
  if(lis%o%wsingle==1) then
!-----------------------------------------------------------------
! Write output with each variable in a single file
!-----------------------------------------------------------------
     if(mod(lis%t%gmt,clmdrv%writeintc2).eq.0)then
!        if ( lis%d%domain == 8 ) then ! special 1km regional domain
        if(lis%d%gridDesc(9) .eq. 0.01) then
           call system('/data1/pool/control/traffic-wait.pl')
        endif
        do i=1,11
           call clm2_singlegather(i,var)
           if(masterproc) then
              call clm2_singleout(var, i)
           endif
        enddo
        do i=14,47
           call clm2_singlegather(i, var)
           if(masterproc) then
              call clm2_singleout(var,i)
           endif
        enddo
        if ( lis%o%wfor == 1 ) then
           do i=48,55
              call clm2_singlegather(i, var)
              if(masterproc) then
                 call clm2_singleout(var,i)
              endif
           enddo
```

```
        endif
        if(lis%d%gridDesc(9) .eq. 0.01) then
           call system('/data1/pool/control/traffic-go.pl')
        endif
        call clm2_totinit()
     endif
  else
!-----------------------------------------------------------------
! Write bundled output
!-----------------------------------------------------------------
     if(mod(lis%t%gmt,clmdrv%writeintc2).eq.0)then
        if(npes > 1 ) then
           call clm2_gather()
        endif
        if(masterproc) then
           call clm2_almaout()
        endif
        call clm2_totinit()
     endif
  endif
```

1.86 Fortran: Module Interface clm2pardef_module.F90 (Source File: clm2pardef_module.F9

This module contains routines that defines MPI derived data types for CLM LSM

REVISION HISTORY:

```
06 Oct 2003; Sujay Kumar  Initial Specification
```

INTERFACE:

```
module clm2pardef_module
```

USES:

```
use clmtype
use clm2drv_module
use spmdMod
```

1.86.1 def_clmpar_struct (Source File: clm2pardef_module.F90)

Routine that defines MPI derived data types for CLM

INTERFACE:

```
subroutine def_clmpar_struct()
```

1.86.2 clm2_restart.F90 (Source File: clm2_restart.F90)

This program reads the restart files for CLM

REVISION HISTORY:

```
20 Jan 2003; Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm2_restart()
```

USES:

```
use spmdMod, only : masterproc,npes
use lisdrv_module, only : lis
use restFileMod   , only : restrd
```

CONTENTS:

```
  if(masterproc) then
     if(lis%o%startcode.eq.1) then
        print*, 'Reading restart files..'
        call restrd()
     endif
  endif
#if ( defined SPMD )
  if ( ( lis%o%startcode == 1 ) .and. ( npes >  1 ) ) then
     call clm2_scatter()
  endif
#endif
```

1.86.3 clm2_scatter (Source File: clm2_scatter.F90)

Distributes clm2 tiles to compute nodes

REVISION HISTORY:

```
Apr 2003 ; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine clm2_scatter()
```

USES:

```
use clm_varder
use tile_spmdMod
use clm2pardef_module
```

1.86.4 clm2_scatterlai (Source File: clm2_scatter.F90)

Distributes clm2 lai tiles to compute nodes

REVISION HISTORY:

```
17 Sep 2004 ; James Geiger, Initial Code
```

INTERFACE:

```
subroutine clm2_scatterlai()
```

USES:

```
use clm_varder
use tile_spmdMod
use clm2pardef_module
use lisdrv_module, only : lis
```

1.86.5 clm2_setup.F90 (Source File: clm2_setup.F90)

Completes the CLM2 setup routines.

REVISION HISTORY:

```
20 Jan 2003  Sujay Kumar Initial Specification
```

INTERFACE:

```
subroutine clm2_setup()
```

USES:

```
use lisdrv_module, only: lis, tile
use spmdMod
use time_manager
use clm_varder
use clm_varcon, only: eccen, obliqr, lambm0 , mvelpp
use clm_varctl, only: nsrest
```

CONTENTS:

```
#if ( ! defined OPENDAP )
  if ( masterproc ) then
#endif

! ----------------------------------------------------------------------
! Get current date
! ----------------------------------------------------------------------

     if ( masterproc ) then
```

```
!        call get_curr_date(yr, mon, day, ncsec)
        yr = lis%t%yr
        mon = lis%t%mo
        day = lis%t%da
        ncsec = lis%t%ss
     endif
! ----------------------------------------------------------------------
! If initial run: initialize time-varying data
! If continuation run: end of initialization because time varying
! read in from restart file
! ----------------------------------------------------------------------

     if (nsrest == 0) then
        call canhtset()
        if (masterproc) then
           write (6,*) ('Attempting to initialize time variant variables .....')
        endif

        readini = .false.
        call iniTimeVar (readini, eccen, obliqr, lambm0 , mvelpp, lis, tile)

        if (masterproc) then
           write (6,*) ('Successfully initialized time variant variables')
           write (6,*)
        endif

     endif

! ----------------------------------------------------------------------
! End initialization
! ----------------------------------------------------------------------

     if (masterproc) then
        write (6,*) ('Successfully initialized the land model')
        if (nsrest == 0) then
           write (6,*) 'begin initial run at: '
        else
           write (6,*) 'begin continuation run at:'
        end if
        write (6,*) '   nstep= ',get_nstep(lis%t), &
             ' year= ',yr,' month= ',mon,' day= ',day,' seconds= ',ncsec
        write (6,*)
        write (6,'(72a1)') ("*",i=1,60)
        write (6,*)
     endif
     clm%totfsa=0.              ! solar absorbed solar radiation [W/m2]
     clm%toteflx_lwrad_net=0.   ! net longwave radiation [W/m2]
     clm%toteflx_lh_tot=0.      ! total latent heat flux [W/m2]
```

```
     clm%toteflx_sh_tot=0.      ! total sensible heat flux [W/m2]
     clm%toteflx_soil_grnd=0.   ! ground heat flux [W/m2]
     clm%totqflx_snomelt=0.     ! snowmelt heat flux [W/m2]
     clm%totrain=0.             ! accumulation of rain [mm]
     clm%totsnow=0.             ! accumulation of snow [mm]
     clm%totqflx_evap=0.        ! total evaporation [mm]
     clm%totqflx_surf=0.        ! surface runoff [mm]
     clm%totqflx_drain=0.       ! subsurface runoff [mm]
     clm%totqflx_ecanop=0.      ! interception evaporation [W/m2]
     clm%totqflx_tran_veg=0.
     clm%totqflx_evap_grnd=0.
     clm%totqflx_sub_snow=0.
     clm%count=0
     clm%canopint = 0
     clm%acond=0.
#if ( ! defined OPENDAP )
     endif
     if ( npes > 1 ) then
        call clm2_scatter
     endif
#endif
     return
```

1.86.6 clm2_singlegather.F90 (Source File: clm2_singlegather.F90)

Gather single variable for output

REVISION HISTORY:

```
   Apr 2003 ; Sujay Kumar, Initial Code
```

INTERFACE:

```
 subroutine clm2_singlegather(index, var)
```

USES:

```
   use lisdrv_module, only : lis
   use clm_varcon, ONLY : istwet,denh2o
   use clm_varpar, ONLY : nlevsoi
   use clm_varder
   use tile_spmdMod
   use clm2pardef_module
```

CONTENTS:

```
  if ( index == 1 ) then
     var_temp = clm%totfsa/float(clm%count)
```

```
elseif ( index == 2 ) then
   var_temp = -1.0*clm%toteflx_lwrad_net/float(clm%count)
elseif ( index == 3 ) then
   var_temp = clm%toteflx_lh_tot/float(clm%count)
elseif ( index == 4 ) then
   var_temp = clm%toteflx_sh_tot/float(clm%count)
elseif ( index == 5 ) then
   var_temp = clm%toteflx_soil_grnd/float(clm%count)
elseif ( index == 6 ) then
   var_temp = clm%totsnow/float(clm%count)
elseif ( index == 7 ) then
   var_temp = clm%totrain/float(clm%count)
elseif ( index == 8 ) then
   var_temp = clm%totqflx_evap/float(clm%count)
elseif ( index == 9 ) then
   var_temp = clm%totqflx_surf/float(clm%count)
elseif ( index == 10 ) then
   var_temp = clm%totqflx_drain/float(clm%count)
elseif ( index == 11 ) then
   var_temp = clm%totqflx_snomelt/float(clm%count)
elseif ( index == 14 ) then  !SnowT
   do t=1,di_array(iam)
      snowtemp(t)=0.
      if ( clm(t)%itypwat /= istwet ) then
         if ( clm(t)%snl < 0 ) then
            totaldepth(t)=0.
            do i=clm(t)%snl+1,0    ! Compute total depth of snow layers
               totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
            enddo
            do i=clm(t)%snl+1,0    ! Compute snow temperature
               snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
            enddo
            snowtemp(t)=snowtemp(t)/totaldepth(t)
         endif
         if (snowtemp(t).eq.0)snowtemp(t)=lis%d%udef
      endif
   enddo
   var_temp = snowtemp
elseif ( index == 15 ) then !VegT
   var_temp = clm%t_veg
elseif ( index == 16 ) then !BareSoilT
   var_temp = clm%t_grnd
elseif ( index == 17 ) then !AvgSurfT
   do t=1,di_array(iam)
      snowt(t) = 0.0
      if ( clm(t)%itypwat /= istwet ) then
         if ( clm(t)%snl < 0 ) then
            snowt(t) = clm(t)%t_soisno(clm(t)%snl+1)
```

```
         endif
      endif
      if ( snowt(t) == 0.0 ) then
         snowt(t) = lis%d%udef
      endif

      if ( snowt(t) /= lis%d%udef ) then
         asurft(t)=clm(t)%frac_sno*snowt(t)+ &
              clm(t)%frac_veg_nosno*clm(t)%t_veg+  &
              (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
              clm(t)%t_grnd
      else
         asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
              (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
      endif
   enddo
   var_temp = asurft
elseif ( index == 18 ) then !AvgSurfT
   var_temp = clm%t_rad
elseif ( index == 19 ) then  !Albedo
   var_temp = clm%surfalb
elseif ( index == 20 ) then  !SWE
   var_temp = clm%h2osno
elseif ( index == 21 ) then  !SoilTemp1
   var_temp = clm%t_soisno(1)
elseif ( index == 22 ) then  !SoilTemp2
   var_temp = clm%t_soisno(2)
elseif ( index == 23 ) then  !SoilTemp3
   var_temp = clm%t_soisno(3)
elseif ( index == 24 ) then  !SoilTemp4
   var_temp = clm%t_soisno(4)
elseif ( index == 25 ) then  !SoilTemp5
   var_temp = clm%t_soisno(5)
elseif ( index == 26 ) then  !SoilTemp6
   var_temp = clm%t_soisno(6)
elseif ( index == 27 ) then  !SoilTemp7
   var_temp = clm%t_soisno(7)
elseif ( index == 28 ) then  !SoilTemp8
   var_temp = clm%t_soisno(8)
elseif ( index == 29 ) then  !SoilTemp9
   var_temp = clm%t_soisno(9)
elseif ( index == 30 ) then  !SoilTemp10
   var_temp = clm%t_soisno(10)
elseif ( index == 31 ) then  !SoilMoist1
   var_temp = clm%h2osoi_liq(1)+clm%h2osoi_ice(1)
elseif ( index == 32 ) then  !SoilMoist2
   var_temp = clm%h2osoi_liq(2)+clm%h2osoi_ice(2)
elseif ( index == 33 ) then  !SoilMoist3
```

```
   var_temp = clm%h2osoi_liq(3)+clm%h2osoi_ice(3)
elseif ( index == 34 ) then  !SoilMoist4
   var_temp = clm%h2osoi_liq(4)+clm%h2osoi_ice(4)
elseif ( index == 35 ) then  !SoilMoist5
   var_temp = clm%h2osoi_liq(5)+clm%h2osoi_ice(5)
elseif ( index == 36 ) then  !SoilMoist6
   var_temp = clm%h2osoi_liq(6)+clm%h2osoi_ice(6)
elseif ( index == 37 ) then  !SoilMoist7
   var_temp = clm%h2osoi_liq(7)+clm%h2osoi_ice(7)
elseif ( index == 38 ) then  !SoilMoist8
   var_temp = clm%h2osoi_liq(8)+clm%h2osoi_ice(8)
elseif ( index == 39 ) then  !SoilMoist9
   var_temp = clm%h2osoi_liq(9)+clm%h2osoi_ice(9)
elseif ( index == 40 ) then  !SoilMoist10
   var_temp = clm%h2osoi_liq(10)+clm%h2osoi_ice(10)
elseif ( index == 41 ) then !RootMoist
   do t=1,di_array(iam)
      soilmr(t) = 0.0
      do m=1,nlevsoi
         soilmr(t) = soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
      enddo
   enddo
   var_temp = soilmr
elseif ( index == 42 ) then !Soilwet
   do t=1,di_array(iam)
      swetint(t) = 0.0
      avgwatsat(t) = 0.0
      totaldepth(t) = 0.0
      do m=1,nlevsoi
         avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
         totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
         swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
      enddo
      avgwatsat(t) = avgwatsat(t)/totaldepth(t)
      swetint(t) = (swetint(t)/denh2o)/totaldepth(t)
      var_temp(t) = 100*swetint(t)/avgwatsat(t)
   enddo
elseif ( index == 43 ) then  !TVeg
   var_temp = clm%totqflx_tran_veg/float(clm%count)
elseif ( index == 44 ) then  !ECanop
   var_temp = clm%totqflx_ecanop/float(clm%count)
   var_temp = var_temp / 2.501E6
elseif ( index == 45 ) then  !ESoil
   var_temp = clm%totqflx_evap_grnd/float(clm%count)
elseif ( index == 46 ) then  !Canopint
   var_temp = clm%canopint
elseif ( index == 47 ) then !ACond
   var_temp = clm%acond
```

```
! Forcing
  elseif ( index == 48 ) then !Wind
     var_temp = sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v)
  elseif ( index == 49 ) then !Rainfall rate
     var_temp = clm%forc_rain
  elseif ( index == 50 ) then !Snowfall rate
     var_temp = clm%forc_snow
  elseif ( index == 51 ) then !Air temperature
     var_temp = clm%forc_t
  elseif ( index == 52 ) then !Specific humidity
     var_temp = clm%forc_q
  elseif ( index == 53 ) then !Surface pressure
     var_temp = clm%forc_pbot
  elseif ( index == 54 ) then !Shortwave down
     var_temp = clm%forc_solad(1)*100.0/35.0
  elseif ( index == 55 ) then !Longwave down
     var_temp = clm%forc_lwrad
  endif
#if (defined SPMD)
  call MPI_GATHERV(var_temp(1:di_array(iam)),di_array(iam), &
       MPI_REAL,var,di_array,displs,MPI_REAL, &
       0,MPI_COMM_WORLD, ierr)
#else
   var = var_temp
#endif
```

1.86.7 clm2_singleout.F90 (Source File: clm2_singleout.F90)

Write output file for a single CLM variable

REVISION HISTORY:

```
14 Jun 2002; Sujay Kumar  Initial Specification
```

INTERFACE:

```
subroutine clm2_singleout (var_array, index)
```

USES:

```
use lisdrv_module, only : lis, tile
use clm_varcon, ONLY : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, ONLY : nlevsoi
use clm_varmap, ONLY : patchvec
use clm_varctl, only : clmdrv
use drv_output_mod, only : t2gr
```

CONTENTS:

```
!-----------------------------------------------------------------------
! Test to see if output writing interval has been reached
!-----------------------------------------------------------------------
  if(mod(lis%t%gmt,clmdrv%writeintc2).eq.0)then
!-----------------------------------------------------------------------
! Generate directory structure and file names for CLM Output
!-----------------------------------------------------------------------
     length = len(trim(vname1(index)))
     WRITE(UNIT=temp, FMT='(A12)') VNAME1(index)
     READ(UNIT=temp,FMT='(12A1)') (FVARNAME(I), I=1,length)
     WRITE(unit=temp,fmt='(I4,I2,I2)')lis%T%YR,lis%T%MO,lis%T%DA
     READ(unit=temp,fmt='(8a1)')FTIME
     DO I=1,8
        IF(FTIME(I).EQ.(' '))FTIME(I)='0'
     ENDDO

     WRITE(unit=temp,fmt='(I4)')lis%T%YR
     READ(unit=temp,fmt='(8a1)')FTIMEC
     DO I=1,4
        IF(FTIMEC(I).EQ.(' '))FTIMEC(I)='0'
     ENDDO

#if 0
     WRITE(unit=temp,fmt='(a6,i3,a1)')'/LIS.E',lis%O%EXPCODE,'.'
     READ(unit=temp,fmt='(80a1)') (FNAME(I),I=1,10)
     DO I=1,10
        IF(FNAME(I).EQ.(' '))FNAME(I)='0'
     ENDDO
#endif

!     idisk = mod(index, lis%o%odirn)
     idisk = 1
!     if ( idisk == 0 ) then
!        idisk = lis%o%odirn
!     endif
!     WRITE(unit=temp,fmt='(a40)') lis%O%ODIR_ARRAY(idisk)
     WRITE(unit=temp,fmt='(a40)') lis%O%ODIR
     READ(unit=temp,fmt='(40a1)') (FBASE(I),I=1,40)
     C=0
     DO I=1,40
        IF(FBASE(I).EQ.(' ').AND.C.EQ.0)C=I-1
     ENDDO

     WRITE(unit=temp,fmt='(A4,I3,A6,I4,A1,I4,I2,I2)')'/EXP', &
          lis%O%EXPCODE,'/CLM2/', &
          lis%t%YR,'/',lis%T%YR,lis%T%MO,lis%T%DA
     READ(unit=temp,fmt='(80A1)') (FYRMODIR(I),I=1,26)
     DO I=1,26
```

```
        IF(FYRMODIR(I).EQ.(' '))FYRMODIR(I)='0'
     ENDDO

     WRITE(unit=temp,fmt='(A9)')'mkdir -p '
     READ(unit=temp,fmt='(80A1)')(FMKDIR(I),I=1,9)

     WRITE(unit=temp,fmt='(80A1)')(FMKDIR(I),I=1,9),(FBASE(I),I=1,C), &
          (FYRMODIR(I),I=1,26)
     READ(unit=temp,fmt='(A80)')MKFYRMO

!-----------------------------------------------------------------------
! Make the directories for the CLM2 output files
!-----------------------------------------------------------------------
     call system(mkfyrmo)

!-----------------------------------------------------------------------
! Generate file name for binary output
!-----------------------------------------------------------------------
     if(lis%o%wout.eq.1)then
        write(unit=temp,fmt='(I4,I2,I2,I2)')lis%t%yr, &
             lis%t%mo,lis%t%da,lis%t%hr
        read(unit=temp,fmt='(10A1)')ftimeb
        do i=1,10
           if(ftimeb(i).eq.(' '))then
              ftimeb(i)='0'
           endif
        enddo

        !write(unit=temp,fmt='(A9)')'.CLM2gbin'
        if(lis%d%gridDesc(9) .eq. 0.01) then
           write(unit=temp,fmt='(A5)')'.gd4r'
           read(unit=temp,fmt='(80A1)') (fsubgb(i),i=1,5)
        else
           write(unit=temp,fmt='(A5)')'.ls4r'
           read(unit=temp,fmt='(80A1)') (fsubgb(i),i=1,5)
        endif

#if 0
        write(unit=temp,fmt='(82A1)')(fbase(i),i=1,c), &
             (FYRMODIR(I),I=1,26), &
             (fname(i),i=1,10),(ftimeb(i),i=1,10), &
             (fvarname(i),i=1,length),(fsubgb(i),i=1,9 )
        read(unit=temp,fmt='(A82)')filengb
#endif
        write(unit=temp,fmt='(69A1)')(fbase(i),i=1,c), &
             (FYRMODIR(I),I=1,26), '/', &
             (ftimeb(i),i=1,10), &
             (fvarname(i),i=1,length),(fsubgb(i),i=1,5 )
```

```
        read(unit=temp,fmt='(A69)')filengb
     endif
     if(lis%o%wout.eq.1)then
       clmdrv%numoutc2=clmdrv%numoutc2+1
        if(lis%d%gridDesc(9) .eq. 0.01) then
           allocate(g2tmp(lis%d%lnc,lis%d%lnr))
           g2tmp = lis%d%UDEF
           do i = 1, lis%d%glbnch
             g2tmp(tile(i)%col, tile(i)%row) = var_array(i)
           enddo

!           if ( lis%o%odirn == 1 ) then
              open(58,file=filengb,form='unformatted',access='direct', &
                   recl=lis%d%lnc * lis%d%lnr * 4)
              write(58, rec=1) g2tmp
!           else !*** multi disk output
!              open(58,file=filengb,form='unformatted',access='direct', &
!                   recl=lis%d%lnc*4)
!              do i = 1, lis%d%lnr
!                 nrec = (lis%d%ir-1) * lis%d%lnr * 50 + ( i - 1 ) * 50 + lis%d%ic
!                 write(58,rec=nrec) g2tmp(:,i)
!              enddo
!           endif
           deallocate(g2tmp)
       else                          ! all other domains
        open(57,file=filengb,form='unformatted')
        allocate(gtmp(lis%d%glbngrid))
        call t2gr(var_array,gtmp,lis%d%glbngrid,lis%d%glbnch,tile)
        write(57) gtmp
        deallocate(gtmp)
       endif
!-----------------------------------------------------------------------
! Write statistical output
!-----------------------------------------------------------------------
        if(lis%d%gridDesc(9) .eq. 0.01) then  ! special 1km regional domain
          call lis_log_msg("DBG: clm2_singleout -- domain 8 cannot write "// &
                           "stats file")
       else
        if(clmdrv%clm2open.eq.0)then
           file='CLMstats.dat'
           call openfile(name,lis%o%odir,lis%o%expcode,file)
           if(lis%o%startcode.eq.1)then
              open(60,file=name,form='formatted',status='unknown', &
                   position='append')
           else
              open(60,file=name,form='formatted',status='replace')
           endif
           clmdrv%clm2open=1
```

```
        endif

        write(60,996)'       Statistical Summary of CLM Output for:  ', &
             lis%t%mo,'/',lis%t%da,'/',lis%t%yr,lis%t%hr,':', &
             lis%t%mn,':',lis%t%ss

996     format(a47,i2,a1,i2,a1,i4,1x,i2,a1,i2,a1,i2)
997     format(t26,'Mean',t40,'StDev',t54,'Min',t68,'Max')

        call stats(var_array,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
             vmax)
        write(60,999) vname(index),vmean,vstdev,vmin,vmax
      endif
     endif
  endif
995 format (1x,a10,I1,a9,4f14.3)
999 format (1x,a15,4f14.3)
998 format (1x,a15,4e14.3)
```

1.86.8 clm2_totinit.F90 (Source File: clm2_totinit.F90)

Initialize CLM output arrays

REVISION HISTORY:

```
    14 Jun 2002  Sujay Kumar  Initial Specification
```

INTERFACE:

```
 subroutine clm2_totinit()
```

USES:

```
   use clm_varder
   use tile_spmdMod
```

CONTENTS:

```
  do t = 1, di_array(iam)
     clm(t)%totfsa=0.              ! solar absorbed solar radiation [W/m2]
     clm(t)%toteflx_lwrad_net=0.   ! net longwave radiation [W/m2]
     clm(t)%toteflx_lh_tot=0.      ! total latent heat flux [W/m2]
     clm(t)%toteflx_sh_tot=0.      ! total sensible heat flux [W/m2]
     clm(t)%toteflx_soil_grnd=0.   ! ground heat flux [W/m2]
     clm(t)%totqflx_snomelt=0.     ! snowmelt heat flux [W/m2]
     clm(t)%totrain=0.             ! accumulation of rain [mm]
     clm(t)%totsnow=0.             ! accumulation of snow [mm]
     clm(t)%totqflx_evap=0.        ! total evaporation [mm]
```

```
    clm(t)%totqflx_surf=0.        ! surface runoff [mm]
    clm(t)%totqflx_drain=0.       ! subsurface runoff [mm]
    clm(t)%totqflx_ecanop=0.      ! interception evaporation [W/m2]
    clm(t)%totqflx_tran_veg=0.
    clm(t)%totqflx_evap_grnd=0.
    clm(t)%totqflx_sub_snow=0.
    clm(t)%count=0
  enddo
  soilmtc = 0.0
  do m=1,nlevsoi
     do t=1,di_array(iam)
        soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
     enddo
  enddo
  do m=1,nlevsoi
     do t=1,di_array(iam)
        soilmtc(t)=soilmtc(t)+soilm(t,m)
     enddo
  enddo
  do t=1,di_array(iam)
     clm(t)%soilmtc_prev = soilmtc(t)
     clm(t)%h2osno_prev = clm(t)%h2osno
  enddo
```

1.86.9 clm2_writestats.F90 (Source File: clm2_writestats.F90)

LIS CLM2 data writer: Write CLM2 stats

REVISION HISTORY:

```
02 Dec  2003: Sujay Kumar; Initial Version
```

INTERFACE:

```
subroutine clm2_writestats(ftn_stats)
```

USES:

```
use lisdrv_module, only : lis
use clm_varcon, only : denh2o, denice, hvap, hsub, hfus, istwet
use clm_varpar, only : nlevsoi
use clm_varmap, only : patchvec
use clm_varder
```

CONTENTS:

```
soilmtc=0.0
```

```
delsoilmoist = 0.0
delswe = 0.0
soilmr=0.0
soilwtc=0.0


do m=1,nlevsoi
   do t=1,lis%d%glbnch
      soilm(t,m)=clm(t)%h2osoi_liq(m)+clm(t)%h2osoi_ice(m)
   enddo
enddo

do m=1,nlevsoi
   do t=1,lis%d%glbnch
      soilmtc(t)=soilmtc(t)+soilm(t,m)
   enddo
enddo

delsoilmoist = (soilmtc-clm%soilmtc_prev)/float(clm%count)

delswe = (clm%h2osno-clm%h2osno_prev)/float(clm%count)
do t=1,lis%d%glbnch
   snowt(t)=0.
   if (clm(t)%itypwat/=istwet)then
      if(clm(t)%snl < 0)then
         snowt(t)=clm(t)%t_soisno(clm(t)%snl+1)
      endif
   endif
   if(snowt(t)==0.)snowt(t)=lis%d%udef
enddo

do t=1,lis%d%glbnch
   if(snowt(t).ne.lis%d%udef)then
      asurft(t)=clm(t)%frac_sno*snowt(t)+ &
           clm(t)%frac_veg_nosno*clm(t)%t_veg+  &
           (1-(clm(t)%frac_sno+clm(t)%frac_veg_nosno))* &
           clm(t)%t_grnd
   else
      asurft(t)=clm(t)%frac_veg_nosno*clm(t)%t_veg+ &
           (1-clm(t)%frac_veg_nosno)*clm(t)%t_grnd
   endif
enddo

cantrn=(clm%totqflx_tran_veg/float(clm%count))
bare=(clm%totqflx_evap_grnd/float(clm%count))
snowevp=(clm%totqflx_sub_snow/float(clm%count))
potevp=lis%d%udef
```

```
  do t=1,lis%d%glbnch
     snowtemp(t)=0.
     if (clm(t)%itypwat/=istwet)then
        if(clm(t)%snl < 0)then
           totaldepth(t)=0.
           do i=clm(t)%snl+1,0    ! Compute total depth of snow layers
              totaldepth(t)=totaldepth(t)+clm(t)%dz(i)
           enddo

           do i=clm(t)%snl+1,0    ! Compute snow temperature
              snowtemp(t)=snowtemp(t)+(clm(t)%t_soisno(i)*clm(t)%dz(i))
           enddo
           snowtemp(t)=snowtemp(t)/totaldepth(t)
        endif
        if(snowtemp(t).eq.0)snowtemp(t)=lis%d%udef
     endif
  enddo

  do m=1,nlevsoi
     do c=1,lis%d%glbnch
        tempvar(c)=soilm(c,m)
     enddo
  enddo
!-----------------------------------------------------------------------
! Total Soil Wetness
! Calculation of Total column soil wetness and root zone soil wetness
! soilwtc = (vertically averaged soilm - wilting point)/
!          (vertically averaged layer porosity - wilting point)
! where average soilm is swetint, the wilting point is swetwilt,
! and avgwatsat is average porosity.
! totaldepth represents the total depth of all of the layers
!-----------------------------------------------------------------------
  do t=1,lis%d%glbnch
     swetint(t)=0.
     swetintr(t)=0.
     totaldepth(t)=0.
     avgwatsat(t)=0.
     do m=1,nlevsoi
        avgwatsat(t)=avgwatsat(t)+clm(t)%dz(m)*clm(t)%watsat(m)
        totaldepth(t)=totaldepth(t)+clm(t)%dz(m)
        swetint(t)=swetint(t)+clm(t)%h2osoi_liq(m)
        swetintr(t)=swetintr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
     enddo
     avgwatsat(t)=avgwatsat(t)/totaldepth(t)
     swetint(t)=(swetint(t)/denh2o)/totaldepth(t)
     swetintr(t)=(swetintr(t)/denh2o)/totaldepth(t)
     soilwtc(t)=100.*swetint(t)/avgwatsat(t)
  enddo
```

```
do t=1,lis%d%glbnch
   soilmr(t)=0.
   do m=1,nlevsoi
      soilmr(t)=soilmr(t)+clm(t)%rootfr(m)*clm(t)%h2osoi_liq(m)
   enddo
enddo

call stats(clm%totfsa,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,999)'SWnet(W/m2): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%toteflx_lwrad_net,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'LWnet (W/m2): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%toteflx_lh_tot,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qle (W/m2):    ', &
     vmean,vstdev,vmin,vmax
call stats(clm%toteflx_sh_tot,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qh (W/m2):    ', &
     vmean,vstdev,vmin,vmax
call stats(clm%toteflx_soil_grnd,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'Qg (W/m2): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%totsnow,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,998)'Snowf (kg/m2s): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%totrain,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,998)'Rainf (kg/m2s): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%totqflx_evap,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Evap (kg/m2s): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%totqflx_surf,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qs (kg/m2s): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%totqflx_drain,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qsb (kg/m2s): ', &
     vmean,vstdev,vmin,vmax
```

```
call stats(clm%totqflx_snomelt/2.5e6,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'Qsm (kg/m2s):  ', &
     vmean,vstdev,vmin,vmax
call stats(delsoilmoist,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelSoilMoist (kg/m2):  ', &
     vmean,vstdev,vmin,vmax
call stats(delswe,lis%d%udef,lis%d%glbnch, &
     vmean,vstdev,vmin,vmax)
write(ftn_stats,998)'DelSWE (kg/m2): ', &
     vmean,vstdev,vmin,vmax
call stats(snowtemp,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SnowT (K): ', &
     vmean,vstdev,vmin,vmax
call stats(clm%t_veg,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,999)'VegT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_grnd,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,999)'BaresoilT (K): ',vmean,vstdev,vmin,vmax
call stats(asurft,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'AvgSurfT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%t_rad,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,999)'RadT (K): ',vmean,vstdev,vmin,vmax
call stats(clm%surfalb,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,999)'Albedo (-): ',vmean,vstdev,vmin,vmax
call stats(clm%h2ocan,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
     vmax)
write(ftn_stats,998)'SWE (kg/m2):        ',vmean,vstdev,vmin,vmax

do m=1,nlevsoi
   do c=1,lis%d%glbnch
      tempvar(c)=soilm(c,m)
   enddo
   call stats(tempvar,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
   write(ftn_stats,995)'SoilMoist',m,' (kg/m2): ',vmean,vstdev,vmin,vmax
enddo

call stats(soilwtc,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,999)'SoilWet (%): ',vmean,vstdev,vmin,vmax

call stats(clm%totqflx_ecanop/2.501E6,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
write(ftn_stats,998) 'ECanop(kg/m2s): ',vmean,vstdev,vmin,vmax

call stats(cantrn,lis%d%udef,lis%d%glbnch, &
```

```
      vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'TVeg (kg/m2s):  ', &
      vmean,vstdev,vmin,vmax
  call stats(bare,lis%d%udef,lis%d%glbnch, &
      vmean,vstdev,vmin,vmax)
  write(ftn_stats,998)'ESoil (kg/m2s): ', &
      vmean,vstdev,vmin,vmax
  call stats(soilmr,lis%d%udef,lis%d%glbnch, &
      vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'RootMoist (kg/m2): ', &
      vmean,vstdev,vmin,vmax
  call stats(clm%acond,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
      vmax)
  call stats(clm%canopint,lis%d%udef,lis%d%glbnch, &
      vmean,vstdev,vmin,vmax)
  write(ftn_stats,999)'CanopInt (kg/m2): ', &
      vmean,vstdev,vmin,vmax
  write(ftn_stats,998)'ACond (m/s): ', &
      vmean,vstdev,vmin,vmax
  if(lis%o%wfor.eq.1) then
     call stats(sqrt(clm%forc_u*clm%forc_u+clm%forc_v*clm%forc_v), &
          lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin,vmax)
     write(ftn_stats,999)'Wind(m/s): ', &
          vmean,vstdev,vmin,vmax
     call stats(clm%forc_rain,lis%d%udef,lis%d%glbnch, &
          vmean,vstdev,vmin,vmax)
     write(ftn_stats,998)'Rainf(kg/m2s): ', &
          vmean,vstdev,vmin,vmax
     call stats(clm%forc_snow,lis%d%udef,lis%d%glbnch, &
          vmean,vstdev,vmin,vmax)
     write(ftn_stats,998)'Snowf(kg/m2s): ', &
          vmean,vstdev,vmin,vmax
     call stats(clm%forc_t,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
          vmax)
     write(ftn_stats,999)'Tair(K): ', &
         vmean,vstdev,vmin,vmax
       call stats(clm%forc_q,lis%d%udef,lis%d%glbnch,vmean,vstdev,vmin, &
            vmax)
       write(ftn_stats,999)'Qair(kg/kg): ', &
            vmean,vstdev,vmin,vmax
       call stats(clm%forc_pbot,lis%d%udef,lis%d%glbnch,vmean, &
            vstdev,vmin,vmax)
       write(ftn_stats,999)'PSurf(Pa): ',&
            vmean,vstdev,vmin,vmax
       call stats(clm%forc_solad(1)*100.0/35.0,lis%d%udef,lis%d%glbnch, &
            vmean,vstdev,vmin,vmax)
       write(ftn_stats,999)'SWdown(W/m2): ', &
            vmean,vstdev,vmin,vmax
```

```
       call stats(clm%forc_lwrad,lis%d%udef,lis%d%glbnch,vmean, &
            vstdev,vmin,vmax)
       write(ftn_stats,999)'LWdown(W/m2): ', &
            vmean,vstdev,vmin,vmax
    endif
995 format (1x,a10,I1,a9,4f14.3)
999 format (1x,a15,4f14.3)
998 format (1x,a15,4e14.3)
```

!ROUTINE:clm2wrst.F90
This program writes the restart files for CLM

REVISION HISTORY:

```
20 Jan 2003; Sujay Kumar Initial Specification
26 Aug 2004; James Geiger, Added support for GrADS-DODS based and MPI based
                           parallel simulations.
```

INTERFACE:

```
subroutine clm2wrst()
```

USES:

```
use spmdMod, only : masterproc, npes
use restFileMod, only : restwrt
use clm_varctl, only : clmdrv
use lisdrv_module, only : lis
```

CONTENTS:

```
  if ( ( lis%t%gmt == (24-clmdrv%writeintc2) )  .or. &
       lis%t%endtime == 1 ) then

#if ( ! defined OPENDAP )
     if ( lis%o%wsingle == 1 .and. npes > 1 ) then
        call clm2_gather()
     endif

     if ( masterproc ) then
#endif
        call lis_log_msg('MSG: clm2wrst -- Writing CLM restart')
        call restwrt()
#if ( ! defined OPENDAP )
     endif
#endif
```

```
  endif
```

1.86.10 clm_varder.F90 (Source File: clm_varder.F90)

Module to initialize clm variables

INTERFACE:

```
 module clm_varder
```

USES:

```
   use clmtype
   use spmdMod

   implicit none
   !ARGUMENTS
   type (clm1d), allocatable :: clm(:)
```

1.86.11 clm_varder_ini (Source File: clm_varder.F90)

Reads in runtime clm parameters, allocates memory for variables

INTERFACE:

```
   subroutine clm_varder_ini(nch)
```

USES:

```
     use infnan
     use tile_spmdMod
     use clm_varmap
     use clm_varcon
     use clm_varctl, only : clmdrv
     use shr_orb_mod
     use initializeMod, only : initialize
     use clm2pardef_module
```

CONTENTS:

```
    if(masterproc) then
       call readclm2crd(clmdrv)
    endif
#if (defined SPMD)
    call def_clmpar_struct
    call MPI_BCAST(clmdrv, 1, MPI_CLMDRV_STRUCT, 0, &
         MPI_COMM_WORLD, ier)
```

```
#endif
!-----------------------------------------------------------------------
! allocate memory for clm derived type
!-----------------------------------------------------------------------
    if(masterproc) then
       allocate (clm(1:nch))
        print*,'MSG: clm_varder_ini -- allocating clm',di_array(iam), &
             ' (',iam,')'
        call clm_varder_init(nch)
    else
       begland = 1
       endland = di_array(iam)
       begpatch = 1
       endpatch = di_array(iam)
       print*,'MSG: clm_varder_ini -- allocating clm',di_array(iam), &
              ' (',iam,')'
       allocate(clm(di_array(iam)))
    endif
    if(masterproc) then
     call shr_orb_params
    endif
#if (defined SPMD)
    call MPI_BCAST(numpatch,1,MPI_INTEGER,0,&
            MPI_COMM_WORLD,ier)
    call MPI_BCAST(eccen,1,MPI_REAL,0,&
            MPI_COMM_WORLD,ier)
    call MPI_BCAST(obliqr,1,MPI_REAL,0,&
            MPI_COMM_WORLD,ier)
    call MPI_BCAST(lambm0,1,MPI_REAL,0,&
            MPI_COMM_WORLD,ier)
    call MPI_BCAST(mvelpp,1,MPI_REAL,0,&
            MPI_COMM_WORLD,ier)
#endif
#if ( defined OPENDAP )
    call initialize
#else
    if(masterproc) then
       call initialize
    endif
#endif
```

1.86.12 clm_varder_init (Source File: clm_varder.F90)

Initializes clm variables

INTERFACE:

```
subroutine clm_varder_init(nch)
```

USES:

```
use infnan
use tile_spmdMod
use clm_varmap
```

1.86.13 iniTimeConst.F90 (Source File: iniTimeConst.F90)

Initialize time invariant clm variables
Method:
Author: Gordon Bonan

USES:

```
use precision
use infnan
use clm_varder
use clm_varpar   , only : nlevsoi, nlevlak
use clm_varmap   , only : begpatch, endpatch, patchvec, numpatch
use clm_varcon   , only : istsoil, istice, istdlak, istslak, istwet, spval
use pft_varcon   , only : ncorn, nwheat, roota_par, rootb_par,  &
                          z0mr, displar, dleaf, rhol, rhos, taul, taus, xl, &
                          qe25, vcmx25, mp, c3psn
use clm_varsur   , only : zlak, dzlak, zsoi, dzsoi, zisoi

use clm_varctl   , only : nsrest
use time_manager , only : get_step_size
use shr_const_mod, only : SHR_CONST_PI
use spmdMod      , only : masterproc

use lisdrv_module
use lis_openfileMod
use lis_indices_module
```

CONTENTS:

```
! ---------------------------------------------------------------------
! Initialize local variables for error checking
! ---------------------------------------------------------------------

  sand(:,:) = inf
  clay(:,:) = inf

! ---------------------------------------------------------------------
! itypveg, isoicol, itypwat, sand, and clay from 2-d surface type
! LIS modification: Read in soil files and equate lat,lon
! LIS grid arrays with clm grid
```

```
! Reynolds soil parameterization scheme
! Read in soil data maps to gridded arrays
! ---------------------------------------------------------------------

     print*, 'MSG: iniTimeConst -- Reading soil, clay and color files'
     print*, 'MSG: iniTimeConst -- Opening soil file ',lis%p%safile
     call readsand(lis%d%soil, sand1)
     call readclay(lis%d%soil, clay1)
     call read_color(color)

     pi = SHR_CONST_PI

     print*, 'MSG: iniTimeConst -- Read soil, clay and color files'
     do k = 1,numpatch
        if (tile(k)%fgrd > 0.0) then     !valid subgrid patch
           if(lis%d%gridDesc(9) .eq. 0.01) then
              line1 = nint((lis%d%gridDesc(4)-lis%d%gridDesc(44))/0.01)+1
              line2 = nint((lis%d%gridDesc(5)-lis%d%gridDesc(45))/0.01)+1
              glnc = line2+tile(k)%col-1
              glnr = line1+tile(k)%row-1
              lat = -59.995+(glnr-1)*0.01
              lon = -179.995+(glnc-1)*0.01
              clm(k)%lat    = lat * pi/180.
              clm(k)%lon    = lon * pi/180.
           endif
           clm(k)%kpatch = k
           clm(k)%itypveg = tile(k)%vegt
           clm(k)%isoicol = int(color(tile(k)%col,tile(k)%row-lis_tnroffset))
!           clm(k)%isoicol = color(tile(k)%col,tile(k)%row)
!           clm(k)%isoicol = 2
           if (tile(k)%vegt == npatch_urban) then !urban, from pcturb
              clm(k)%itypwat = istsoil
              print*, 'ERR: iniTimeConst -- not supposed to be in urban block'
           else if (tile(k)%vegt == npatch_lake) then  !deep lake, from pctlak
              clm(k)%itypwat = istdlak
              do l = 1,nlevsoi
                 sand(l,k) = 0._r8
                 clay(l,k) = 0._r8
              end do
           else if (tile(k)%vegt == npatch_wet) then   !wetland, from pctwet
              clm(k)%itypwat = istwet
              do l = 1,nlevsoi
                 sand(l,k) = 0._r8
                 clay(l,k) = 0._r8
              end do
           else if (tile(k)%vegt == npatch_gla) then   !glacier, from pctgla
              clm(k)%itypwat = istice
              print*, 'ERR: iniTimeConst -- not supposed to be in glacier block'
```

```
           else                              !soil
              clm(k)%itypwat = istsoil
              do l = 1, nlevsoi
!                 sand(l,k)=sand1(k)
!                 clay(l,k)=clay1(k)
                 sand(l,k)=sand1(tile(k)%col,tile(k)%row-lis_tnroffset)
                 clay(l,k)=clay1(tile(k)%col,tile(k)%row-lis_tnroffset)
!                 if(sand(l,k).lt.0) then sand(l,k) = 0.0
                 if(clay(l,k).lt.0) print*, 'missing clay',k, tile(k)%col, tile(k)%row, grid(k
                 if(sand(l,k).lt.0) print*, 'missing sand..',k, tile(k)%col, tile(k)%row,grid(
              enddo
           end if
        end if
     enddo
! ---------------------------------------------------------------------
! tag lake points
! ---------------------------------------------------------------------

  do k = 1,numpatch
     if (clm(k)%itypwat==istdlak .or. clm(k)%itypwat==istslak) then
        clm(k)%lakpoi = .true.
     else
        clm(k)%lakpoi = .false.
     end if
  end do

! ---------------------------------------------------------------------
! latitudes and longitudes
! ---------------------------------------------------------------------
  if(lis%d%gridDesc(9) .ne. 0.01) then
     pi = SHR_CONST_PI
     do k=1,numpatch
        clm(k)%lat = grid(tile(k)%index-lis_grid_offset)%lat *pi /180
        clm(k)%lon = grid(tile(k)%index-lis_grid_offset)%lon *pi /180
     enddo
  endif
! ---------------------------------------------------------------------
! Define layer structure for soil and lakes
! Vertical profile of snow is initialized in routine iniTimeVar
! ---------------------------------------------------------------------

  if (nlevlak /= nlevsoi) then
     write(6,*)'number of soil levels and number of lake levels must be the same'
     write(6,*)'nlevsoi= ',nlevsoi,' nlevlak= ',nlevlak
     call endrun
  endif

  dzlak(1) = 1.
```

```
dzlak(2) = 2.
dzlak(3) = 3.
dzlak(4) = 4.
dzlak(5) = 5.
dzlak(6) = 7.
dzlak(7) = 7.
dzlak(8) = 7.
dzlak(9) = 7.
dzlak(10)= 7.

zlak(1) =  0.5
zlak(2) =  1.5
zlak(3) =  4.5
zlak(4) =  8.0
zlak(5) = 12.5
zlak(6) = 18.5
zlak(7) = 25.5
zlak(8) = 32.5
zlak(9) = 39.5
zlak(10)= 46.5

do j = 1, nlevsoi
   zsoi(j) = scalez*(exp(0.5*(j-0.5))-1.)    !node depths
enddo

dzsoi(1) = 0.5*(zsoi(1)+zsoi(2))             !thickness b/n two interfaces
do j = 2,nlevsoi-1
   dzsoi(j)= 0.5*(zsoi(j+1)-zsoi(j-1))
enddo
dzsoi(nlevsoi) = zsoi(nlevsoi)-zsoi(nlevsoi-1)

zisoi(0) = 0.
do j = 1, nlevsoi-1
   zisoi(j) = 0.5*(zsoi(j)+zsoi(j+1))         !interface depths
enddo
zisoi(nlevsoi) = zsoi(nlevsoi) + 0.5*dzsoi(nlevsoi)

do k = 1,numpatch
   if (clm(k)%itypwat == istdlak) then        !assume all lakes are deep lakes
      clm(k)%z(1:nlevlak)  = zlak(1:nlevlak)
      clm(k)%dz(1:nlevlak) = dzlak(1:nlevlak)
   else if (clm(k)%itypwat == istslak) then   !shallow lake (not used)
      clm(k)%dz(1:nlevlak) = NaN
      clm(k)%z(1:nlevlak)  = NaN
   else                                       !soil, ice, wetland
      clm(k)%z(1:nlevsoi)  = zsoi(1:nlevsoi)
      clm(k)%dz(1:nlevsoi) = dzsoi(1:nlevsoi)
      clm(k)%zi(0:nlevsoi) = zisoi(0:nlevsoi)
```

```
     endif
  end do

! ---------------------------------------------------------------------
! Initialize root fraction (computing from surface, d is depth in meter):
! Y = 1 -1/2 (exp(-ad)+exp(-bd) under the constraint that
! Y(d =0.1m) = 1-beta^(10 cm) and Y(d=d_obs)=0.99 with beta & d_obs
! given in Zeng et al. (1998).
! ---------------------------------------------------------------------
!  do k = begpatch, endpatch
  do k = 1,numpatch
     if (.not. clm(k)%lakpoi) then
        ivt = clm(k)%itypveg
        do j = 1, nlevsoi-1
           clm(k)%rootfr(j) = .5*( exp(-roota_par(ivt)*clm(k)%zi(j-1))  &
                                 + exp(-rootb_par(ivt)*clm(k)%zi(j-1))  &
                                 - exp(-roota_par(ivt)*clm(k)%zi(j  ))  &
                                 - exp(-rootb_par(ivt)*clm(k)%zi(j  )) )
        end do
        clm(k)%rootfr(nlevsoi) = .5*( exp(-roota_par(ivt)*clm(k)%zi(nlevsoi-1))  &
                                    + exp(-rootb_par(ivt)*clm(k)%zi(nlevsoi-1)) )

     else
        clm(k)%rootfr(1:nlevsoi) = spval
     end if
  end do

! ---------------------------------------------------------------------
! Initialize soil thermal and hydraulic properties
! ---------------------------------------------------------------------
  do k = 1,numpatch
     if (clm(k)%itypwat == istsoil) then ! .and. &
!          clm(k)%itypveg.ne.13) then                !soil
        do j = 1, nlevsoi
           clm(k)%bsw(j)    = 2.91 + 0.159*clay(j,k)*100.0
           clm(k)%watsat(j) = 0.489 - 0.00126*sand(j,k)*100.0
           xksat            = 0.0070556 *( 10.**(-0.884+0.0153*sand(j,k)*100.0) ) ! mm/s
           clm(k)%hksat(j)  = xksat * exp(-clm(k)%zi(j)/hkdepth)
           clm(k)%sucsat(j) = 10. * ( 10.**(1.88-0.0131*sand(j,k)*100.0) )
           tkm              = (8.80*sand(j,k)*100.0+2.92*clay(j,k)*100.0)/(sand(j,k)*100.0+clay
           bd               = (1.-clm(k)%watsat(j))*2.7e3
           clm(k)%tkmg(j)   = tkm ** (1.- clm(k)%watsat(j))
           clm(k)%tksatu(j) = clm(k)%tkmg(j)*0.57**clm(k)%watsat(j)
           clm(k)%tkdry(j)  = (0.135*bd + 64.7) / (2.7e3 - 0.947*bd)
           clm(k)%csol(j)   = (2.128*sand(j,k)*100.0+2.385*clay(j,k)*100.0)/ (sand(j,k)*100.0+c
!           print*, 'watsat..',k,clm(k)%watsat(j)
!           print*, 'hksat..',k,clm(k)%hksat(j)
!           print*,'tksatu..',k,clm(k)%tksatu(j)
```

```
!           print*,'tkdry..',k,clm(k)%tkdry(j)
!           print*,'csol ..',k,clm(k)%csol(j)
        end do
     elseif(clm(k)%itypveg.eq.13) then
        do j = 1, nlevsoi
           clm(k)%csol(j) = 1.940E6
           clm(k)%tkdry(j) = 1.28
           clm(k)%tkmg(j) = 1.28
           clm(k)%tksatu(j)  =1.28
           clm(k)%hksat(j) = 1e-10
           clm(k)%watsat(j) = 0.001
        enddo
     else                                                !ice, lakes, wetlands
        do j = 1, nlevsoi
           clm(k)%bsw(j)    =  spval
           clm(k)%watsat(j) =  spval
           clm(k)%hksat(j)  =  spval
           clm(k)%sucsat(j) =  spval
           clm(k)%tkmg(j)   =  spval
           clm(k)%tksatu(j) =  spval
           clm(k)%tkdry(j)  =  spval
           clm(k)%csol(j)   =  spval
        end do
     end if
  end do

! ---------------------------------------------------------------------
! Initialize clm derived type components from pft_varcon to avoid
! indirect addressing and be compatible with offline CLM code
! ---------------------------------------------------------------------

!  do k = begpatch, endpatch
  do k = 1,numpatch
     ivt = clm(k)%itypveg
     clm(k)%z0mr    = z0mr(ivt)
     clm(k)%displar = displar(ivt)
     clm(k)%dleaf  = dleaf(ivt)
     clm(k)%xl     = xl(ivt)
     do ib = 1,numrad
        clm(k)%rhol(ib) = rhol(ivt,ib)
        clm(k)%rhos(ib) = rhos(ivt,ib)
        clm(k)%taul(ib) = taul(ivt,ib)
        clm(k)%taus(ib) = taus(ivt,ib)
     end do
     clm(k)%qe25   = qe25(ivt)      ! quantum efficiency at 25c (umol co2 / umol photon)
     clm(k)%vcmx25 = vcmx25(ivt)    ! maximum rate of carboxylation at 25c (umol co2/m**2/s)
     clm(k)%mp     = mp(ivt)        ! slope for conductance-to-photosynthesis relationship
     clm(k)%c3psn  = c3psn(ivt)     ! photosynthetic pathway: 0. = c4, 1. = c3
```

```
  end do

!Initialize other misc derived type components - note that for a
!restart run, dtime is set in routine restrd()

  if (nsrest == 0) then
!     do k = begpatch, endpatch
     if ( masterproc ) then
        tmp_dtime = get_step_size(lis%t)
     endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
     call MPI_BCAST(tmp_dtime,1,MPI_REAL,0,MPI_COMM_WORLD,ierr)
#endif
     do k = 1,numpatch
        !clm(k)%dtime = get_step_size()
        clm(k)%dtime = tmp_dtime
     end do
  endif

  print*, "DBG: iniTimeConst -- dtime", clm(1)%dtime,' (',iam,')'

  if (masterproc) then
     write(6,*)
     write(6,30)
     do j = 1,nlevlak
        write(6,40)zlak(j),dzlak(j)
     end do
     write(6,*)
     write(6,35)
     do j = 1,nlevsoi
        write(6,45)zsoi(j),dzsoi(j),zisoi(j)
     end do
     write(6,50)
     write(6,*)
  endif

30 format(' ',' lake levels ',' lake thickness(m)')
35 format(' ',' soil levels ',' soil thickness(m)',' soil interfaces(m)')
40 format(' ',2(f7.3,8x))
45 format(' ',3(f7.3,8x))
50 format(' ','Note: top level soil interface is set to 0')

  return
```

1.86.14 iniTimeVar.F90 (Source File: iniTimeVar.F90)

Initialize 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

Note - h2osoi_vol is needed by clm_soilalb -this is not needed on restart since it is computed before the soil albedo computation is called
Note - remaining variables are initialized by calls to ecosystem dynamics and albedo subroutines.
Method: Initial data is saved to instantaneous initial data files for each of the [maxpatch] subgrid patches for each of the [numland] land points. If a subgrid patch is not active (e.g., 3 patches rather than [maxpatch]), the inactive subgrid patches have data values for the first subgrid patch. This way, as long as the land mask DOES NOT change among runs (i.e., [numland] is the same), an initial data file can be used in numerous experiments even if the surface types (and hence [numpatch]) differ
Author: Mariana Vertenstein

INTERFACE:

```
subroutine iniTimeVar (readini, eccen, obliqr, lambm0 , mvelpp, lis, tile)
```

USES:

```
  use lis_module
  use tile_module
  use precision
  use clm_varder
  use clm_varctl, only : clmdrv
  use clm_varmap  , only : numpatch
  use clm_varcon  , only : bdsno, istice, istwet, istsoil, denice, denh2o, tfrz, spval, doalb
  use inicFileMod , only : type_inidat, inicrd, histrd
  use shr_sys_mod , only : shr_sys_abort
  use spmdMod     , only : masterproc
  use time_manager, only : get_nstep, get_curr_calday
#if (defined SPMD)
  use mpishorthand, only : mpicom, mpichar
#endif
```

CONTENTS:

```
 if (readini) then
    if ( masterproc ) write (6,*) 'Reading initial data '
    call type_inidat(initype)
    if (trim(initype) == 'INICFILE') then
       call inicrd ()
    else if (trim(initype) == 'HISTFILE') then
       call histrd ()
    else
```

```
         call shr_sys_abort('initial data type is limited to INIC or HIST file only')
      endif
      do k = 1,numpatch
         do j = 1,nlevsoi
            clm(k)%h2osoi_vol(j) = clm(k)%h2osoi_liq(j)/(clm(k)%dz(j)*denh2o) &
                                 + clm(k)%h2osoi_ice(j)/(clm(k)%dz(j)*denice)
         end do
      end do
   else
      if ( masterproc ) write (6,*) 'Setting initial data to non-spun up values'

! ========================================================================
! Set snow water
! ========================================================================

! NOTE: h2ocan, h2osno, snowdp and snowage has valid values everywhere

      do k = 1,numpatch
         if (lis%o%startcode == 4) then
            clm(k)%h2ocan  = 0.
            clm(k)%snowage = 0.
         else
            clm(k)%h2ocan = 0.
            if (clm(k)%itypwat == istice) then
               clm(k)%h2osno = 1000.
            else
               clm(k)%h2osno = clmdrv%clm2_iscv
            endif
            clm(k)%snowdp  = clm(k)%h2osno/bdsno
            clm(k)%snowage = 0.
         endif
      end do

! ========================================================================
! Set snow layer number, depth and thickiness
! ========================================================================
      call snowdp2lev ()

! ========================================================================
! Set snow/soil temperature
! ========================================================================

! NOTE:
! t$_-$soisno only has valid values over non-lake
! t$_-$lake   only has valid values over lake
! t$_-$grnd has valid values over all land
! t$_-$veg  has valid values over all land
```

```
do k =1,numpatch
   clm(k)%t_soisno(-nlevsno+1:nlevsoi) = 0
   if (lis%o%startcode == 4) then
      clm(k)%t_veg   = clm(k)%forc_t
   else
      clm(k)%t_veg = clmdrv%clm2_it
   endif
   if (.not. clm(k)%lakpoi) then  !not lake
      clm(k)%t_soisno(-nlevsno+1:0) = spval
      if (clm(k)%snl < 0) then    !snow layer temperatures
         do i = clm(k)%snl+1, 0
            if (lis%o%startcode == 4) then
               if (clm(k)%forc_t  < 273.16) then
                  clm(k)%t_soisno(i) = clm(k)%forc_t
               else
                  clm(k)%t_soisno(i) = 273.16 - 1.
               endif
            else
               if (clmdrv%clm2_it  < 273.16) then
                  clm(k)%t_soisno(i) = clmdrv%clm2_it
               else
                  clm(k)%t_soisno(i) = 273.16 - 1.
               endif
            endif
         enddo
      endif
      do i = 1, nlevsoi
         if (lis%o%startcode == 4) then
            if (clm(k)%itypwat == istice) then
               clm(k)%t_soisno(i) = clm(k)%forc_t
            else if (clm(k)%itypwat ==istwet) then
               clm(k)%t_soisno(i) = clm(k)%forc_t
            else
               clm(k)%t_soisno(i) = clm(k)%forc_t
            endif
         else
            if (clm(k)%itypwat == istice) then
               clm(k)%t_soisno(i) = clmdrv%clm2_it
            else if (clm(k)%itypwat == istwet) then
               clm(k)%t_soisno(i) = clmdrv%clm2_it
            else
               clm(k)%t_soisno(i) = clmdrv%clm2_it
            endif
         endif
      enddo
      clm(k)%t_grnd = clm(k)%t_soisno(clm(k)%snl+1)
   else                           !lake
      if (lis%o%startcode == 4) then
```

```
            clm(k)%t_grnd = clm(k)%forc_t
         else
            clm(k)%t_grnd = clmdrv%clm2_it
         endif
      endif
   end do

! ========================================================================
! Set snow/soil ice and liquid mass
! ========================================================================

! volumetric water is set first and liquid content and ice lens are
! then obtained
! NOTE: h2osoi$_-$vol, h2osoi$_-$liq and h2osoi$_-$ice only have valid values
! over soil
   do k = 1,numpatch
      clm(k)%h2osoi_vol(         1:nlevsoi) = spval
      clm(k)%h2osoi_liq(-nlevsno+1:nlevsoi) = spval
      clm(k)%h2osoi_ice(-nlevsno+1:nlevsoi) = spval

      if (.not. clm(k)%lakpoi) then  !not lake
         if (clm(k)%snl < 0) then    !snow
            do i = clm(k)%snl+1, 0
               clm(k)%h2osoi_ice(i) = clm(k)%dz(i)*250.
               clm(k)%h2osoi_liq(i) = 0.
            enddo
         endif
         do i = 1, nlevsoi           !soil layers
            if (clm(k)%t_soisno(i) <= tfrz) then
               if (lis%o%startcode == 4) then
            else
               clm(k)%h2osoi_ice(i) = clm(k)%dz(i)* &
                    clmdrv%clm2_ism*clm(k)%watsat(i)*denice
            endif
            clm(k)%h2osoi_liq(i) = 0.
            if (clm(k)%itypwat==istwet .or. clm(k)%itypwat==istice) &
            clm(k)%h2osoi_ice(i)=clm(k)%dz(i)*denice
         else
            if (lis%o%startcode == 4) then
               print*, 'Not supposed to be called..'
             else
                clm(k)%h2osoi_liq(i) = clm(k)%dz(i)* &
                     clmdrv%clm2_ism*clm(k)%watsat(i)*denh2o
             endif
             clm(k)%h2osoi_ice(i) = 0.
             if (clm(k)%itypwat==istwet .or. clm(k)%itypwat==istice) &
             clm(k)%h2osoi_liq(i)=clm(k)%dz(i)*denh2o
          endif
```

```
         enddo

         do i = 1,nlevsoi
            if (clm(k)%itypwat == istsoil) then
               clm(k)%h2osoi_vol(i) = 0.3_r8
               clm(k)%h2osoi_vol(i) = clm(k)%h2osoi_liq(i)/&
                    (clm(k)%dz(i)*denh2o) &
                    + clm(k)%h2osoi_ice(i)/(clm(k)%dz(i)*denice)
            else
               clm(k)%h2osoi_vol(i) = 1.0_r8
            endif
              clm(k)%h2osoi_vol(i) = min(clm(k)%h2osoi_vol(i),clm(k)%watsat(i))
           end do
        endif
     end do
  end if  ! end of arbitrary initialization if-block

! ========================================================================
! Remaining variables are initialized by calls to ecosystem dynamics and
! albedo subroutines.
! Note: elai, esai, frac_veg_nosno are computed in Ecosysdyn and needed
! by Fwet and SurfaceAlbedo
! Note: fwet is needed in routine clm_twostream (called by clm_surfalb)
! ========================================================================

  if ( masterproc ) then
     calday = get_curr_calday(lis%t)
  endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
  call MPI_BCAST(calday,1,MPI_REAL,0,MPI_COMM_WORLD,ier)
#endif
  doalb = .true.

    print*, 'DBG: iniTimeVar -- calling clm2lairead',' (',iam,')'
    call clm2lairead

#if (defined DGVM)
  call iniTimeConstDGVM()
#endif

  if ( masterproc ) then
     tmp_nstep = get_nstep(lis%t)
  endif
#if ( ( defined OPENDAP ) && ( defined SPMD ) )
  call MPI_BCAST(tmp_nstep,1,MPI_INTEGER,0,MPI_COMM_WORLD,ier)
#endif
  do k = 1,numpatch
     clm(k)%nstep = tmp_nstep
```

```
    call EcosystemDyn (clm(k), doalb, .false.)
    clm(k)%frac_sno = clm(k)%snowdp/(0.1 + clm(k)%snowdp)
    clm(k)%frac_veg_nosno = clm(k)%frac_veg_nosno_alb
    if(clm(k)%itypveg.eq.12) then
       clm(k)%frac_veg_nosno = 0
    endif
    call Fwet(clm(k))
    call SurfaceAlbedo (clm(k), calday, eccen, obliqr, lambm0, mvelpp)
  end do
  return
```

end subroutine iniTimeVar
!————————————————————————————————————- ! NASA Goddard Space Flight Center Land Information System (LIS) V3.0 ! Released May 2004 ! ! See SOFTWARE DISTRIBUTION POLICY for software distribution policies ! ! The LIS source code and documentation are in the public domain, ! available without fee for educational, research, non-commercial and ! commercial purposes. Users may distribute the binary or source ! code to third parties provided this statement appears on all copies and ! that no charge is made for such copies. ! ! NASA GSFC MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THE ! SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED AS IS WITHOUT EXPRESS OR ! IMPLIED WARRANTY. NEITHER NASA GSFC NOR THE US GOVERNMENT SHALL BE ! LIABLE FOR ANY DAMAGES SUFFERED BY THE USER OF THIS SOFTWARE. ! ! See COPYRIGHT.TXT for copyright details. ! !————————————————————————————————————-

---

!ROUTINE : readclm2crd.F90
Routine to read CLM specific parameters from the card file.

REVISION HISTORY:

```
14 Oct 2003; Sujay Kumar, Initial Code
```

INTERFACE:

```
subroutine readclm2crd(clmdrv)
```

USES:

```
use clm2drv_module
```

CONTENTS:

```
open(11,file='lis.crd',form='formatted',status='old')
read(unit=11,NML=clm2)
print*,'Running CLM2 LSM:'
print*,'CLM2 Active Restart File: ', clmdrv%CLM2_RFILE
clmdrv%clm2open=0
close(11)
```

1.86.15 readkpdsclm.F90 (Source File: readkpdsclm.F90)

Reads the kpds array from the grib table

REVISION HISTORY:

```
24 Oct 2003; Sujay Kumar; Initial Version
```

INTERFACE:

```
subroutine readkpdsclm(ftn, kpds)
```

INTERFACE:

```
use clm_varctl, only : clmdrv
```